Elec 4309 Senior Design

Wendell H Chun

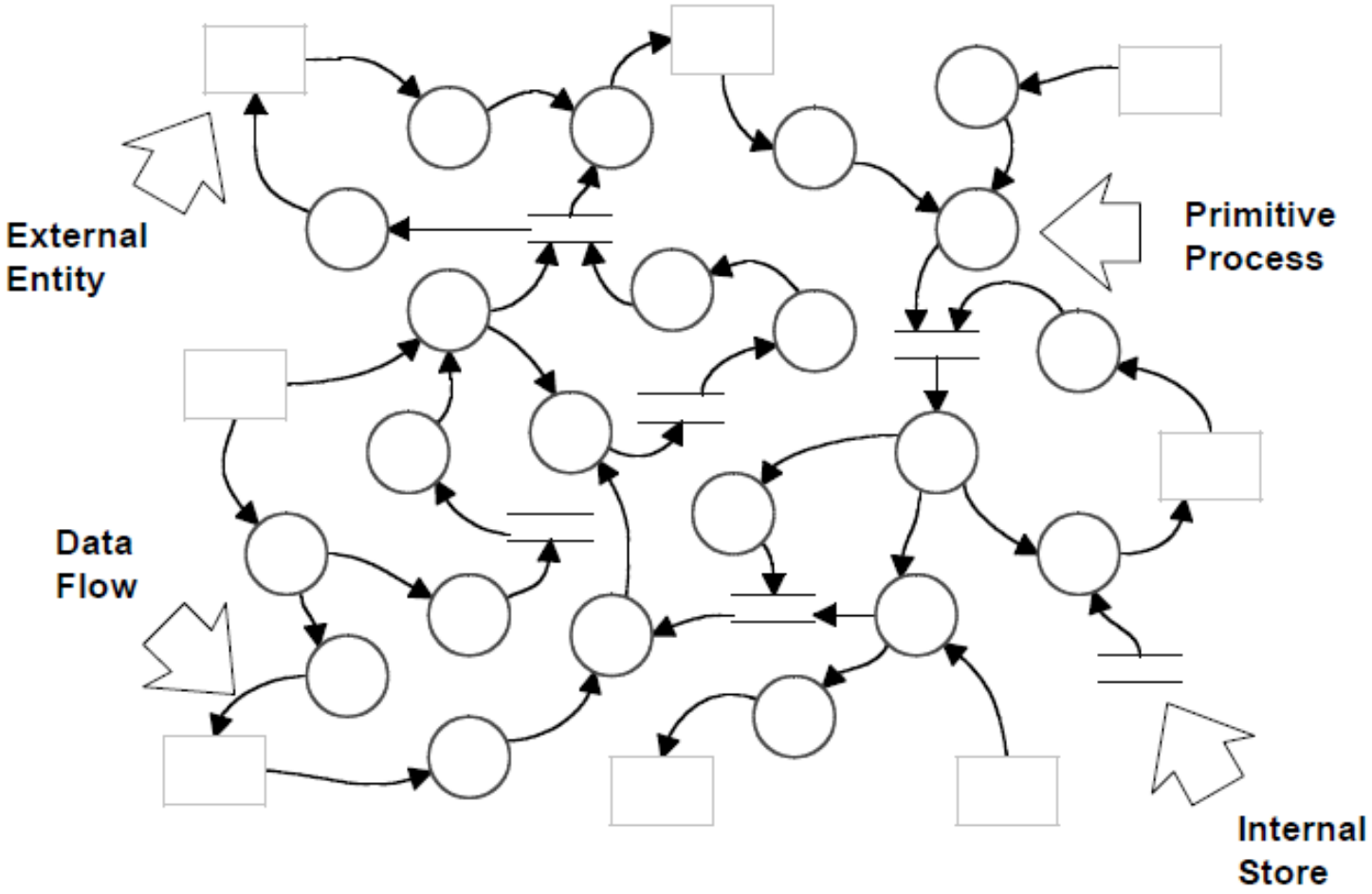
Oct. 17, 2017

The DeMarco Model

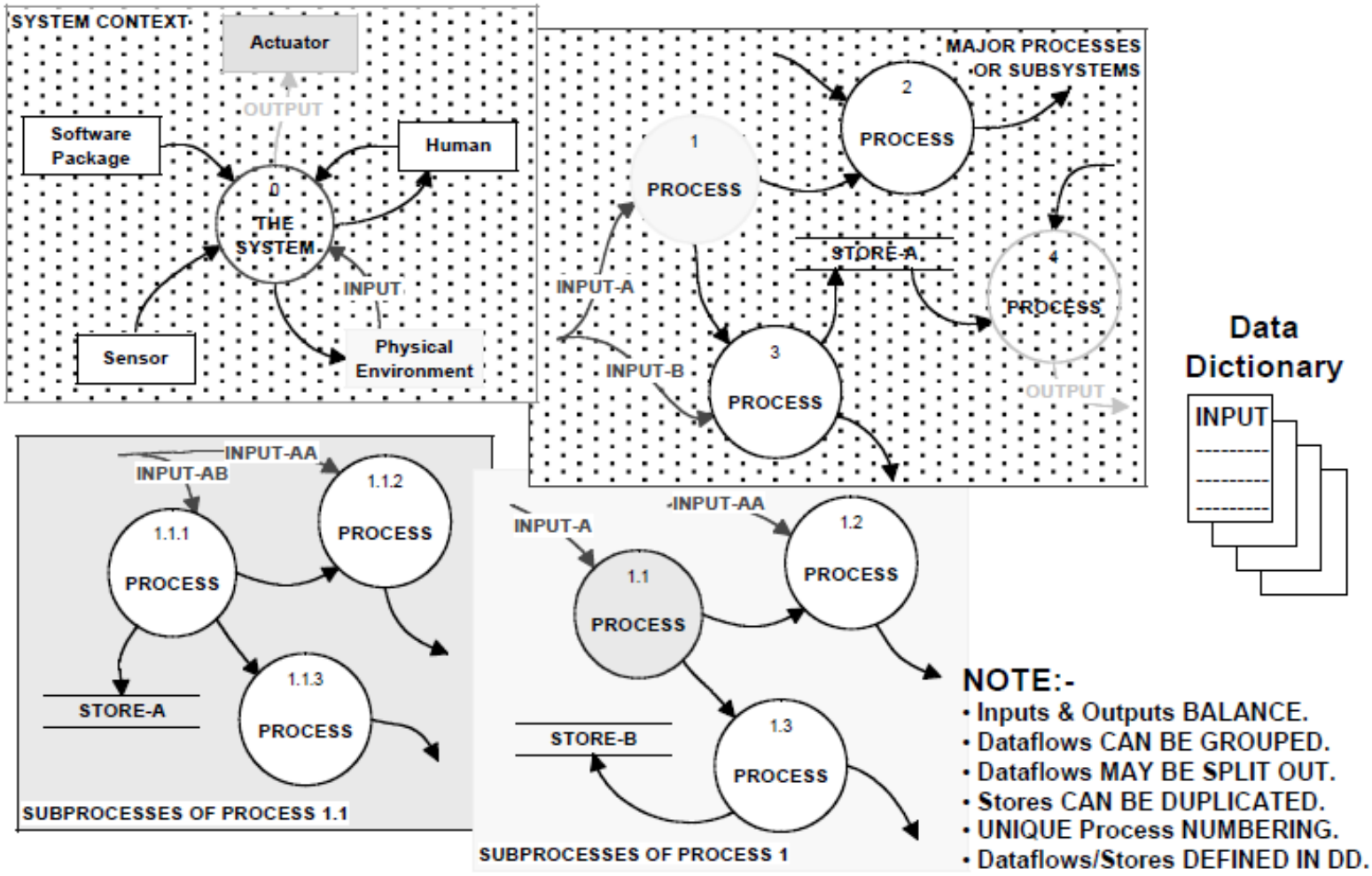
Basic Characteristics:-

- **Semi-formal framework in which to construct a set of Requirements Specifications describing what functional processing a System must do.**
- **A logical model that ignores implementation issues.**
  - » **Logical models are ideal and thus do not represent the eventual system structure.**
- **Assumes perfect technology:**
  - » **(Data) input triggered with instantaneous response - *no timing*.**
  - » **No control issues - *sequentiality or concurrency is not determined*.**
  - » **No storage limitations.**
- **Can be viewed as a large network of primitive (single-purpose) processes communicating via data flows, but is more conveniently represented as an abstracted hierarchy of functional processes.**

A Primitive Model



Dataflow Diagram Decomposition

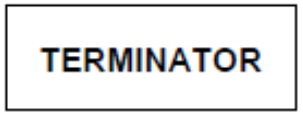


DFD Elements

Processes should be named with a short action clause summarising *what* is to be done to the *input* (data) in order to produce the *output* (data).

- **Dataflows indicate the content and direction of flow of information (or materials) to and from processes, stores and terminators.**
- **Treat them as pipelines along which single or groups of data/material items of known content and nature can flow.**
- **Their names reflect their content - nouns or adjectives.**
- **They *do not* contain or represent dynamic behaviour - no verbal names.**

- **Stores represent dataflows that are frozen for an indeterminate time.**
- **The information/materials they represent can be accessed at any time and in any order.**
- **Nouns and/or adjectives should be used - sometimes plural.**



Terminators represent things that are external to the system, but which are important because they provide &/or receive system input and output.

What is Shown in a Set of DFDs for a System

- **System scope/environment - Context Diagram.**
- **Processes representing combinations of human, mechanical or software functionality.**
- **Processes named according to what must be done to input(s) in order to produce corresponding output(s).**
- **Functional abstraction. All subprocesses must contribute to the overall functionality of the major process from which they are decomposed.**
- **Dataflows representing tangible and/or non-tangible inputs/outputs that a process must receive/produce.**

