State-Oriented Models

- Examples:
  - Automata (DFAs, NFAs, PDAs)
  - Turing Machines
- A finite state machine is a hypothetical machine that can be in only one of a given number of states at a specific time.
- In response to a stimulus, the machine performs an action (possibly generating output) and changes state.
- State diagrams capture the behavior of the machine.

Toggle Switch State Diagram
Microwave Oven State Diagram

State Machines

• Actions are processes that occur “quickly” and are not interruptible.
• A single state in a state diagram may be decomposed into several states in a less abstract view.

Function-Oriented Models

• The specification of the external behavior of a system is primarily a description of how the system outputs relate to the system inputs. The classic example for function-oriented techniques is the mathematical function.
Function-Oriented Models (2)

- All actions are atomic, effectively instantaneous
- There are no observable intermediate states
- No action, once started, is affected by any external state

Function F maps a value X to a value Y: F(X) = Y.

Process P consumes input X and produces output Y.

Function-Oriented Models (3)

• Sequential processes may be modeled as composition of functions:

\[ F_3 (F_2 (F_1 (X))) = Y \]

Data Flow Diagrams

- **Double Square**: Source or destination of data
- **Rounded Rectangle**: Flow of data
- **Open-ended Rectangle**: Process which transforms a flow of data
- **Open-ended Arrow**: Store of data
What Are We Modeling?

- State machine and data flow diagrams can be used to help specify the design of a software system. In this case they are showing state transitions or data flows within the system to be built.

- The same notations could be used during analysis to document state transitions or data flows in the real world (the problem domain).

- For example, employee timecards might be a database with timecard data, or it might be a stack of physical cards. Calculate Gross Pay might be a function of the system, or it might be a process performed by hand.

What Are We Modeling? (2)

- These same notations could even be used to specify requirements of the system:

- Here we are showing only data flows into and out of the system. We are not specifying what happens within the system.
Process Frameworks

- Define the basic elements of a process model and how they relate to each other.
- Define how process models are decomposed into greater detail.

IDEF0/SADT Process Model

- Input
  - Arrow entering the left side of the box are inputs. Inputs are transformed or consumed by the function to produce outputs.
- Control/Constraint
  - Arrows entering the box on the top are controls. Controls specify the conditions required for the function to produce correct outputs.
- Output
  - Arrows leaving the box on the right are outputs. Outputs are the data or objects produced by the function.
- Resource/Mechanism
  - Identify some of the means that support the execution of the function.

IDEF0 Example (1a)
IDEF0 Example (1b)

Select Vendor

Select Color

Arrange Financing

Paint Car

Painting Requirements

Color Options

Phone book

Advertisements

Recommendations

Owner

Vendor

IDEF0 Example (2a)

Build the System

Requirements

Schedule

Budget

Standards

Code

Documentation

Staff

Tools

IDEF0 Example (2b)

User Analysis

Requirements Elicitation

Requirements Scope

Interview Results

Customer License

Problem Statement

User Specialist

Evaluation Results

Vendor Characteristics

Survey Results

Review Guidelines

Requirements Specification

Requirements Review

Requirements Scope

Requirements Scope

Requirements Scope

Requirements Scope
Humphrey’s Process Architecture

- Unit cells are defined to accomplish specific tasks
- Each cell has required
  - Entry conditions: to met before task initiation
  - Exit conditions: results produced
  - Task definition: standards, procedures, responsibilities.
  - Feedback: in from other cells, out to other cells
  - Measurements: task, output, and feedback measures

Humphrey’s Architecture (2)

<table>
<thead>
<tr>
<th>Requirements Issues</th>
<th>Design Issues</th>
<th>Implementation Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Issues</td>
<td>Design Issues</td>
<td>Implementation Issues</td>
</tr>
</tbody>
</table>

Humphrey’s Architecture (3)

<table>
<thead>
<tr>
<th>Cell</th>
<th>001</th>
<th>002</th>
<th>003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry</td>
<td>Approved requirements, changes, and development plan</td>
<td>Inspected and approved design and changes</td>
<td>Inspected and approved code and changes</td>
</tr>
<tr>
<td>Exit</td>
<td>Inspected and approved design and changes</td>
<td>Inspected and approved code and changes</td>
<td>Inspected, tested, and approved software</td>
</tr>
<tr>
<td>Feedback In</td>
<td>Design issues</td>
<td>Implementation issues</td>
<td></td>
</tr>
<tr>
<td>Feedback Out</td>
<td>Requirements issues</td>
<td>Requirements and design issues</td>
<td>Requirements, design, and implementation issues</td>
</tr>
<tr>
<td>Task</td>
<td>Design</td>
<td>Implementation, inspection, and unit test</td>
<td>Testing, integration, function, system, and acceptance</td>
</tr>
<tr>
<td>Measures</td>
<td>Changes, errors, design, document pages</td>
<td>Changes, errors, design, document pages</td>
<td>Changes, errors, design, document pages, test suite</td>
</tr>
</tbody>
</table>
Programming Paradigms

- Imperative Paradigm
  - Program is a sequence of instructions (commands) that change the state of a program
- Declarative Paradigm
  - Program is a set of declarations that provide the system with information

Programming Languages

- Imperative
  - Procedural
  - Object-oriented
  - Parallel Processing
- Declarative
  - Logic
  - Functional
  - Database
Functional Languages

- In functional programming languages, a program is expressed as a set of (mathematical) functions. The main program is the function that is evaluated when the program is executed.
- Functions may be defined by composition of other existing functions:
  \[
  \text{quadruple}(x) = \text{double}(\text{double}(x))
  \]
  \[
  \text{double}(x) = \text{add}(x, x)
  \]

Functional Programming

- Advantages
  - Precise definition of the problem
  - Relatively simple correctness proofs
  - Direct and formal mapping of specification to code
  - High degree of modularity and abstraction
- Disadvantages
  - Slow execution
  - Difficult for compilers to optimize code without losing formal correctness
  - Side effects (such as i/o) are not modeled

Logic Languages

- A logic program consists only of rules and facts. The program is executed by asking a query:
  **Fact:** append ([], Y, Y)
  **Rule:** append (AB, C, AD) if append (B, C, D)
- Queries:
  append (ab, X, abcd) ?
    \[ X = \text{cd} \]
  append (X, dc, abcd) ?
    \[ \text{No.} \]
Logic Programming

• Advantages
  – Programmer does not specify how the problem is to be solved
  – The focus is on understanding the problem

• Disadvantages
  – Execution speed is slow