How to THINK like a Programmer

Problem Solving for the Bewildered

paul vickers

chapter 8

looking forward to program design

Purpose

- Identify different formalized program design methods and their associated graphical notations
- Understand the difference between bottom-up and top-down approaches and between data structure, data flow and object-oriented methods
- Use different graphical notations to highlight different aspects of a given problem

HTTLAP and algorithms

- The How To Think Like A Programmer (HTTLAP) strategy gives us a way to think about and understand problems and leads us into the construction of algorithms
- Specialized program design methods and notations can be used alongside HTTLAP to give us more ways to view problems
- Specific program design methods and notations have been developed with particular types of problem in mind
- Pick the method/notation that best fits your type of problem

Dysfunctional decomposition

- A popular ‘technique’ today (and dating back to the 1970s) is the so called top-down or functional-decomposition approach (aka stepwise refinement)
- Has some serious conceptual flaws (see the book)

Bottom-up design

- In bottom-up we consider the small details first
- Solutions to the many small problems are worked out and these solutions are then assembled into a hierarchy of functions in a bottom-up manner
- Often difficult to make these separately designed solutions fit together later on as no account has been taken of the overall shape of the problem and its solution

Data structure approaches

- All programs process data
- When the data can be structured in some formal way (e.g. into hierarchies) we can use one of the data structure approaches
- Jackson Structured Programming (JSP -- not the same as Java Server Pages also called JSP) from 1975 is one of the best data structure approaches
- Also see
  - LCP (Logical Construction of Programs)
  - Warnier-Orr, also known as DSSD (Data Structure Systems Development)

Data flow approaches

- In data flow thinking we approach the problem by considering the data in the problem and how it flows between the things that process it
- Uses data flow diagrams (DFD)
- Common methods include
  - SSADM (Structured Systems Analysis & Design Method)
  - YSM (Yourdon Structured Method)
- Data flow methods typically use the top-down philosophy (*dys*functional decomposition!)

Object oriented design

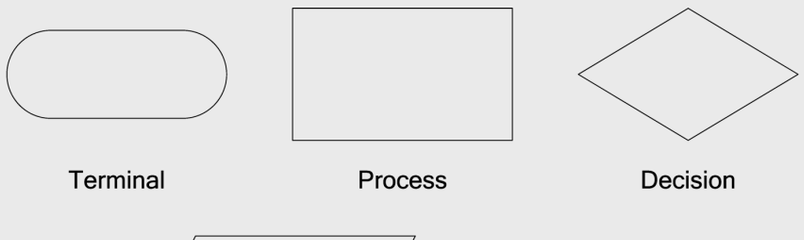
- When using languages like C++ or Java we might use an OO design technique to complement them
- Many methods around today, most of them use UML (the Unified Modeling Language) to deal with the diagrams and documentation

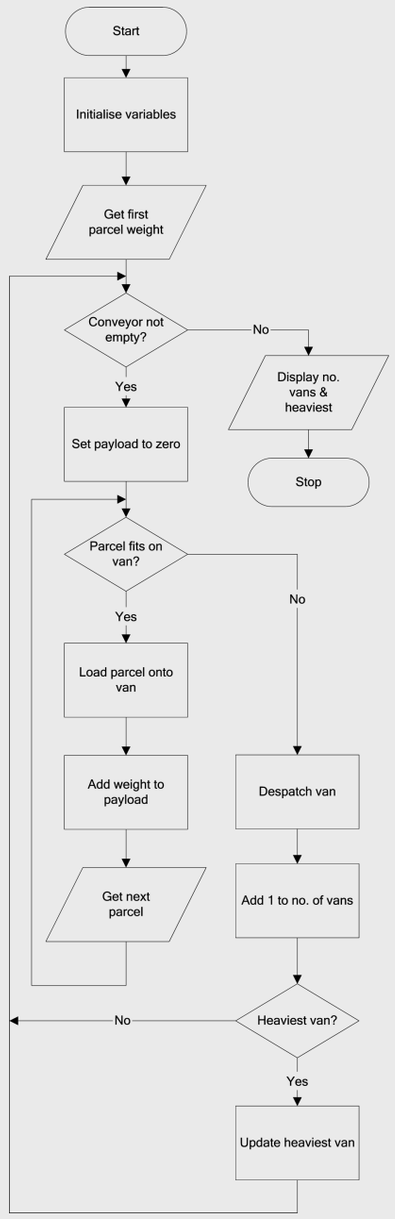
Graphical notations

- Many graphical notations exist to support the different kinds of program design methods
- We will consider a few of the most common
  - Flowcharts
  - Tree diagrams (Jackson structure diagrams)
  - State transition diagrams
  - Data flow diagrams

