

COMPONENT-LEVEL DESIGN

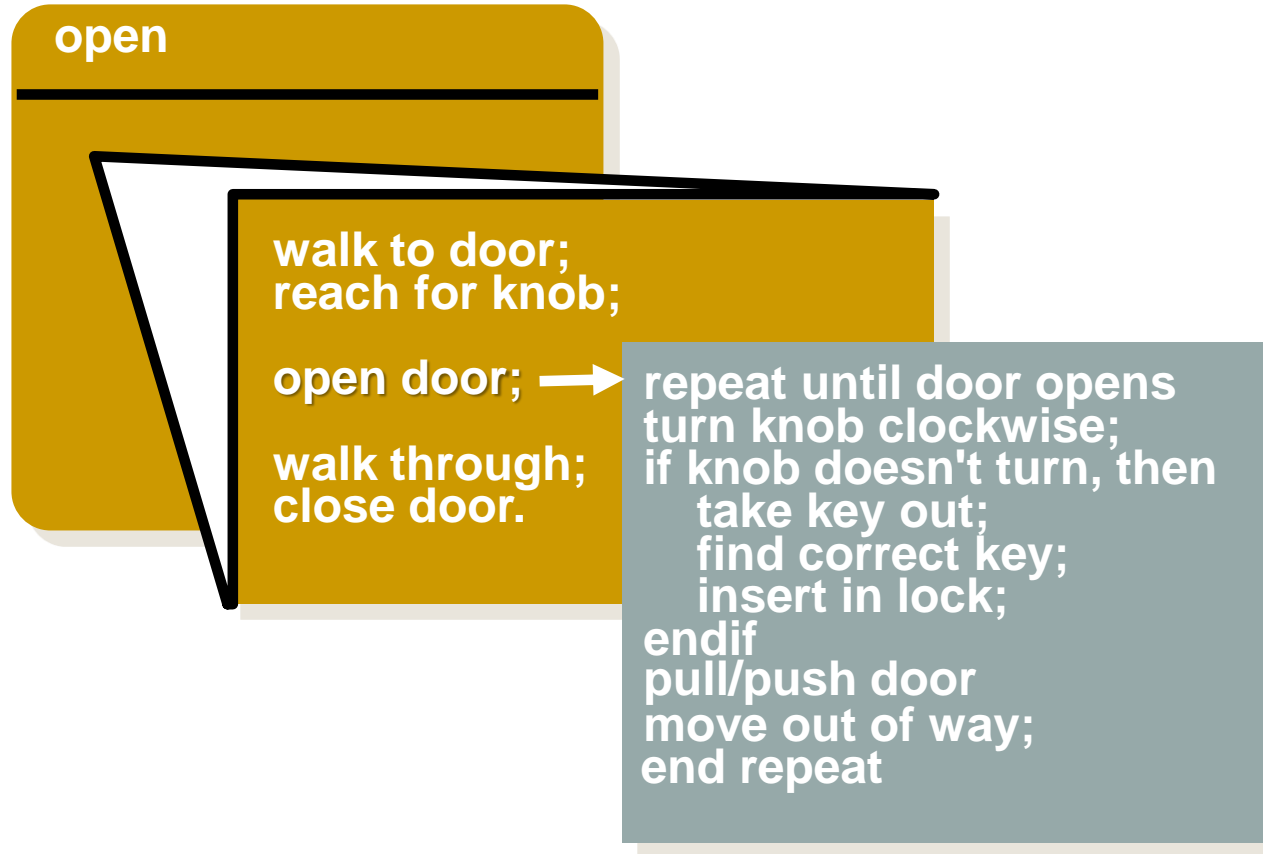


COMPONENT-LEVEL DESIGN

- the closest design activity to coding
- the approach:
 - review the design description for the component
 - use stepwise refinement to develop algorithm
 - use structured programming to implement procedural logic
 - review and iterate as required



STEPWISE REFINEMENT



THE COMPONENT-LEVEL DESIGN MODEL

- **represents the algorithm at a level of detail that can be reviewed for quality**
- **options:**
 - **graphical (e.g. flowchart, box diagram)**
 - **pseudocode (e.g., PDL) ... choice of many**
 - **programming language**
 - **decision table**
 - **conduct walkthrough to assess quality**