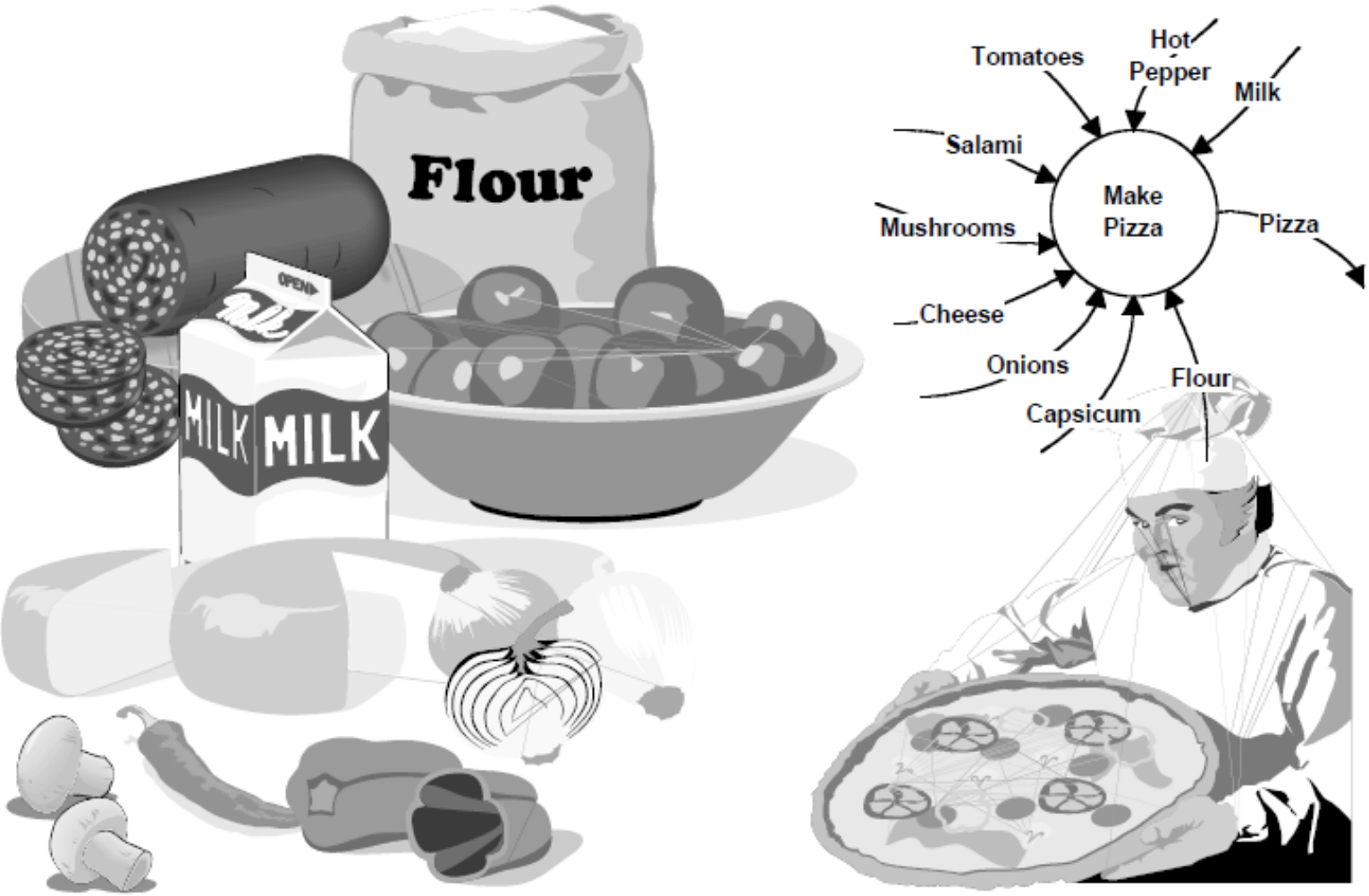
What is Shown in a Set of DFDs for a System

- **Stores representing anything that may be considered as a temporary repository.**
- **A readable summary of what is supposed to be done.**
- **Dataflows & Stores which are fully defined in the Data Dictionary.**
- **Process and dataflow type that is largely determined by the scope of the problem to be analysed.**
- **Terminators - usually shown only at the context level, but may be also shown at other levels for convenience.**

What is Shown in a Set of DFDs for a System

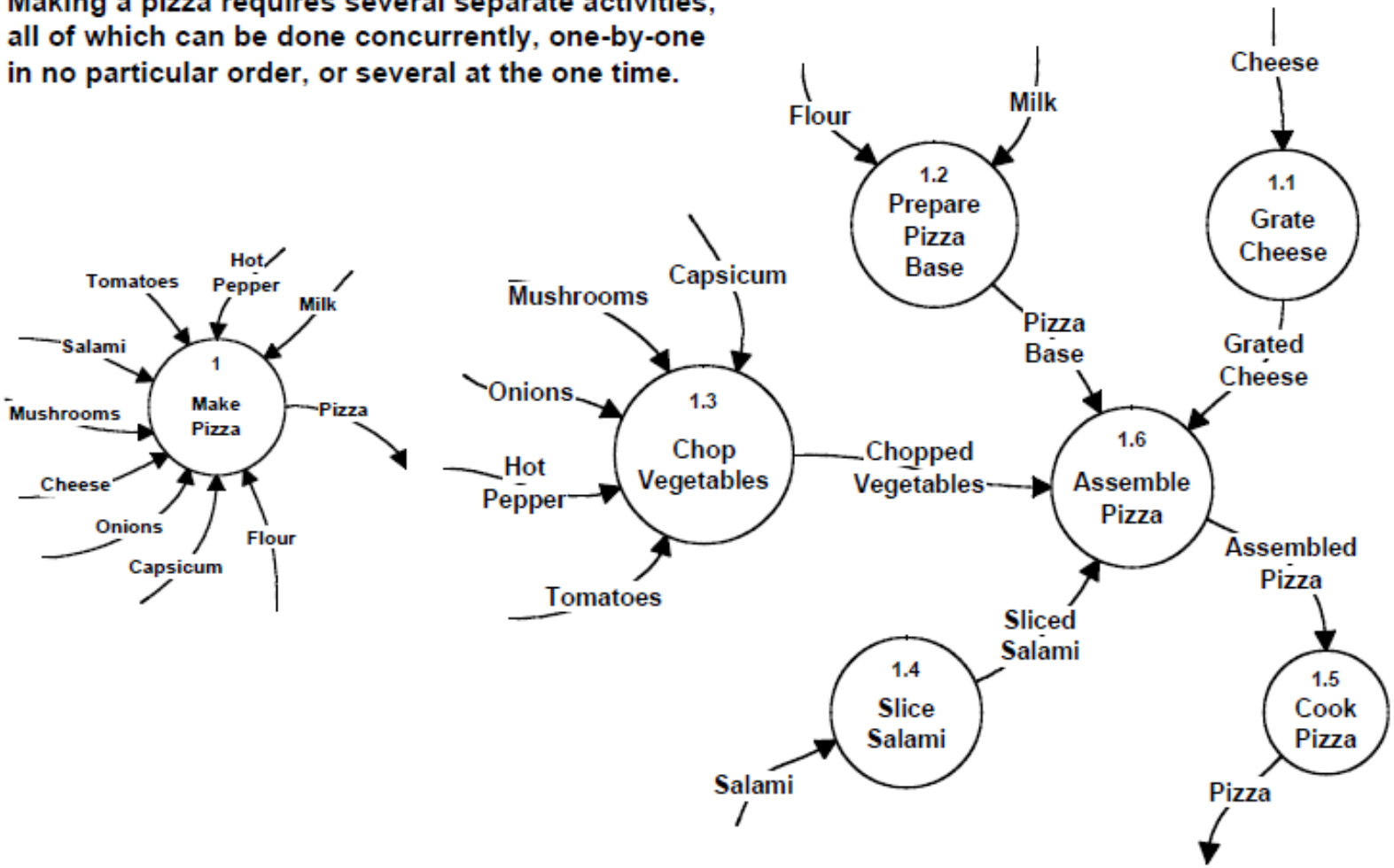
- **Deferring of issues about initialisation and termination - assume steady-state operation of the system.**
- **Omission of processing of trivial errors.**

Making Pizza



Making Pizza in More Detail

Making a pizza requires several separate activities, all of which can be done concurrently, one-by-one in no particular order, or several at the one time.

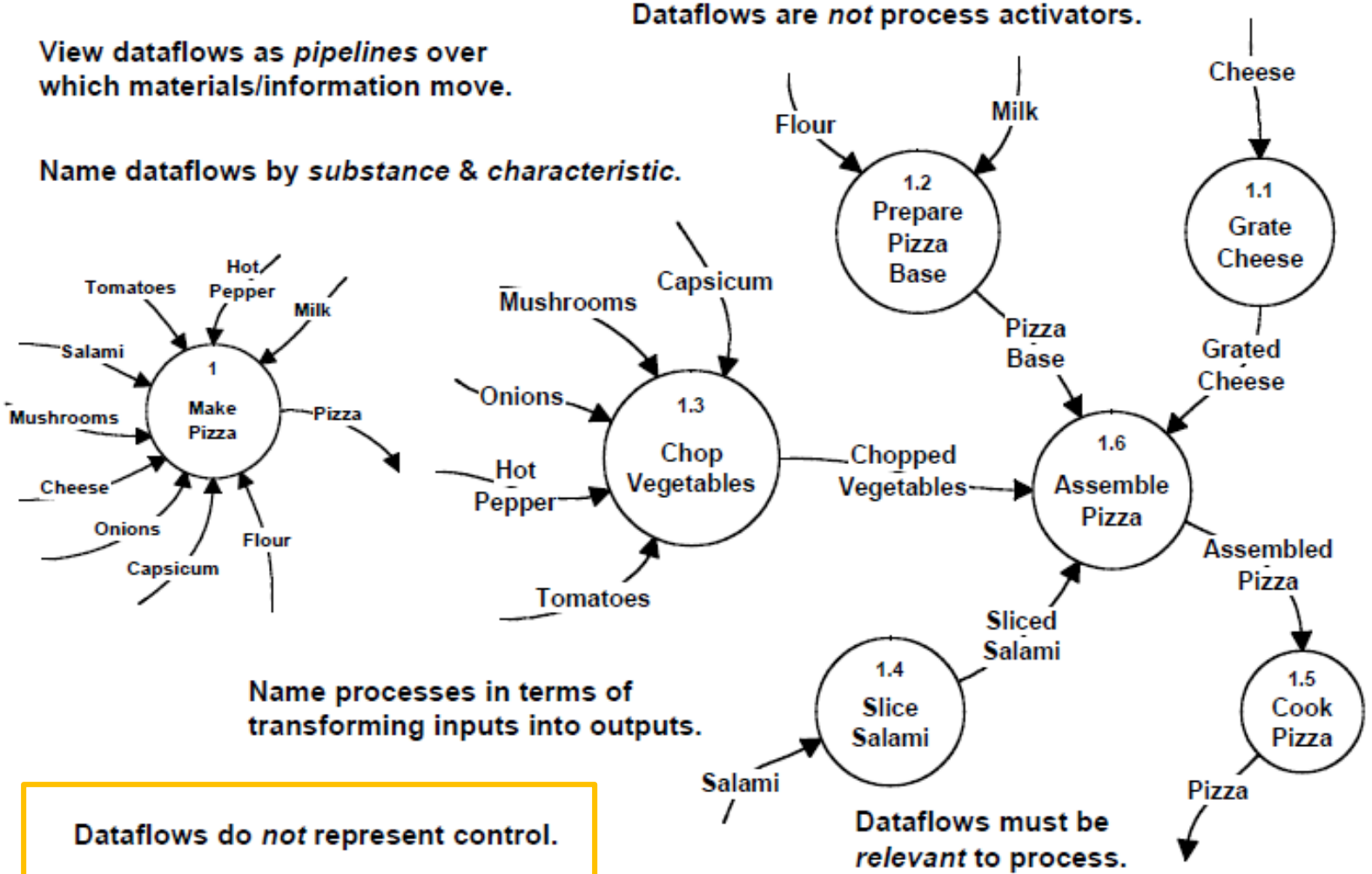


Characteristics of DFDs

Dataflows are *not* process activators.

View dataflows as *pipelines* over which materials/information move.

Name dataflows by *substance* & *characteristic*.