Flowcharts

- Diagrammatic representation of an algorithm





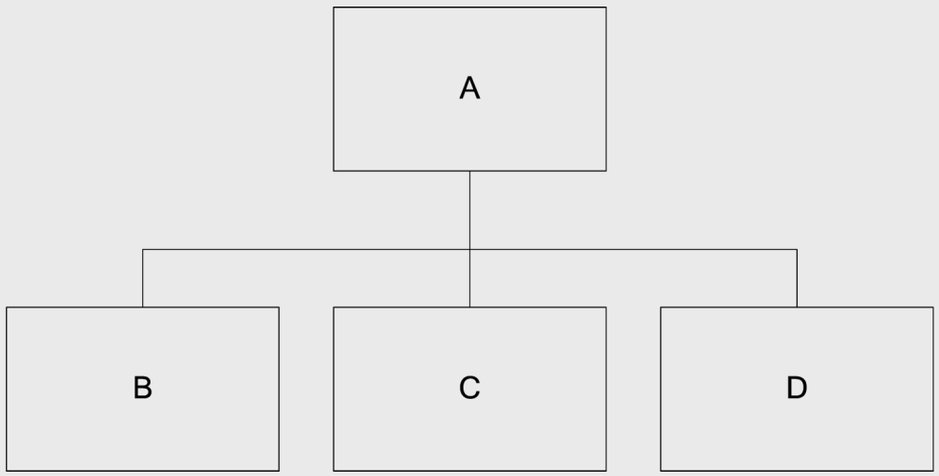
- Many ways to draw the same algorithm (see Fig 8.3 & 8.4)

Tree diagrams

- Used by the Jackson Structured Programming (JSP), Jackson System Development (JSD) and SSADM methods
- Very good at showing the structure of algorithms
- Show sequence, iteration, and selection very clearly

Tree diagram: sequence

- Here's the tree diagram notation for a sequence

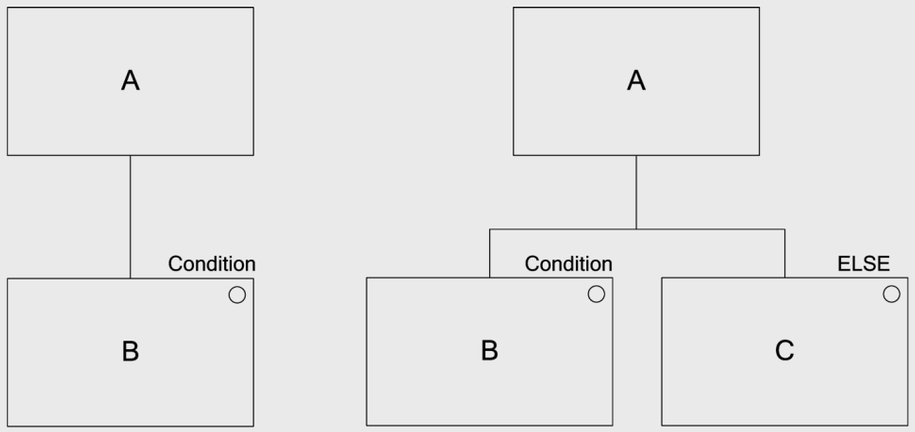


- And the corresponding pseudo-code

```
// Sequence A
Action B ;
Action C ;
Action D ;
// End of Sequence A
```

Tree diagram: selection

- Here's the tree diagram notation for two selections



Example (a) Example (b)

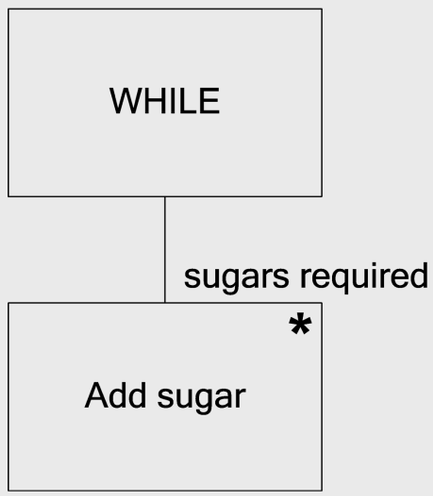
- And the corresponding pseudo-code

```
// Selection A
IF (condition)
   Action B ;
ENDIF
// End Selection A
```

```
// Sequence A
IF (condition)
   Action B ;
ELSE
   Action C ;
ENDIF
// End of Sequence A
```