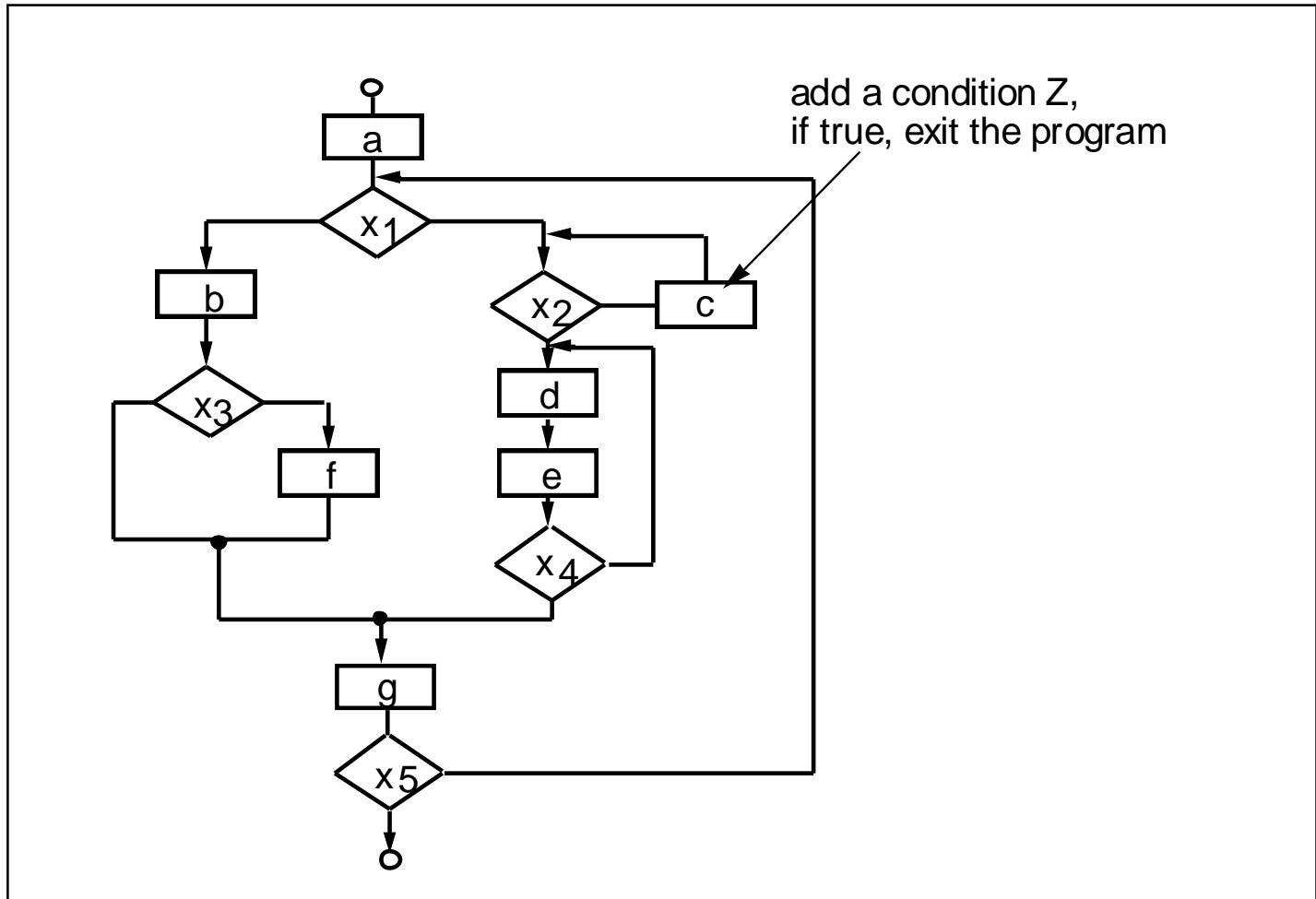


STRUCTURED PROGRAMMING **FOR PROCEDURAL DESIGN**

- **uses a limited set of logical constructs:**
 - ▣ *sequence*
 - ▣ *conditional* — if-then-else, select-case
 - ▣ *loops* — do-while, repeat until
- **leads to more readable, testable code**
- **important for achieving high quality,
but not enough**