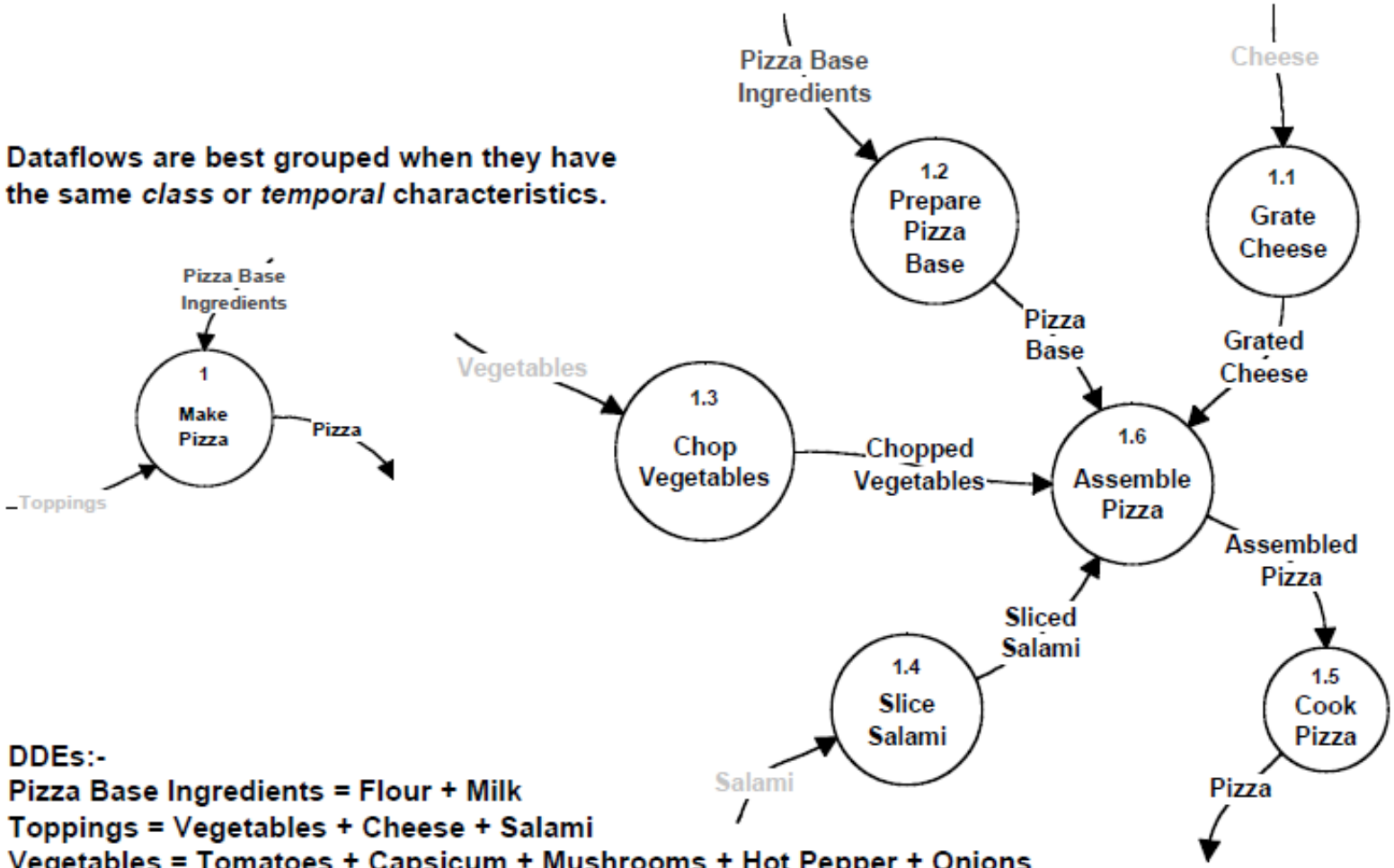
Name processes in terms of transforming inputs into outputs.

Dataflows do *not* represent control.

Dataflows must be *relevant* to process.

Dataflow Grouping

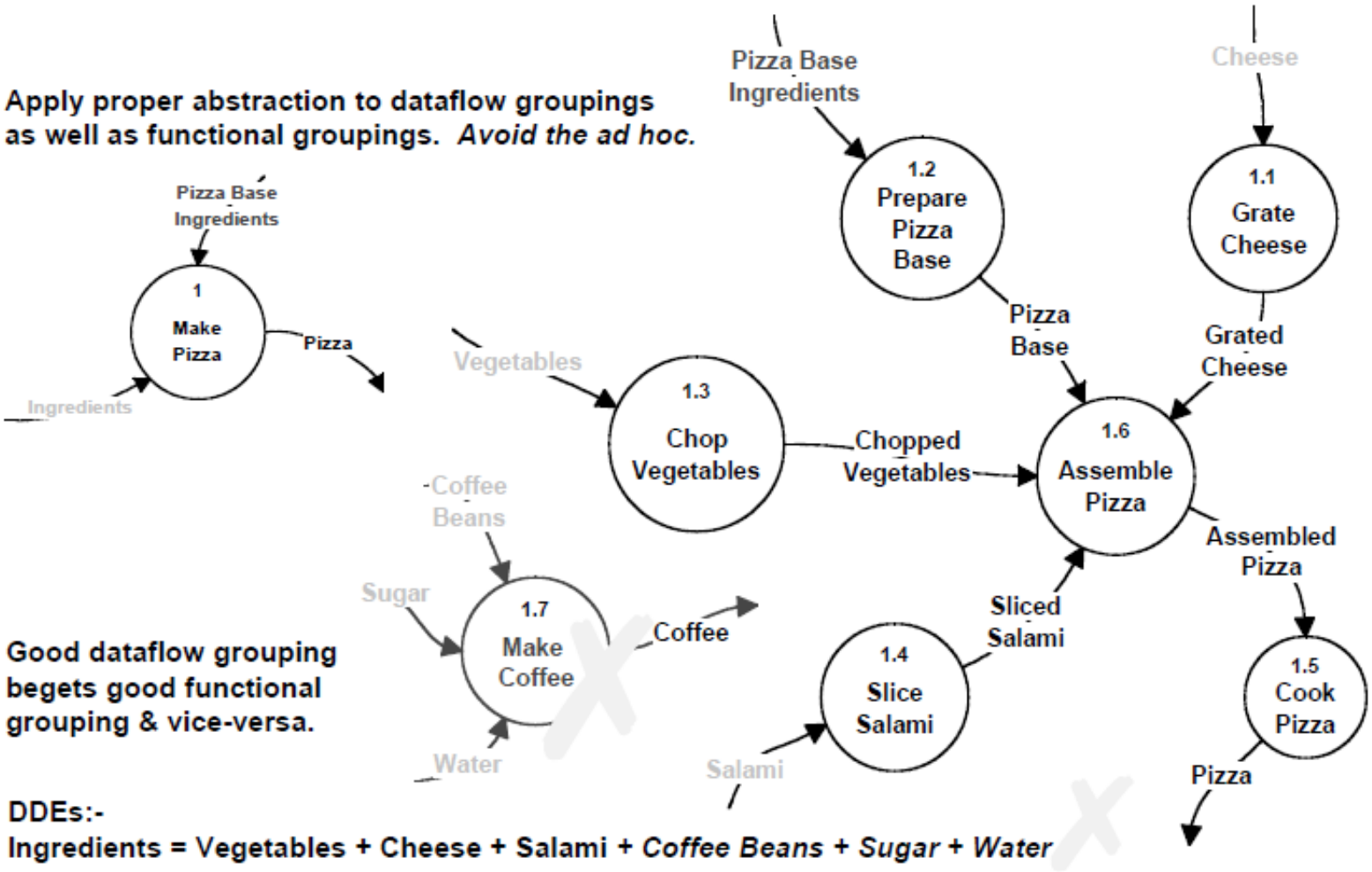
Dataflows are best grouped when they have the same *class* or *temporal* characteristics.



DDEs:-
Pizza Base Ingredients = Flour + Milk
Toppings = Vegetables + Cheese + Salami
Vegetables = Tomatoes + Capsicum + Mushrooms + Hot Pepper + Onions

Dataflow & Process Abstraction

Apply proper abstraction to dataflow groupings as well as functional groupings. *Avoid the ad hoc.*

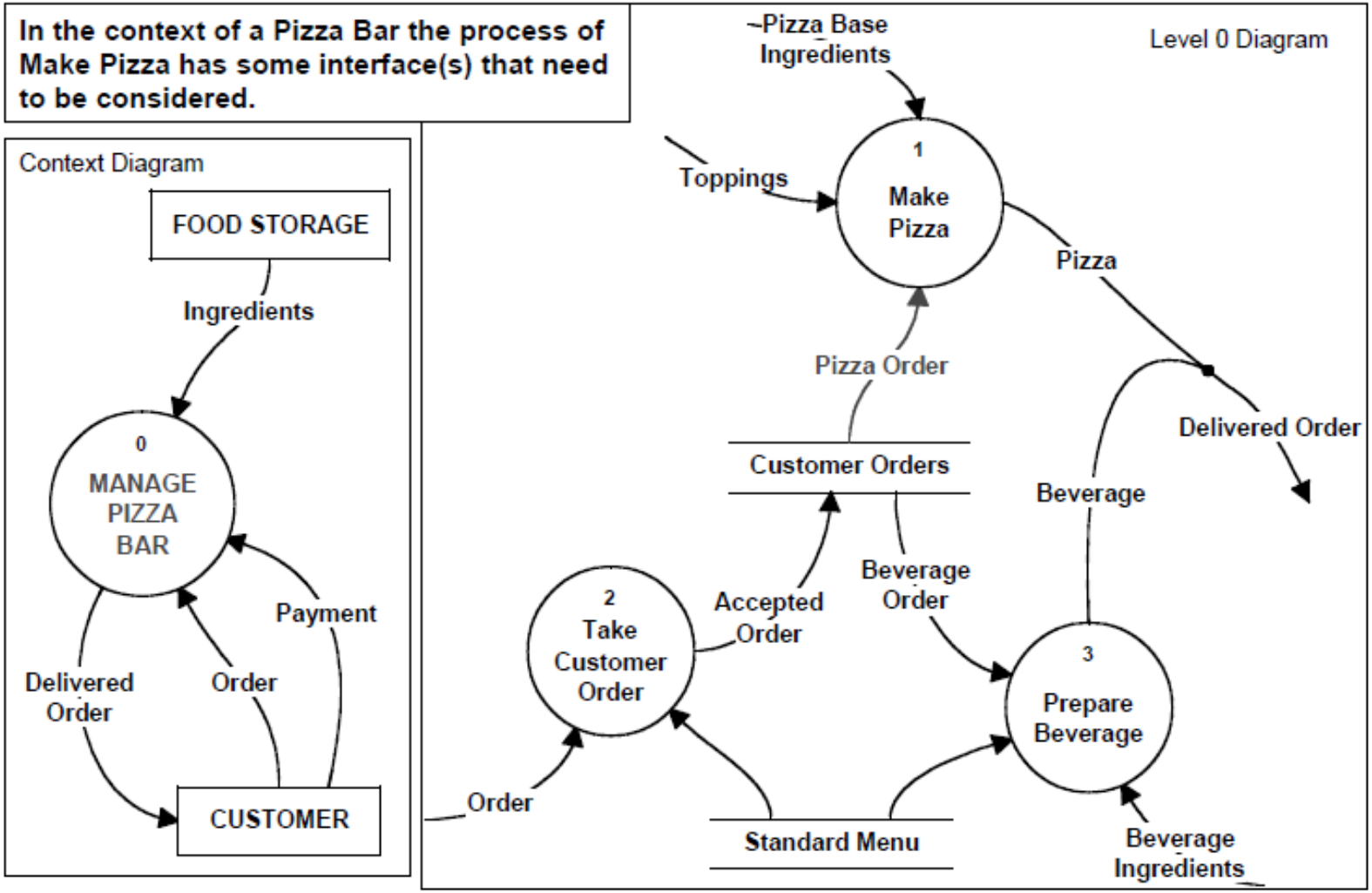


Good dataflow grouping begets good functional grouping & vice-versa.