Tree diagram: iteration

- Here's the tree diagram notation for iteration



- And the corresponding pseudo-code

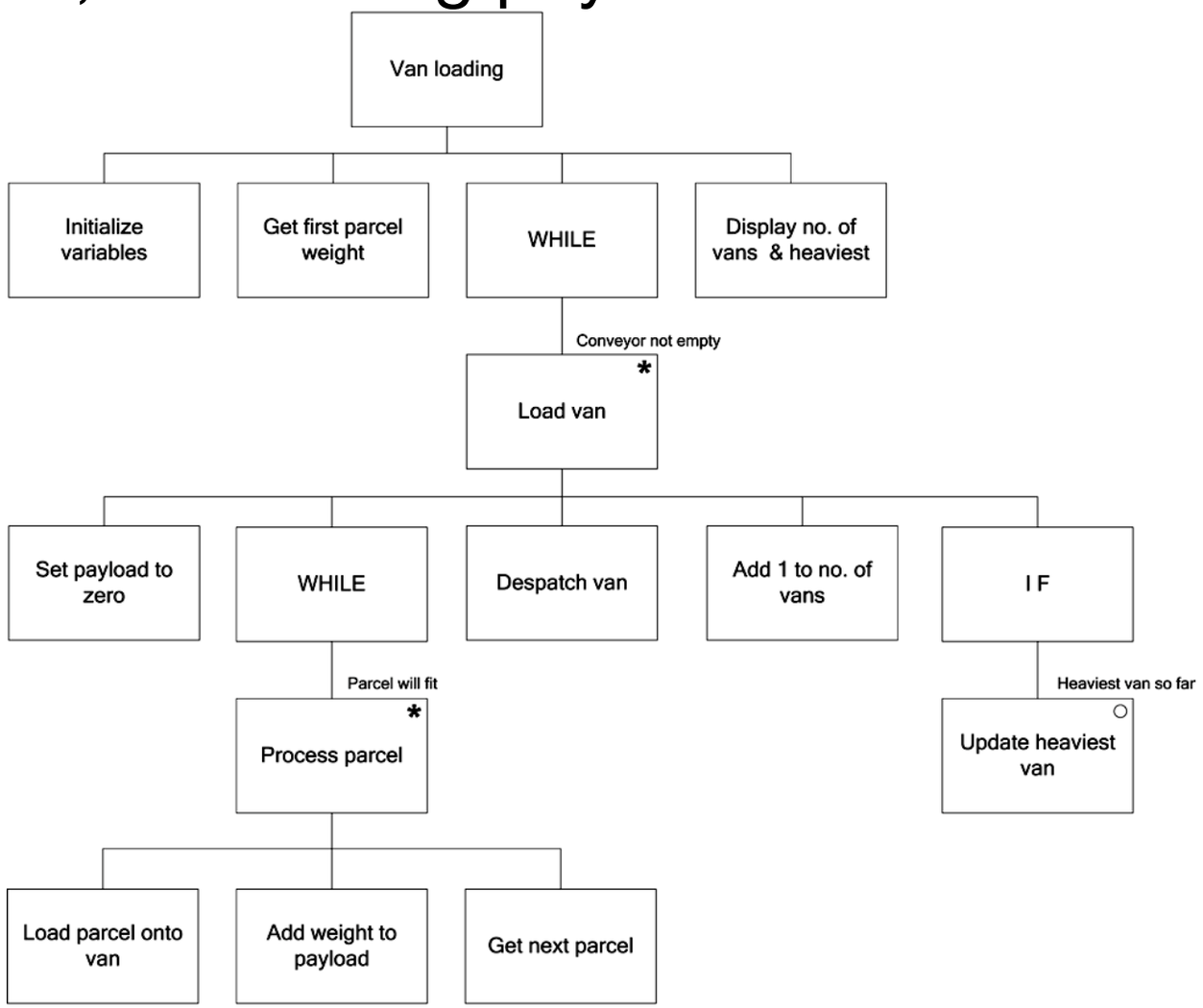
```
// Adding sugar
WHILE (sugars required)
    Add sugar ;
ENDWHILE
// End of adding sugar
```

Tree diagrams

- Very good at highlighting the relationships between components
- You can easily see what parts belong to what
- Nested structures are plainly visible and the consequences of following certain selection paths are clear
- Aka structure diagrams because they show the structure of an algorithm or a data structure

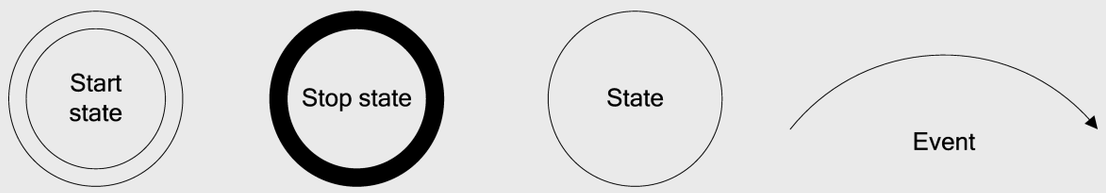
ACTIVITY

Look at the tree diagram below for the van loading solution. What order are the following actions carried out in: processing a parcel, despatching a van, and setting payload to zero?