A STRUCTURED PROCEDURAL DESIGN



TABULAR DESIGN NOTATIONS USING DECISION TABLES

DT is divided into four sections

- 1. Upper left-hand quadrant contains all conditions**
- 2. Lower left-hand quadrant contains all actions**
- 3. Upper Right-hand quadrant contains conditions combinations**
- 4. Lower Right-hand quadrant contains corresponding actions to conditions**



Rules

Conditions	1	2	3	4					n
Condition #1	✓			✓	✓				
Condition #2		✓		✓					
Condition #3			✓		✓				
Actions									
Action #1	✓			✓	✓				
Action #2		✓		✓					
Action #3			✓						
Action #4			✓	✓	✓				
Action #5	✓	✓			✓				

Decision table nomenclature



EXAMPLE

You must decide how to dress for the weather. The conditions are that it may or may not be cloudy and it may or may not be cold. If it is cloudy, take an umbrella. If it is cold, take a coat.

Conditions	Combinations of Conditions			
Cloudy	Y	Y	N	N
Cold	Y	N	Y	N
Actions	Combinations of Actions			
Take umbrella	X	X		-
Take Coat	X		X	-



Example

Condition s	Combinations of Conditions			
Important	Y	Y	N	N
Urgent	Y	N	Y	N
Actions	Combinations of Actions			
Do	X			
Plan		X		
Delegate			X	
Eliminate				X

PRACTICE

EXAMPLE:

- **Conditions:**
 - **Valid user ID**
 - **Valid password**
 - **Sufficient Balance**

- **Actions**
 - **Login accepted**
 - **Amount transferred**



SOLUTION:

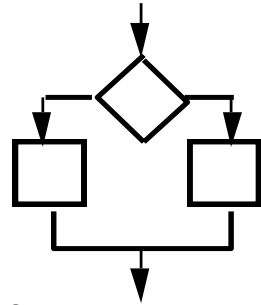
Conditions	Combinations of Conditions							
Valid User Name	Y	Y	Y	Y	N	N	N	N
Valid ID	Y	Y	N	N	Y	Y	N	N
Sufficient Balance	Y	N	Y	N	Y	N	Y	N
Actions	Combinations of Actions							
Login accepted	T	T	F	F	F	F	F	F
Amount Transferred	T	F	F	F	F	F	F	F



PROGRAM DESIGN LANGUAGE (PDL):

- **Program design language (PDL), also called *structured English* or *pseudo code*. It uses the vocabulary of one language (i.e., English) and the overall syntax of another (i.e., a structured programming language)**
- ***The difference between PDL and a real programming language lies in the use of narrative text (e.g., English) embedded directly within PDL statements.***
- **PDL tools currently exist to *translate* PDL into a programming language (such as C and ADA) "skeleton" and/or a graphical representation.**





if-then-else

```
if condition x
  then process a;
  else process b;
endif
```

PDL

- ❑ easy to combine with source code
- ❑ machine readable, no need for graphics input
- ❑ graphics can be generated from PDL
- ❑ enables declaration of data as well as procedure
- ❑ easier to maintain