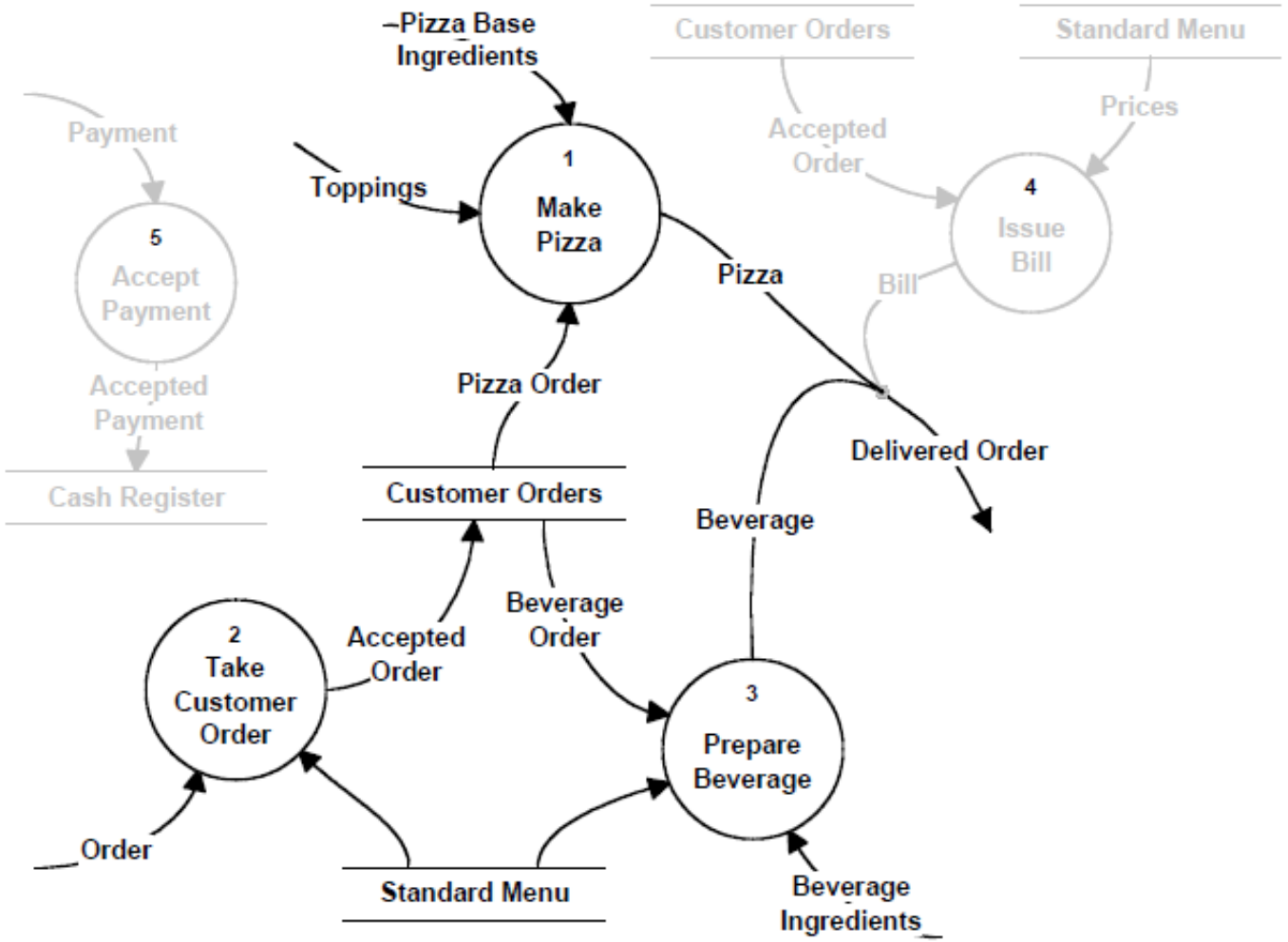
DDEs:-

Ingredients = Vegetables + Cheese + Salami + *Coffee Beans* + *Sugar* + *Water*

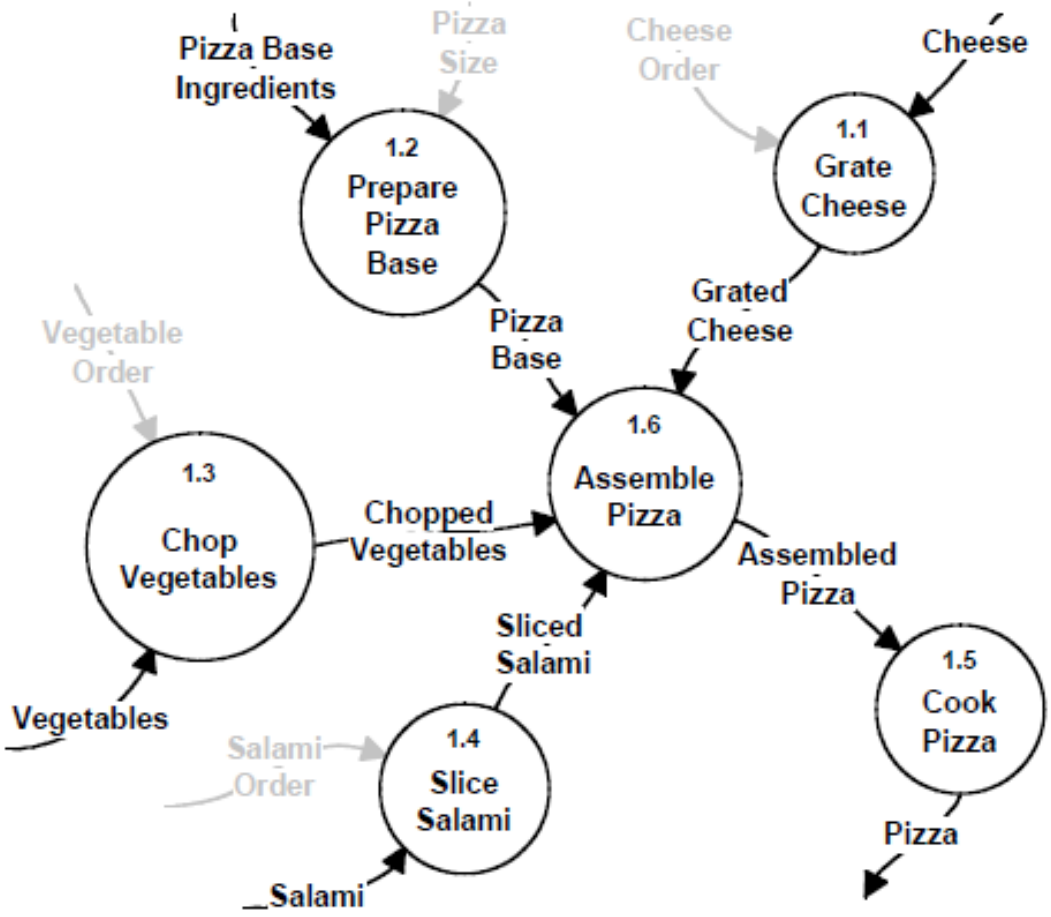
Process Context



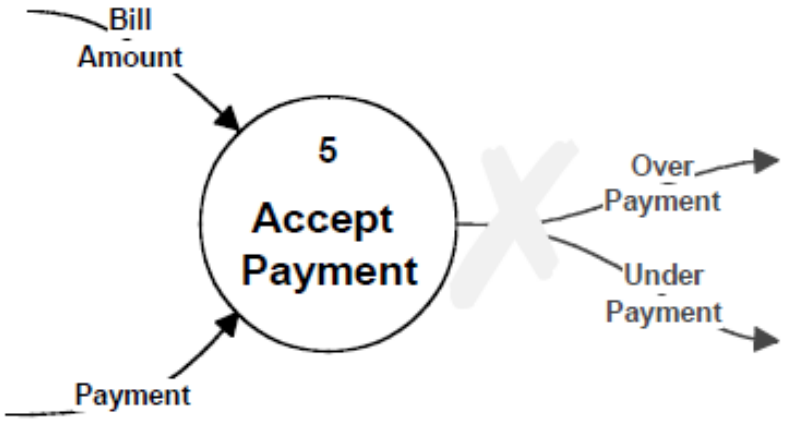
A Proposed Answer for DFD 0



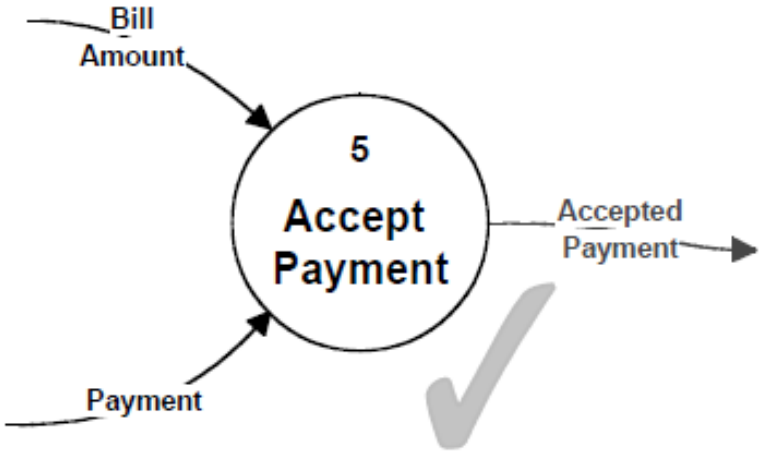
A Proposed Answer for DFD 1



Do's and Don'ts of DFDs



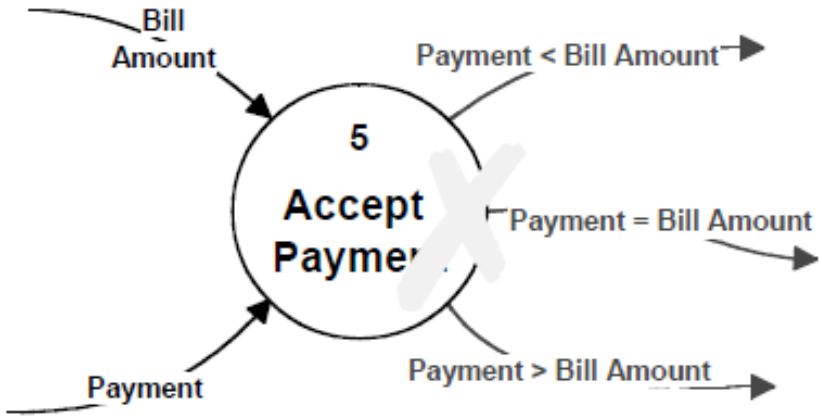
Don't show possible outcomes of a process as split data flows.



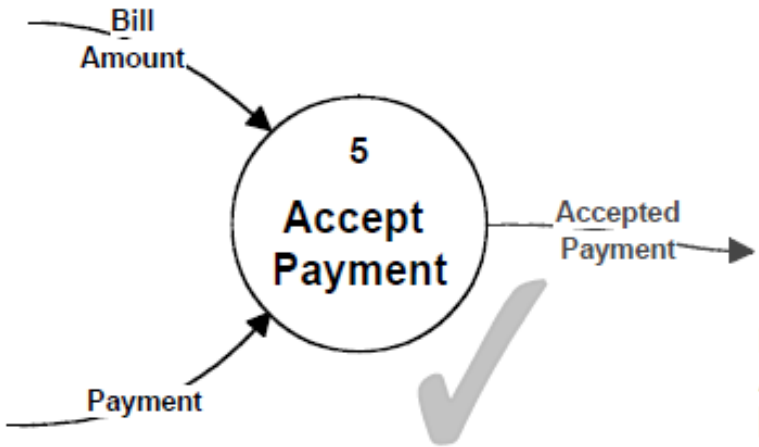
Do label data flows according to content and define any possible outcomes in the data dictionary.