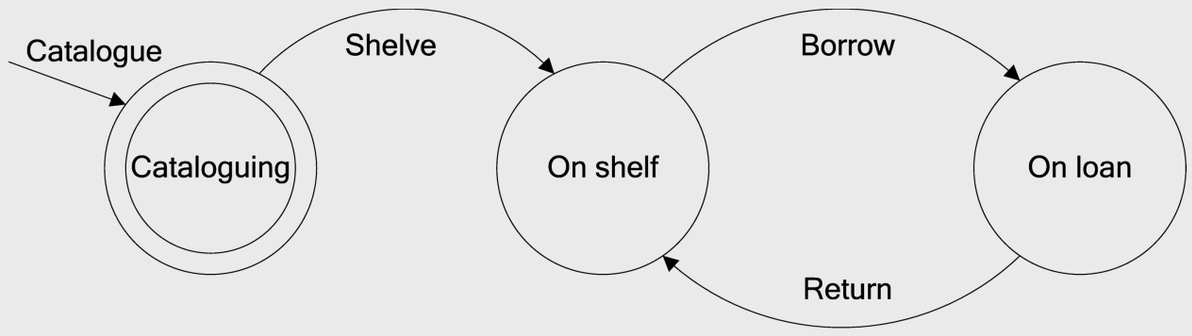


State transition

- State transition diagrams are good show showing what events lead to what states in an algorithm
- Here are some common symbols (n.b. different design methods use different shapes, but the principles are the same)