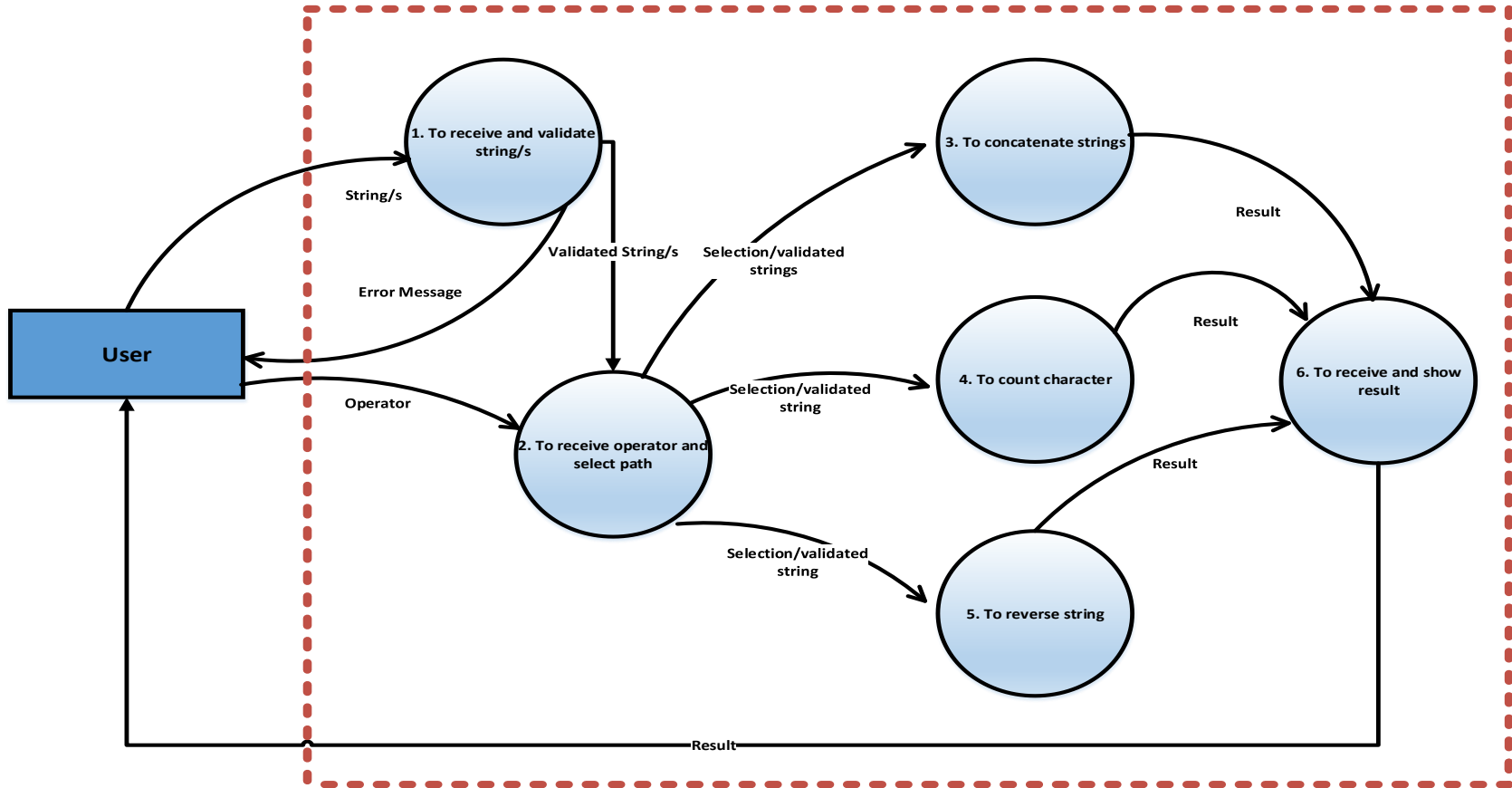
WHY DESIGN LANGUAGE?

- ☐ can be a derivative e.g., Ada PDL
- ☐ machine readable and processable
- ☐ can be embedded with source code, therefore easier to maintain
- ☐ can be represented in great detail, if designer and coder are different
- ☐ easy to review



EXAMPLE:

Level 1 DFD




```
Char* concat(char* S1, char *S2)
{
    char* newstr = new char[Len(S1) + Len(S2)];
    int c=0;
    for (i=0; i<Len(S1); i++)
        newstr[c++] = S1[i];
    newstr[c++] = ' ';
    for (i=0; i<Len(S2); i++)
        newstr[c++] = S2[i];
    newstr[c]='\0';
    return newstr;
}
```