Example:
Accepted Payment =
[Over Payment | Under Payment]

Do's and Don'ts of DFDs



Don't show control processing on data flows.

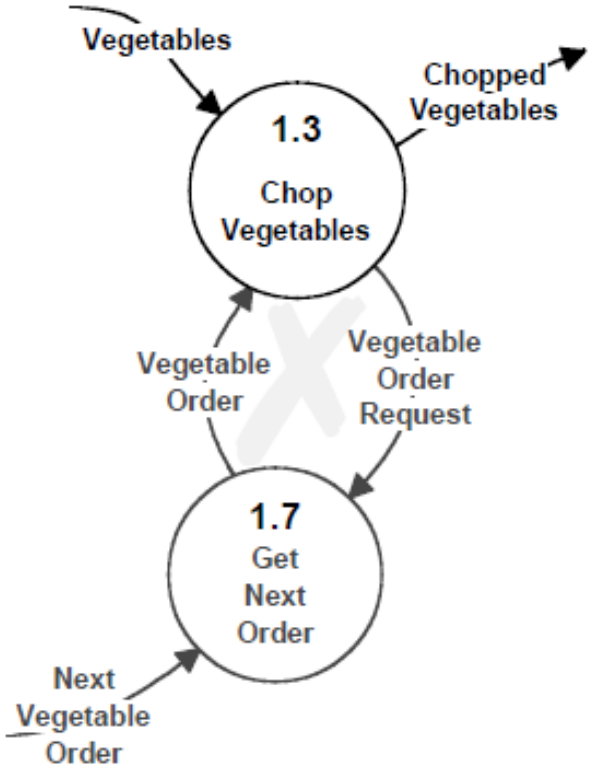


Do label data flows according to content and show control processing inside *functional primitives* with outcomes defined in the data dictionary.

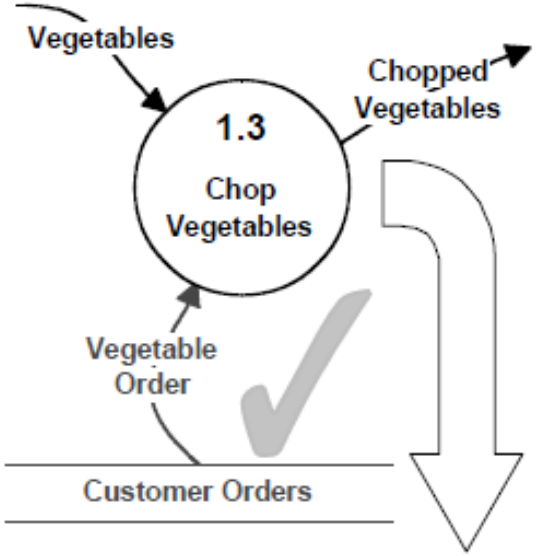
Example:
Accepted Payment =
[Over Payment | Exact Payment | Under Payment]

Do's and Don'ts of DFDs

Don't loop



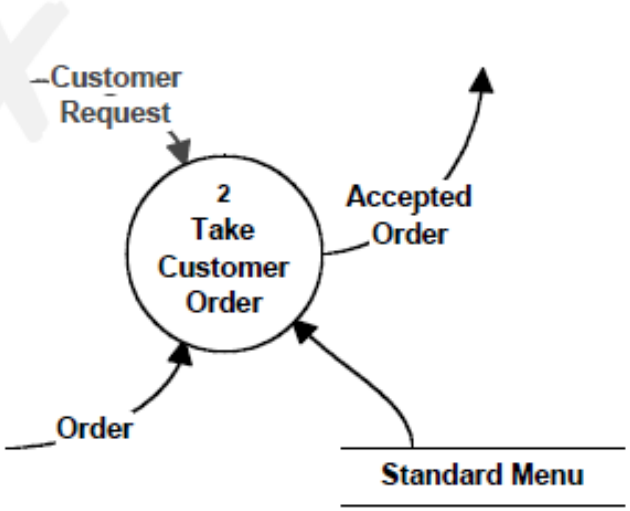
Do describe iterative processing within functional primitives.



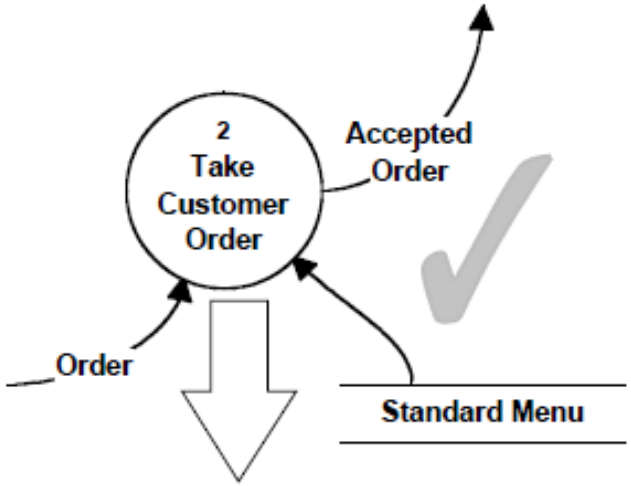
Repeat until no further input:
Read Vegetable Order from Customer Orders;
Select Vegetables according to the Vegetable Order:
Chop up all the selected vegetables;
Collect the Chopped Vegetables together and pass them on;

Do's and Don'ts of DFDs

Don't use dataflows as Process activation signals



Do describe control events within functional primitives.

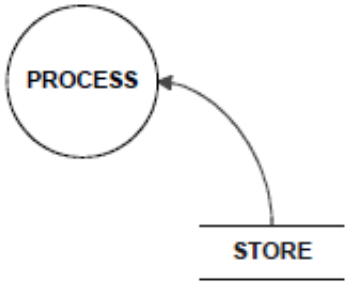


Whenever a customer indicates the intention to order:
Go to the customer prepared to take an Order;
When necessary check the Order with Standard Menu;
If .....
Then .....
.....
Place Accepted Order with any other Customer Orders.

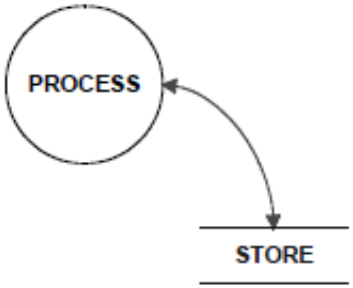
Do's and Don'ts of DFDs

- **Don't draw too many process bubbles on a DFD**
  - » Use the Miller Factor (7 plus/minus 2) wisely
  - » Too many bubbles may indicate:
    - Too little composition.
    - System is too large.
- **Don't draw too many trivial DFDs**
  - » Lots of DFDs with only a few bubbles may indicate:
    - Too much partitioning
    - A very small system
- **Do draw DFDs that contain a high degree of functional abstraction**

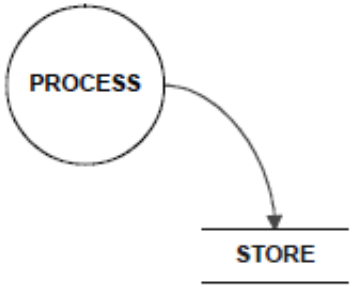
Store Access Conventions of DeMarco



READ ONLY

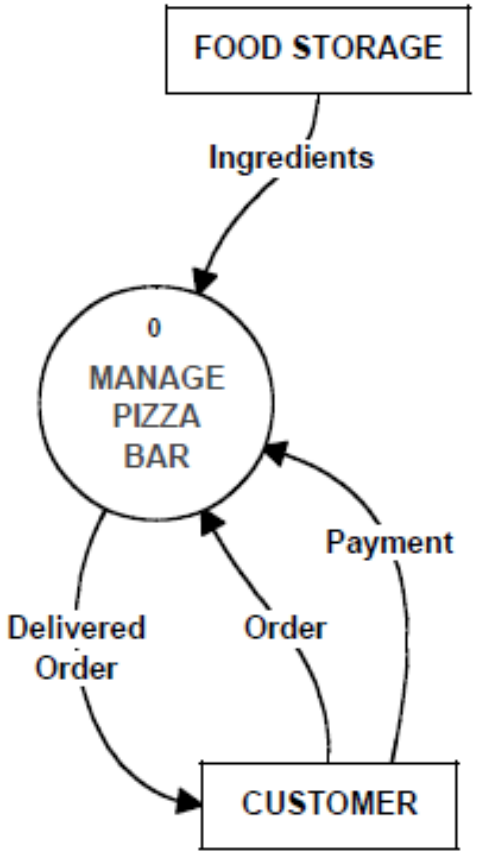


READ AND THEN WRITE



UPDATE (NET FLOW)

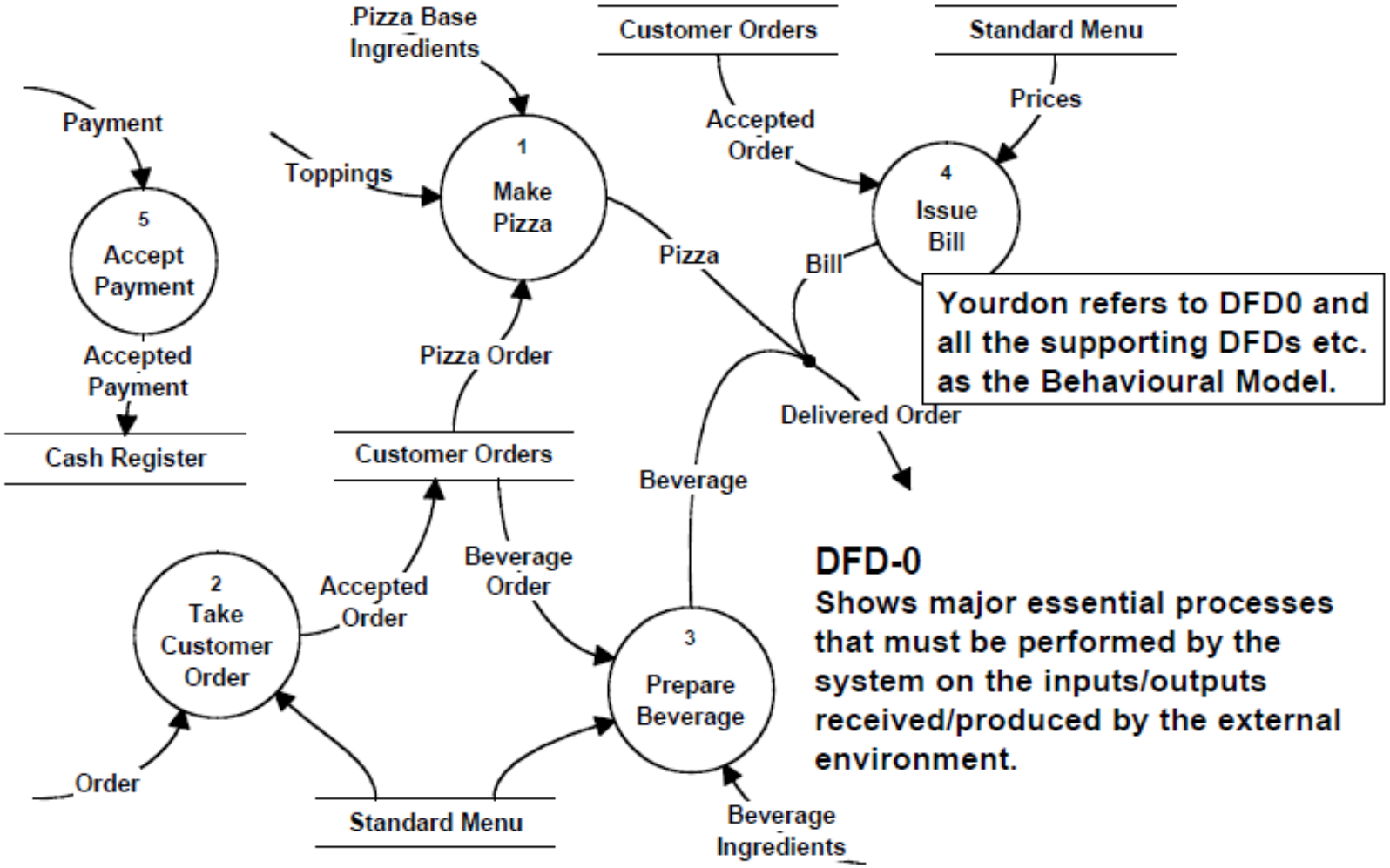
Essential Environment



System Context
Shows essential external entities with which the system (isolated as a single process) must communicate information or material items.