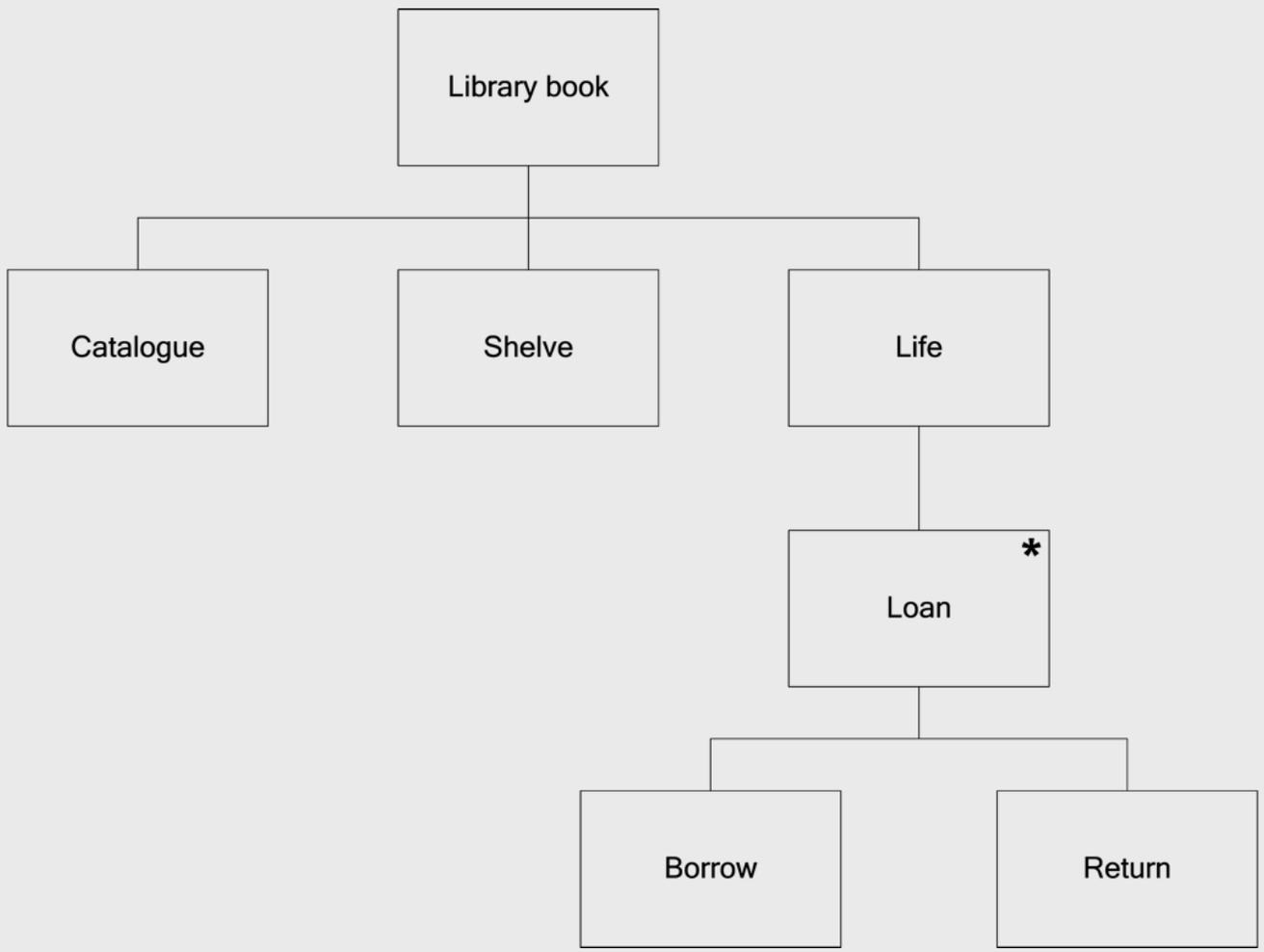


- STD for a library book:



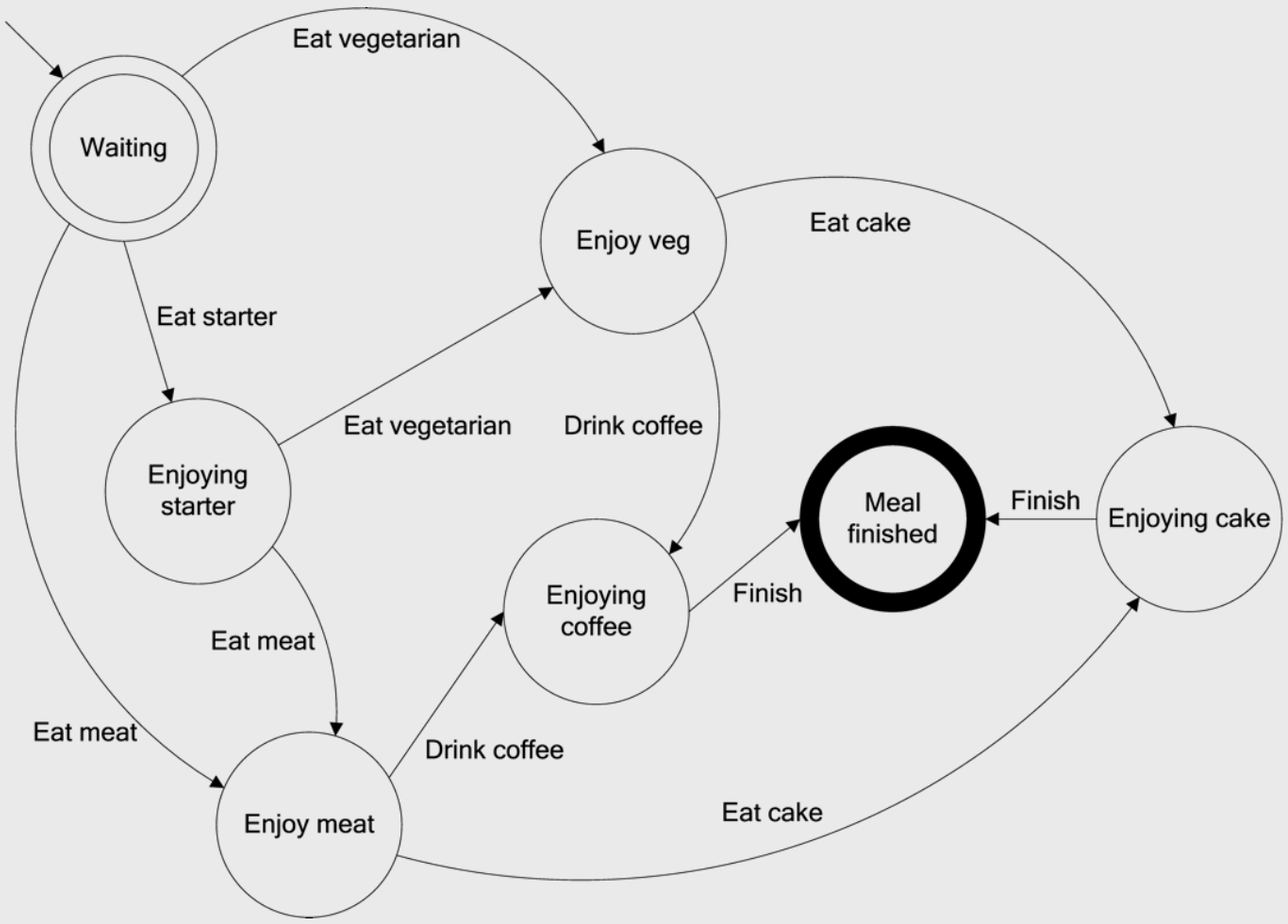
convert to a tree

- A state transition diagram can be changed to a tree diagram (and vice versa)
- Here's the tree diagram for the same library book:



