Design methods: JSP and JSD†

- Stands for: *Jackson Structured Programming* and *Jackson System Development*.

- Primary applicable notations:
  - *System specification diagrams* (SSDs), correspond roughly to DFDs.
  - *Entity-structure diagrams* (ESDs), process interpretation of structure diagrams.
  - Pseudo-code.

- Overall objective: *Derive complete detailed design from requirements specification in the case of JSD; derive pseudo-code for an individual process in an SSD, in the case of JSP.*

† Material from text by Budgen and from “Software Engineering Concepts”, by Richard Fairley.

CS646: Software Design and Architectures
JSP: Overall process

Starting with

1. Derive the structure diagrams for the input and output streams.

2. The input-output structure diagrams are converted to a sequential processing diagram by identifying points of correspondence between nodes in the input and output structure diagrams.

3. The sequential processing diagram is expanded into a detailed design model with pseudo-code that contains the operations needed to solve the problem.

CS646: Software Design and Architectures
Example: an inventory reporting application

Inventory data contains a collection of transactions, each of which has a part number and number of units of that item issued or received in the transaction. An output report is to be produced that contains a heading and a net movement line for each part number that occurs in the collection.
The input and output structure diagrams
Identifying points of correspondence

- input stream
- output report
- part group
- heading
- movement record
- net movement line
- issue
- receipt
- body
Derive sequential processing diagram
Derive additional operations

1. Open sort stream.
2. Close sort stream.
3. Terminate program.
4. Read a [PartNumber, CreditDebit, Amount] transaction.
5. Write report heading.
6. Write a net movement line.
7. Set NetMovement to zero.
8. Add Amount to NetMovement.
9. Subtract Amount from NetMovement
Add operations to sequential processing diagram
Derivation of pseudo-code

begin
  open sort stream;
  read a [PartNumber, CreditDebit, Amount] transaction;
  write heading;
  loop while not end-of-sort-stream
    set NetMovement to zero;
    loop while same-part-number
      if (CreditDebit = ‘debit’) then
        Subtract Amount from NetMovement
      else
        Add Amount to NetMovement;
      end if
      read a [PartNumber, CreditDebit, Amount] transaction;
    end loop;
    write a net movement line;
  end loop;
  close sort stream;
  terminate program;
end
Complications

- Structure clashes: occur when points of commonality between input and output structure diagrams cannot be determined. A suggested solution is termed *program inversion*, and is essentially the decomposition of the process into two processes.

- The need for lookahead. E.g., processing any element of a sequence of items depends on all items being error free. Solution is to employ *backtracking*.
JSD: Overall process

A process model for both analysis and design. Overall process can be described by an ESD (i.e., entity structure diagram).
JSD: Overall process (cont’d)

Results of each stage and phase:

**Entity analysis**: a collection of ESDs.

**Initial model phase**: an SSD and additional ESDs.

**Adding interactive functions**: a modified SSD and additional ESDs.

**Adding information functions**: a modified SSD and additional ESDs.

**Process timing and synchronization**: a modified SSD, additional ESDs and timing constraints.

**Implementation stage**: pseudo-code for ESDs, process packaging, etc.
JSD: SSDs versus DFDs

transactions

inventory data

transactions (ordered by part number)

generate report

net movement report

manager

inventory data manager

SV

sort by part number

1

generate report

2
JSD: SSDs versus DFDs (cont’d)

- State vectors are “owned” by a single process.

- A “rough merge” can be specified for input streams.