Yourdon refers to the Context Diagram as the Environmental Model.

Essential Behavior



The Data Dictionary

PURPOSE:

Maintain an ordered list of rigorous definitions for:

- Data Flows and Data Stores
- Elementary components of Data Flows and Stores
- Data Store Relationships

Definitions should include:

- Name (including aliases)
- Description of meaning (usually as commentary)
- Data attributes (value range, units, rate, accuracy)
- Composition (grouped or primitive)

Data Dictionary Notation

Symbol	Meaning
=	composed of
This + That	This together with That
n{ That }m	n to m iterations of That
[This | That | Another | ...]	select one of This or That or Another or ...
(This)	This is optional
" That "	literally the word That
* Note about this & that *	comment field and/or primitive element
This + That + Another + ...	key attribute of data entity (@ in TeamWork)
<That>This	That version of This (TeamWork only)

Data Dictionary Examples

Toppings = Vegetables + Cheese + Salami

*** Basic components that will form topping on pizzas ***

Vegetables =

Tomatoes + (Capsicum) + (Mushrooms) + (Hot Pepper) + (Onions)

*** The available choice of vegetables that can be mixed together***

Cheese =

*** Any type of cheese typically used for pizzas ***

Salami =

*** Any one of an available choice of salami ***

Data Dictionary Examples

Chopped Vegetables =

<Chopped>Tomatoes +
(<Chopped>Capsicum) +
(<Chopped>Mushrooms) +
(<Chopped>Hot Pepper) +
(<Chopped>Onions)

Tomatoes =
*** Any type of ripe tomato ***

Colour Range:	**Light Red - Dark Red**
Firmness Range:	**Slightly Squashy - Very Firm**
Rate:	**As needed**
Other Versions:	**<Chopped>**

Data Dictionary Examples

Salami =

*** Any one of an available choice of salami ***

Size Range:	**50mm - 75mm diameter.**
Quality Range:	**Follow Health Regulation TAFF - 1994.**
Rate:	**As needed.**
Other Versions:	**<Pepperoni>, <Hungarian>, <Danish>**

Sliced Salami =

*** Any one of an available choice of salami that is sliced ***

Thickness Range:	**2mm - 4mm.**
Rate:	**As needed.**
Other Versions:	**<Pepperoni>, <Hungarian>, <Danish>**

Data Dictionary Examples

Standard Menu =

"PIZZA CHOICES	Small	Medium	Large
Napoletana	$3.00	$5.00	$7.00
Capricciosa	$3.50	$5.50	$8.00
Mexicana (Hot)	$2.80	$4.50	$6.50
Roll-Your-Own	$3.90	$6.00	$8.40

Vegetable Choice for RYO: Capsicum, Mushroom, Hot Peppers, Onions.
Salami Choice for RYO: Pepperoni, Hungarian, Danish.

BEVERAGES

Capaccino	$1.80	Tea	$1.20
Long Black Coffee	$1.50	Coke	$1.40
Turkish Coffee	$1.50	Lemonade	$1.00
White Coffee	$1.60	Iced Coffee	$2.00"

* The menu from which customers order *

Data Dictionary Examples

Order = [Pizza Order
| Beverage Order
| Pizza Order + Beverage Order]

Accepted Order = Table Number + 1{Order}

Customer Orders = {Accepted_Order}

Pizza Order = * A pizza name chosen from the menu *
+ (Variations)

Beverage Order = * A beverage name chosen from the menu *

Data Dictionary

SUMMARY:

- **The DD should contain a definition for every Data Flow and Data Store shown on the set of DFDs.**
- **All known properties of each Data Flow and Data Store should be included with it's dictionary definition.**
- **Constructing the DD is essential to creating a specification that has the properties of being consistent and unambiguous.**
- **DD construction can be tedious so it is important to adopt an incremental approach.**

Process Specifications

PURPOSE:

- **To describe what a *primitive process* must accomplish.**
  - **Descriptions must explain what has to be done to (data) inputs in order to produce the resultant (data) outputs.**
  - **Statements must be:-**

    Readable

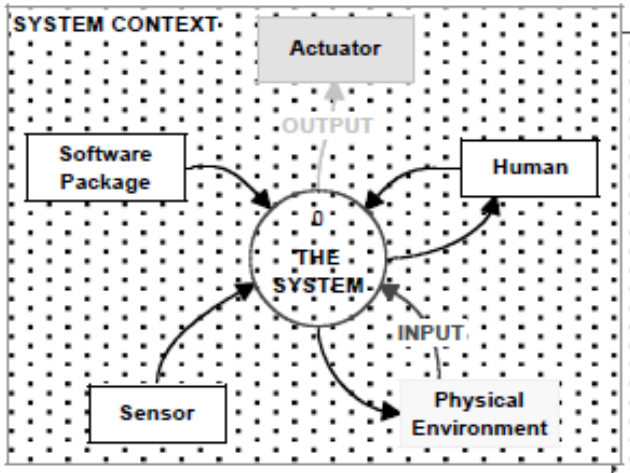
    Verifiable

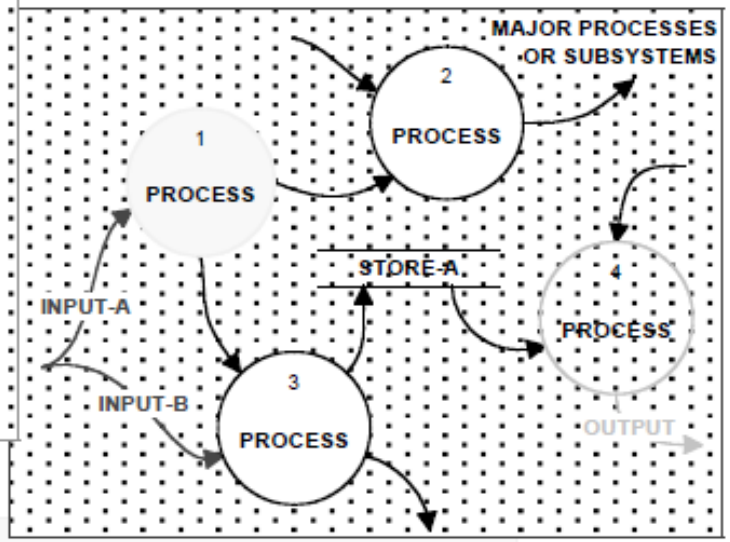
    Understandable

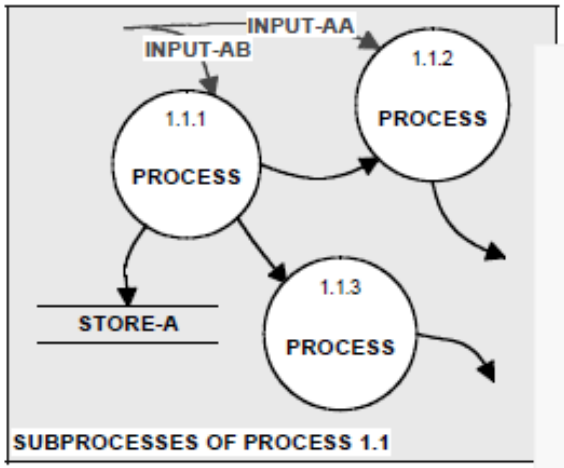
    Precise

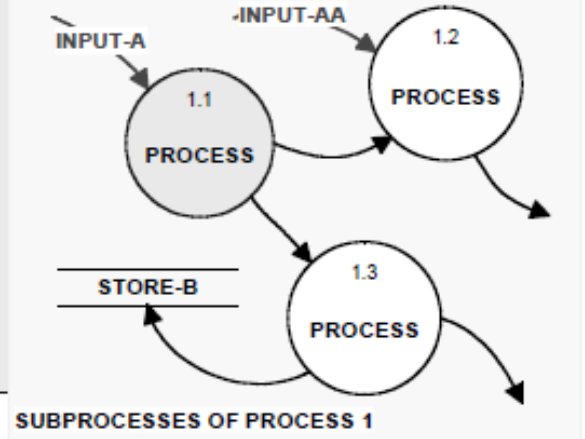
    Succinct

Process Specifications and DFDs





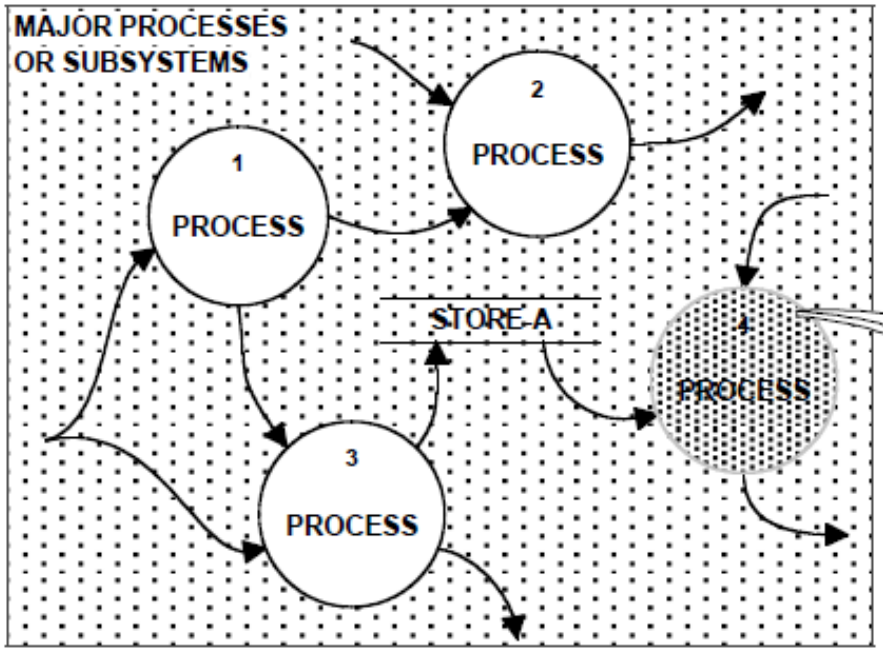




The DFD structure:-

- **IS NOT a specification.**
- **IS a logical layout.**

Process Specifications



A PSpec can occur anywhere within the DFD structure.