Eating a meal

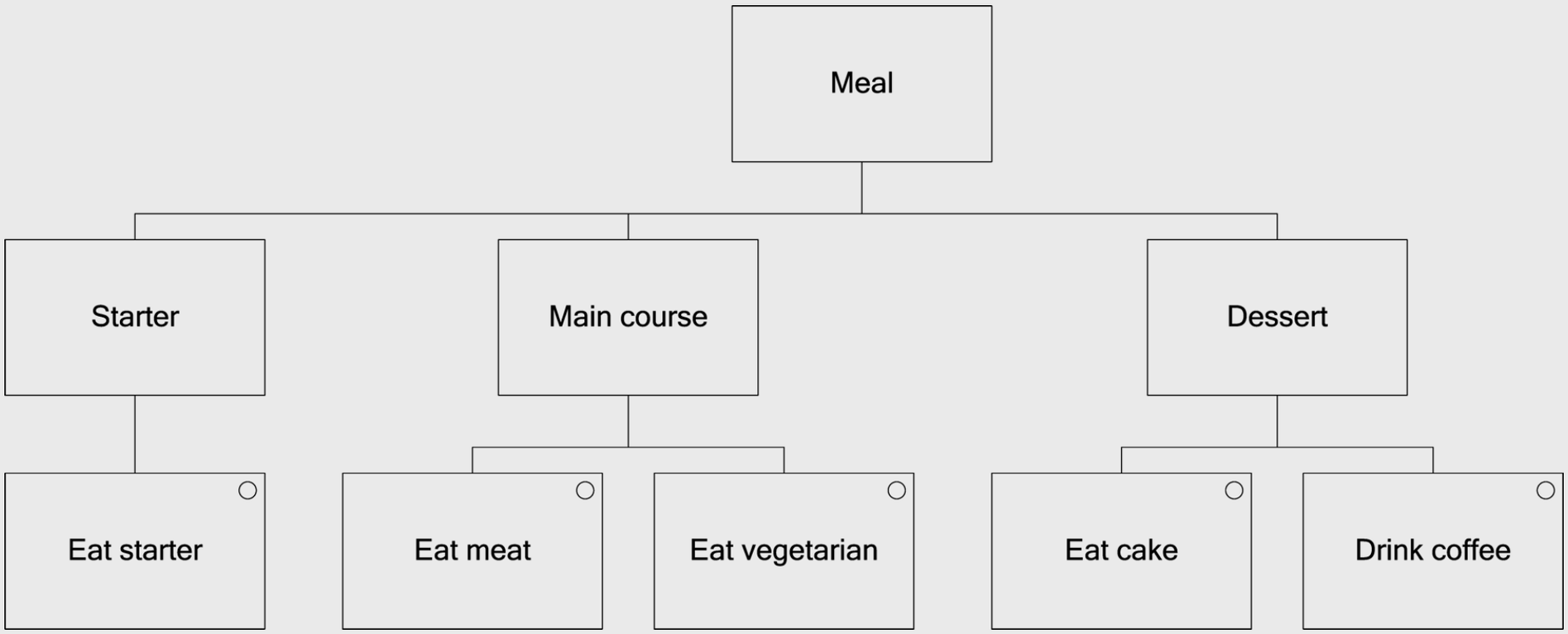
- State diagram for eating a meal



- Shows which state may be reached from any other state. Sequential progression of meal unclear