It is not usual for processes shown on DFD0 to become PSpecs unless the system is small (say < 20 Pspecs in all)

Process Specification

NAME:

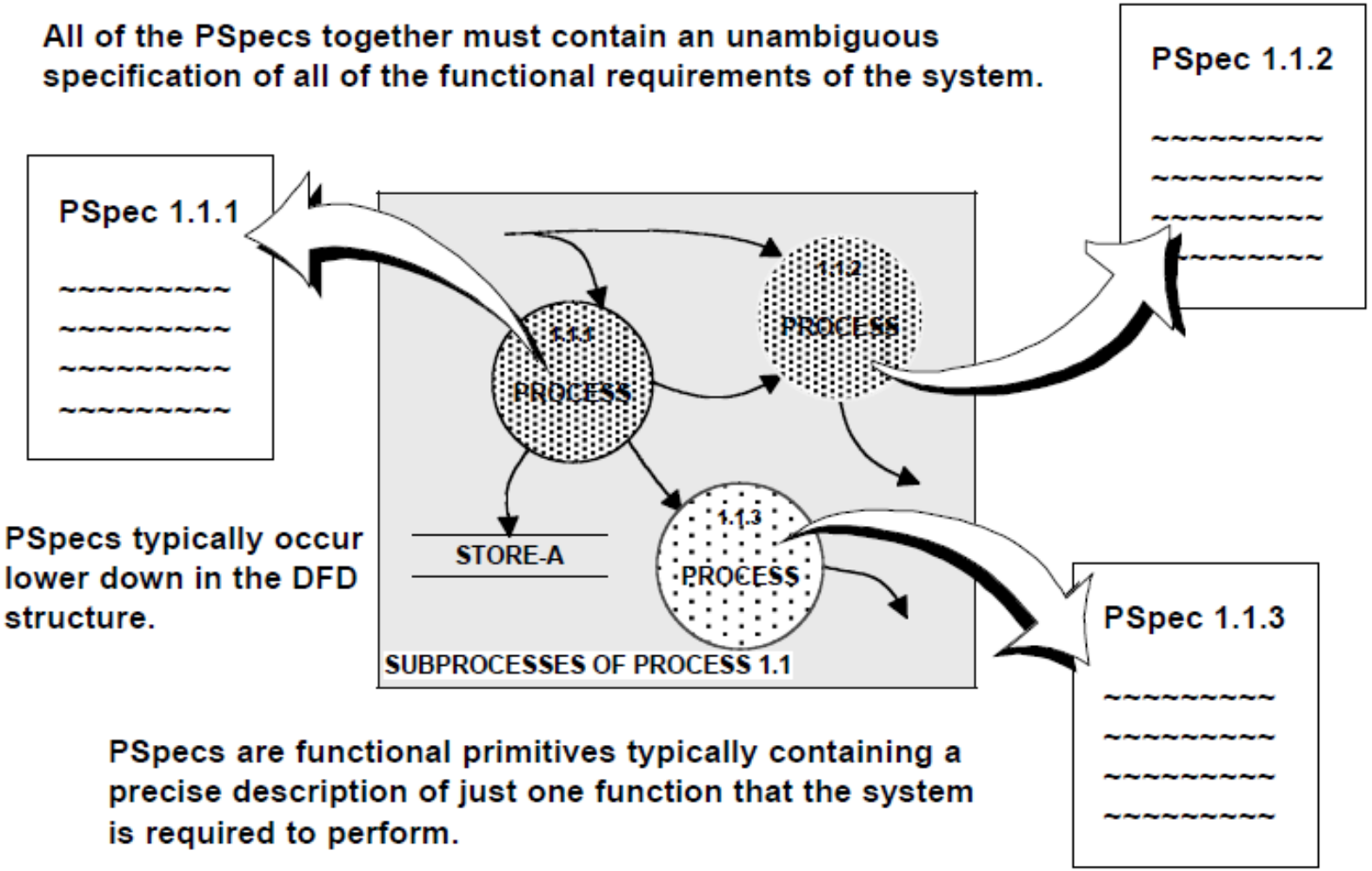
(Data) I/O:

Description:

- Textual
- Graphical
- Tabular
- Mathematical

Process Specifications

All of the PSpecs together must contain an unambiguous specification of all of the functional requirements of the system.



PSpecs typically occur lower down in the DFD structure.

PSpecs are functional primitives typically containing a precise description of just one function that the system is required to perform.

Writing Process Specifications

Describe functionality within PSpecs using:

- **Structured Language**
  - » Restricted set of English (Japanese, Korean, Indian etc.) verbs.
  - » Sequence, Selection, Iteration constructs
- **Mathematical Equations and Identities**
  - » Algebraic expressions
  - » Standard mathematical notation
- **Decision Tables/Trees**
  - » For simplification of complex decision making processes.
- **Graphs and Charts**
  - » Expressing properties/relationships

Structured Language Guidelines

1. **Choose a restricted set of suitable verbs.**
   - Preferable to select a set of action words with which the customer/user is familiar.
   - When possible, obtain these verbs from the original requirements document (such as the OCD).
   - Ensure that all such terms are used consistently.
   - Create a glossary - try and limit it to less than 100 terms.
   - Avoid using imprecise verbs or generalised entity names.

2. **Construct simple rather than compound sentences.**
   - Simple sentences are more succinct and less ambiguous than compound ones.
   - Use mathematical notation freely.
   - Restrict object/subject of a simple sentence to (data) I/O or any necessary local terms.

Structured Language Guidelines

3. Include sequence, selection & iteration constructs.

» **Sequence:-**

The natural order of statements (sentences) within a PSpec.
A PSpec should be read from top to bottom.

» **Selection:-**

IF - THEN - ELSE
CASE (WHEN) - DO

» **Iteration:-**

REPEAT - UNTIL
DO - WHILE

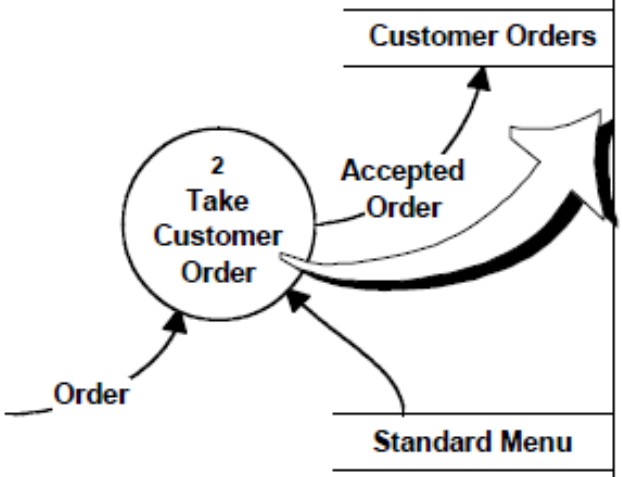
Structured Language Guidelines

4. **Develop preconditions by isolating the following:**
   - Necessary availability of particular inputs.
   - Relationships between (fields of) different inputs or conditions of elements within an input.
   - Relationships between (fields of) inputs and elements of (data) stores.
   - Relationships between (elements of) different stores or conditions of elements within a store.

5. **Develop postconditions by isolating the following:**
   - Particular outputs that will be generated from the PSpec.
   - Relationships between output values and input values.
   - Relationships between output values and values (of elements) in stores with which the process communicates.
   - Alterations to stores (Additions, Modifications, Deletions).

PSpec Examples

This maybe a first attempt at writing a process specification.



What is wrong with this PSpec?

Can it be improved?

NAME: Take Customer Order

I/O:
- Order: input
- Standard Menu: input
- Accepted Order: output

Assumptions:-
- Standard Menu is available to customer(s).
- Customer(s) at a single table has indicated readiness to order.

Description:

Repeat until all customers at table have made their (individual) Order:

Obtain from each customer or a representative:

The type of pizza (s)he wants to order

Record in writing as Pizza Order

The type of beverage (s)he wants to order

Record in writing as Beverage Order.

Verify Order against the Standard Menu.
Record the table number (or it's position) together with customer Order(s) and place with other Customer Orders as an Accepted Order.

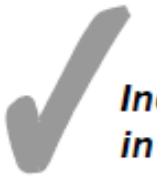
PSpec Examples

NAME: Take Customer Order
I/O: Order: input
Standard Menu: input
Accepted Order: output

Are these the only inputs & outputs?

Assumptions:-

- Standard Menu is available to customer(s).
- Customer(s) at a single table has indicated readiness to order.



Including some preconditions in this way is very helpful.

Description:
Repeat until all customers at table have made their (individual) Order:
Obtain from each customer or a representative:
The type of pizza (s)he wants to order
Record in writing as Pizza Order
The type of beverage (s)he wants to order
Record in writing as Beverage Order.

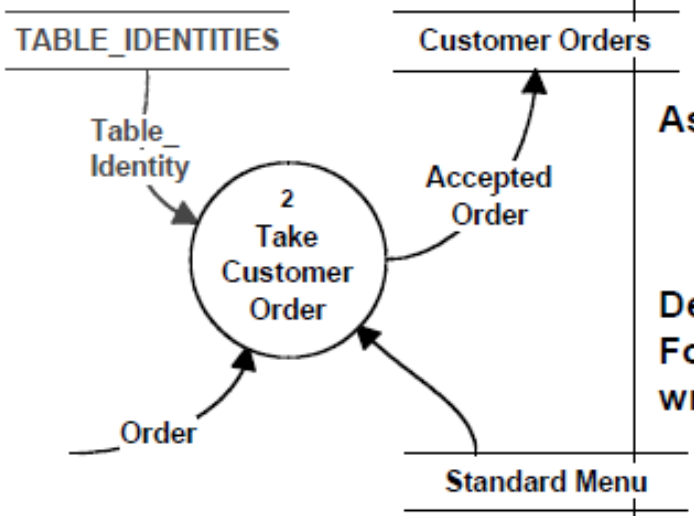
This part of the PSpec is confusing & ambiguous. Here it would seem that the pizza & beverage order are merely an output. The DD definition for Order suggests otherwise.

Verify Order against the Standard Menu.
Record the table number (or it's position) together with customer Order(s) and place with other Customer Orders as an Accepted Order.

Where from?

PSpec Examples

This attempt is better.



Can this PSpec be improved further?

NAME: Take Customer Order

I/O:

Order:	input
Standard Menu:	input
Table Identity:	input
Accepted Order:	output

Assumptions:-

- Standard Menu is available to customer(s).
- Customer(s) at a single table has indicated readiness to order.

Description:

For each customer at a table, obtain and record in writing:

an Order consisting of:

an (individual) Pizza Order and/or

an (individual) Beverage Order

Verify Order against Standard Menu.

Produce the Accepted Order by recording the *Table Identity* together with all the verified (individual) customer Order(s) from the table. Place Accepted Order with other Customer Orders.

PSpec Examples

NAME: Take Customer Order
I/O: Order: input
Standard Menu: input
Table Identity: input
Accepted Order: output

Assumptions:-
- Standard Menu is available to customer(s).
- Customer(s) at a single table has indicated readiness to order.

Description:
For each customer at a table, obtain and record in writing:
an Order consisting of:
an (individual) Pizza Order and/or
an (individual) Beverage Order
Verify Order against Standard Menu.

Produce the Accepted Order by recording the Table Identity *together with all the verified (individual) customer Order(s) from the table.*
Place Accepted Order with other Customer Orders.

Remember:
- Write succinct sentences
- Make specifications technology free.
- To use DD notation

The underlined wording can be reduced without loss of meaning.

Ambiguity and implied technology!

The underlined wording can be replaced with DD notational constructs.

The circled word supports an implied technology.

PSpec Examples

NAME: Take Customer Order

I/O:

	Order:	input
	Standard Menu:	input
	Table Identity:	input
	Accepted Order:	output

Assumptions:-

- Standard Menu is available to customer(s).
- Customer(s) at a single table is/are ready to order.

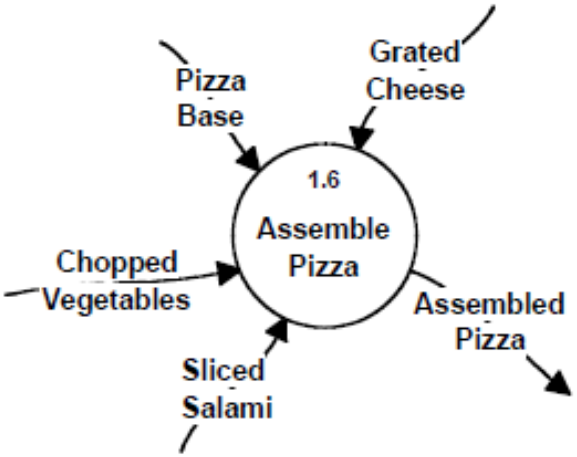
Description:

For each customer at a table, obtain and record:
an Order consisting of:
an (individual) Pizza Order and/or
an (individual) Beverage Order
Verify Order against Standard Menu.

Produce Accepted Order as a record of Table Identity + {Order}.
Store Accepted Order with other Customer Orders.

PSpec Examples

Formulate PSpecs such that the functionality described is applicable to a particular category of objects.



The description in this PSpec is applicable to all types of pizza assembled within the context of this enterprise.

NAME: Assemble Pizza

I/O:		
	Pizza Base:	input
	Chopped Vegetables:	input
	Grated Cheese:	input
	Sliced Salami:	input
	Assembled Pizza:	output

Description:
Lay the Pizza Base flat.

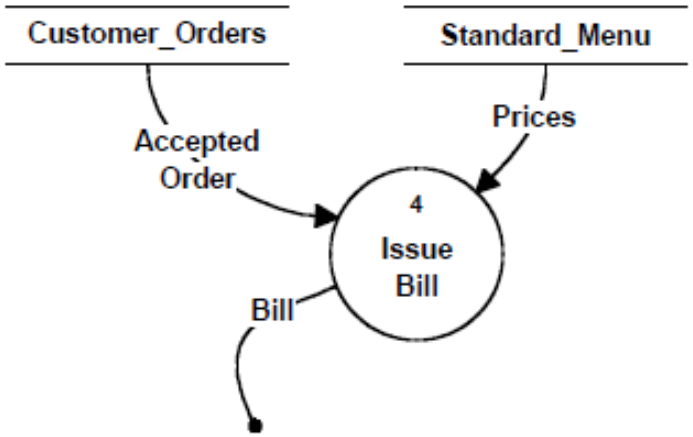
Produce Assembled Pizza by arranging the various ingredients on the Pizza Base as follows:

Spread <Chopped>Tomato evenly (on Pizza Base);
Sprinkle Grated Cheese evenly (on Pizza Base);

If other Chopped Vegetables are available
then spread them evenly (on Pizza Base)

Place Sliced Salami (on Pizza Base) such that slices do not overlap.

PSpec Examples



NAME: Issue Bill

I/O:

	Prices:	input
	Accepted Order:	input
	Bill:	output

Assumption:
A bill for Accepted Order has not already been issued.

Description:
Fetch Accepted Order from Customer Orders.

For each Order within Accepted Order:
Extract Prices from the Standard Menu.
Calculate the Total Price.
Record the Total Price (with the Order).

Calculate the Total Amount as $\sum$(Total Price).

Construct Bill as a record of:
{Order + Total price} + Total Amount.

Issue Bill as part of Delivered Order for Table Number shown on Accepted Order.

PSpec Examples

REMEMBER!!

Process Specifications are NOT PROGRAMS

Structured Language should be chosen appropriately for the particular system being specified

PSpecs as Decision Tables/Trees

- **When written descriptions become too difficult to follow because of complex decision logic then use a Decision Table/Tree to express the logic more clearly.**

- **Decision Tables/Trees are constructed when various combinations of inputs to a decision making process result in differing actions.**

Input-1	Input-2	Input-3	Result-1	Result-2	Result-3
Value-1	Value-1	Value-1	X		
Value-1	Value-2	Value-1		X	
Value-2	Value-3	Value-2		X	
Value-2	Value-1	Value-2			X
Value-1	Value-3	Value-1	X		

Decision Table Example

WRITTEN DESCRIPTION

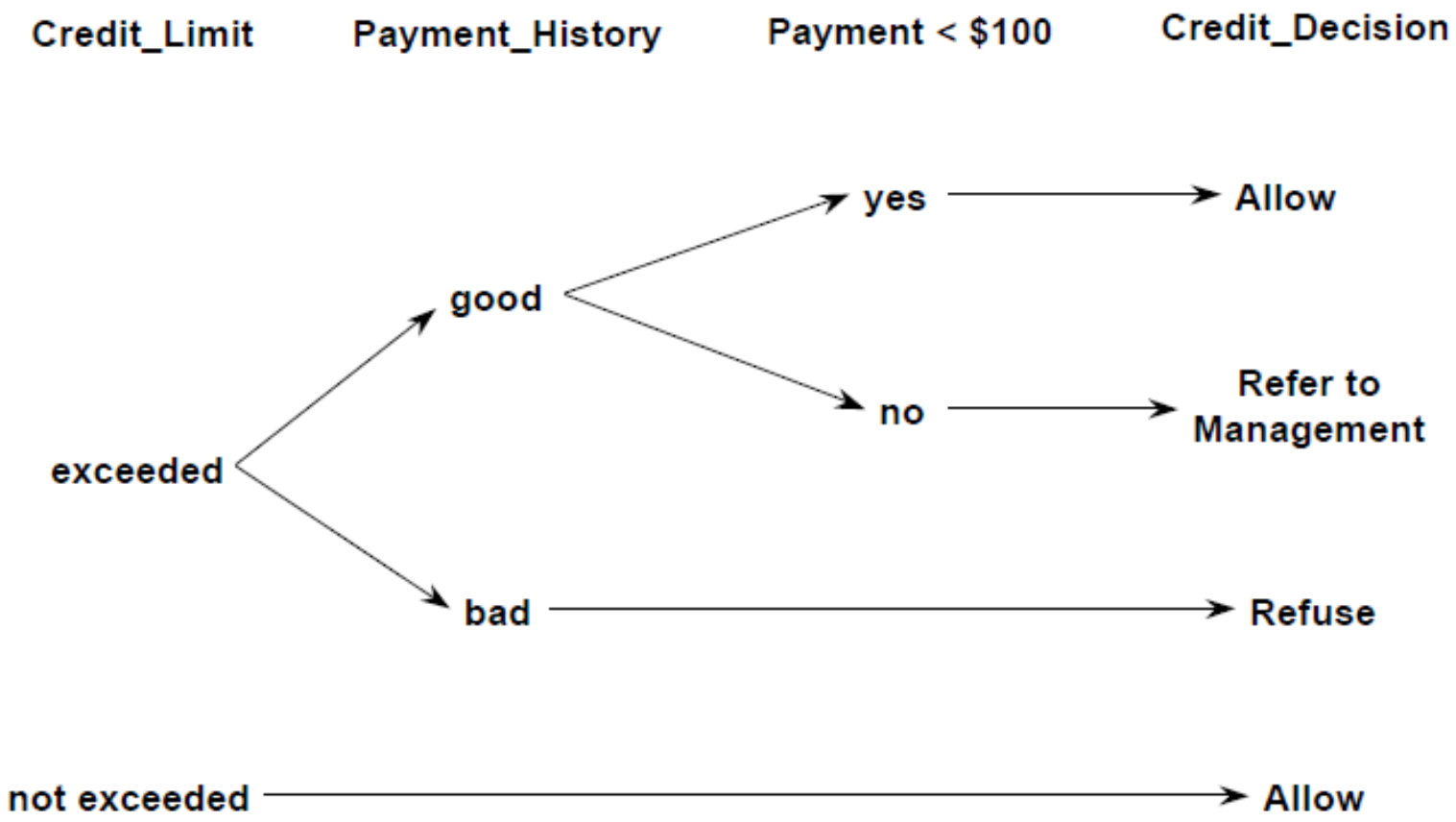
```
If Credit_Limit is not exceeded
then
      Allow Credit
else (Credit_Limit is exceeded)
      If Payment_History is bad
      then
            Refuse Credit
      else if Payment_History is good
                  and Purchase < $100
            then
                  Allow Credit
      else
            Refer to Manager
```

DT inputs and their combined values which determine resulting action

Credit_Limit is exceeded	Y	Y	Y	Y	N	N	N	N
Payment_History is good	Y	Y	N	N	Y	Y	N	N
Purchase < $100	Y	N	N	Y	N	Y	N	Y
Allow Credit	X				X	X	X	X
Refuse Credit			X	X				
Refer to Manager		X						

DT outputs as actions resulting from combinational values of inputs

Decision Tree Example



Process Specification

SUMMARY:

- **PSpecs are the most important component of the DeMarco Model.**
- **There is no advantage in trivialising PSpecs.**
- **Be prepared to write and rewrite PSpecs in order to reduce the possibility of misinterpretation and the consequent introduction of errors.**
- **Sometimes it is worthwhile writing the equivalent of PSpecs for every process bubble - this leads to good abstraction.**
- **Use of formal specifications in PSpecs is worthwhile.**

DeMarco Model & CASE Tools

FEATURES SUPPORTED:

- **Data Flow Diagrams**
  - » Usually full graphical support for drawing DFDs
  - » Balance Checking between levels
  - » Completeness Checking
  - » Syntactical Checking
- **PSpec Descriptions**
  - » I/O checking
  - » Free text format capability
- **Data Dictionary Definitions**
  - » Superset of DeMarco's DD notation
  - » Syntactical checking
  - » Usually Checking Capability includes DDE/DFD reconciliation

DeMarco Model & CASE Tools

FEATURES NOT USUALLY SUPPORTED:

- **PSpec Descriptions**
  - » Construction of Structured Language (Vocabulary and Syntax)
  - » Syntax checking of Structured Language
  - » No checking of inclusion of I/O in free format text.
- **Decision Tables and Trees**
  - » Usually done in an indirect fashion by annotation

DeMarco Model & CASE Tools

Typical Requirements Errors:

• Incorrect Facts	~50%
• Omissions	~30%
• Inconsistencies	~13%
• Ambiguities	~ 5%
• Other	~ 2%

- Neither DeMarco nor CASE Tools alone will guarantee the removal of all of these errors.
- DeMarco & CASE Tools together with proper inspection procedures will help remove most of these errors.
- Non-removal of these errors at an early stage of development will guarantee high costs.

DeMarco Model & System Development