Eating a meal

- And the corresponding tree diagram



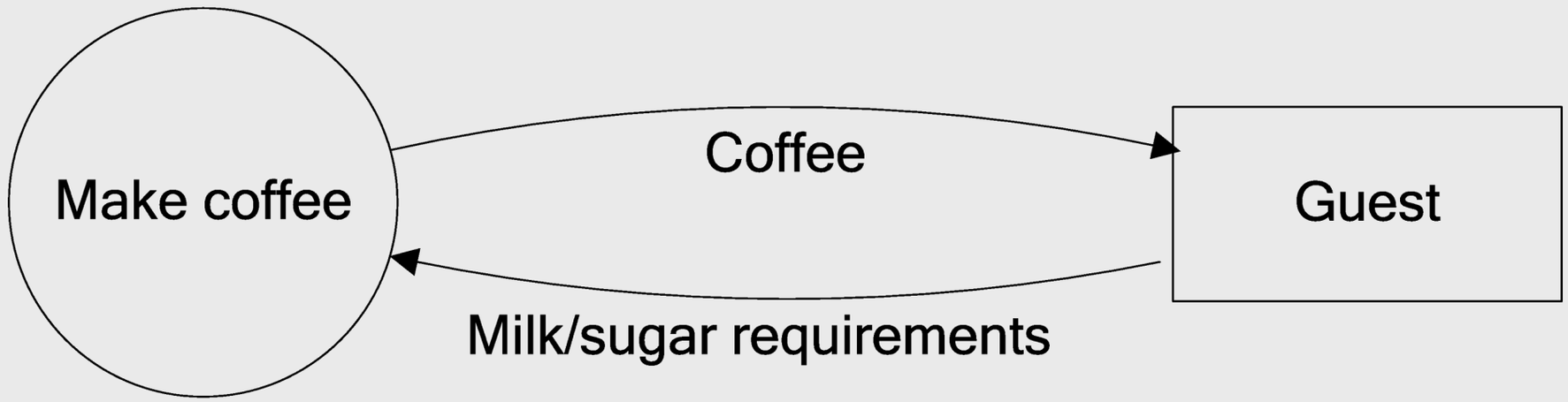
- Doesn't clearly show states but does show the sequential progression through the meal

Data flow diagram

- Popular in top-down methods
- Different notational styles, but the principles are the same
- Show how data flows between the architectural units of a program

DFD for making coffee

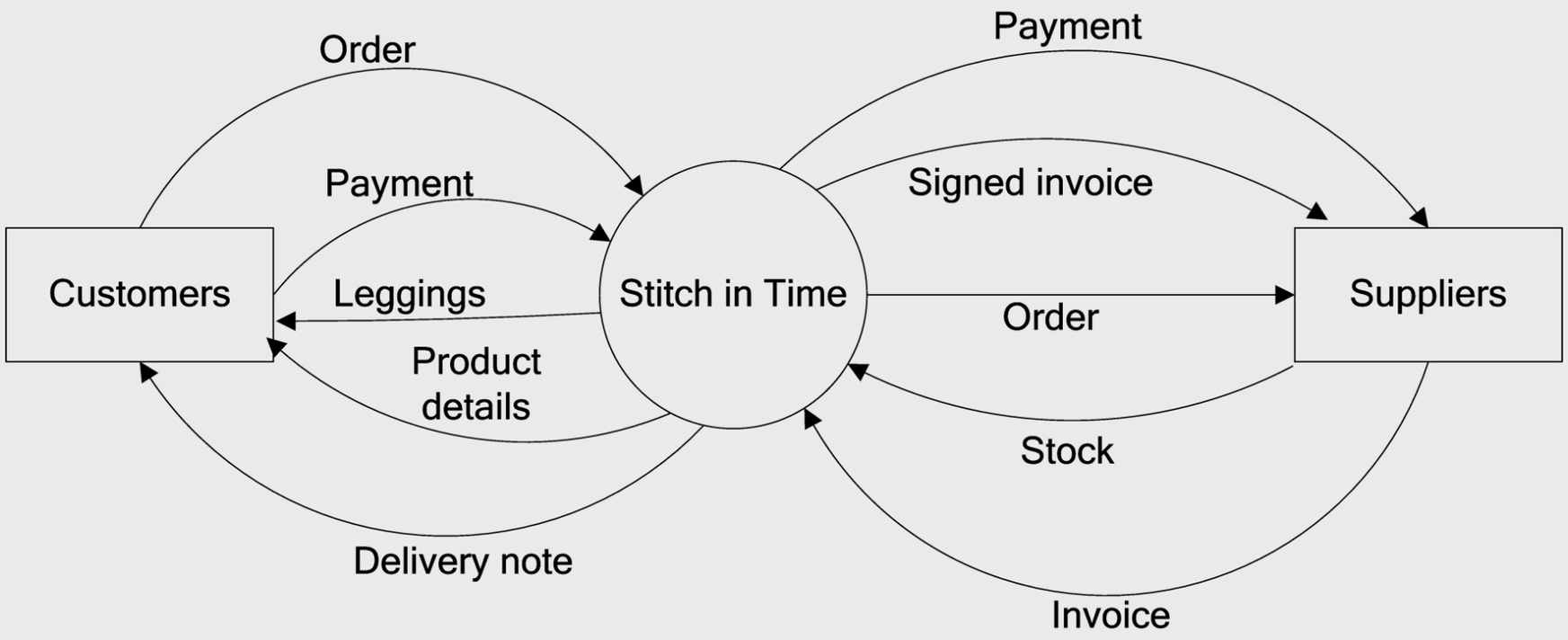
- Here's a DFD for our coffee making problem



- Typically we start with a top-level DFD (called a context diagram) and then decompose it to show increasing detail (lowering the abstraction level)

Context diagram

- Here's a context diagram for a fictitious company (see p. 201) that sells leggings to pregnant women

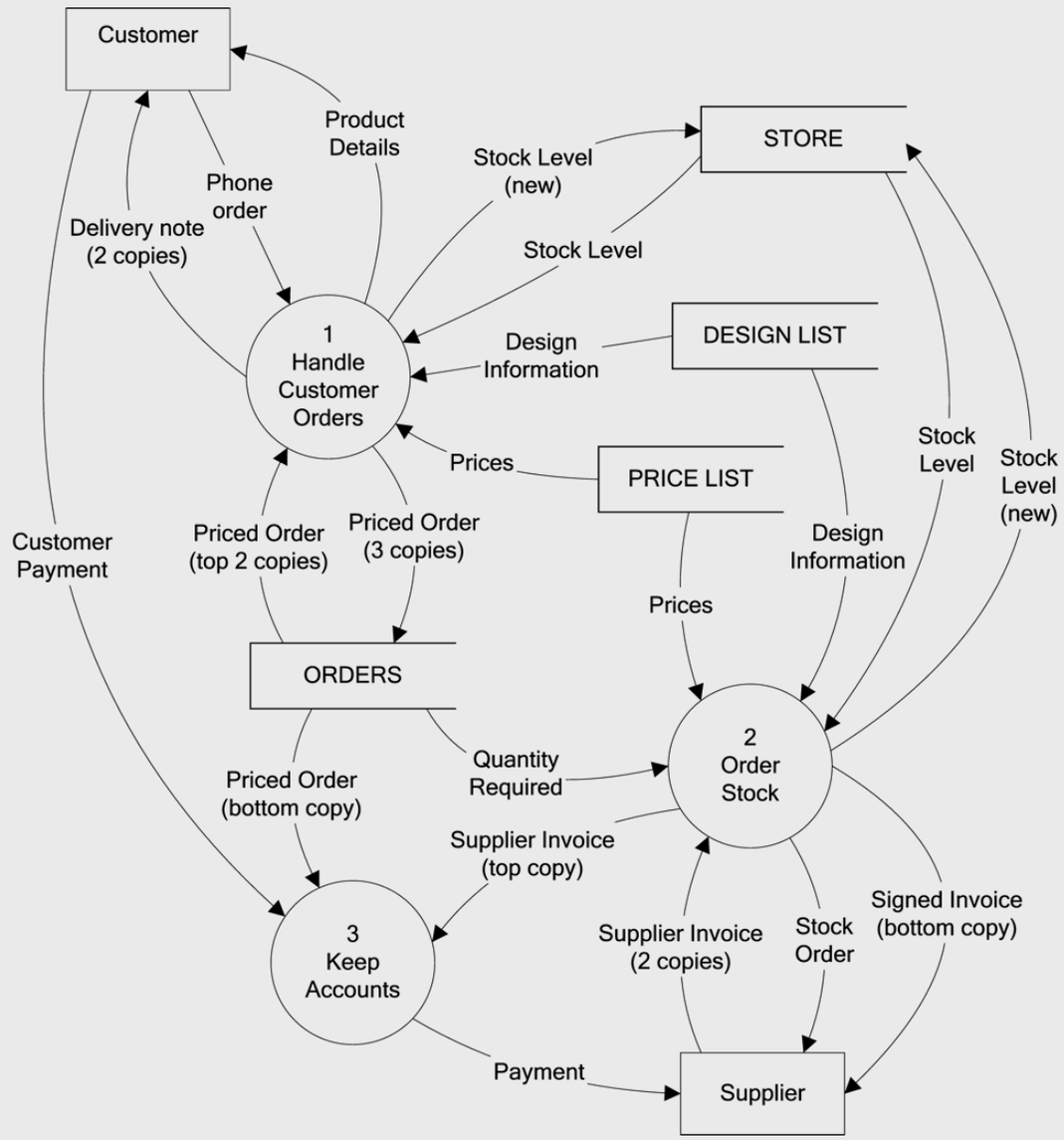


- The central process represents everything the company does

A decomposed DFD

- We can show how the main processes within the Stitch in Time company thus:

Each sub process can be further decomposed using more DFDs



end of chapter 8