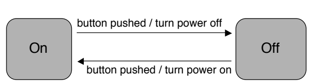
Computational Paradigms and Process Frameworks

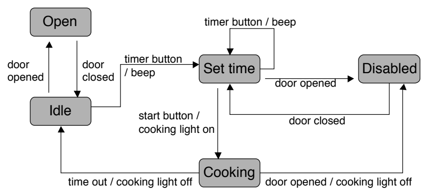
State-Oriented Models

- 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

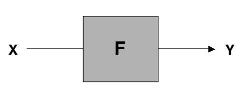


State Machines

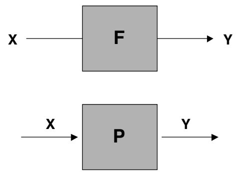
- Actions are processes that occur "quickly" and are not interruptible.
- A single state in a state diagram may be decomposed into several states in a less abstract view.

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)



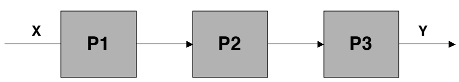
Function F maps a value X to a value Y. F(X) = Y.

Process P consumes input X and produces output Y

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

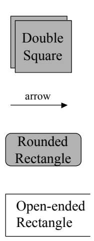
Function-Oriented Models (3)

- Sequential processes may be modeled as composition of functions:




```
F3(F2(F1(X))) = Y
```

Data Flow Diagrams



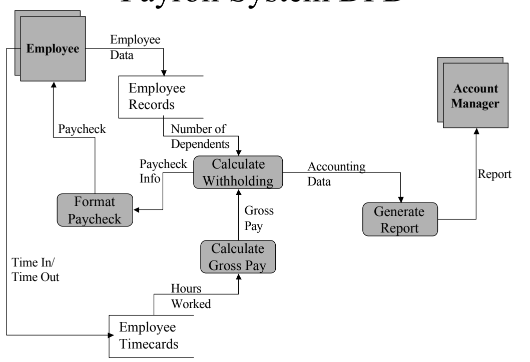
Source or destination of data

Flow of data

Process which transforms a flow of data

Store of data

Payroll System DFD

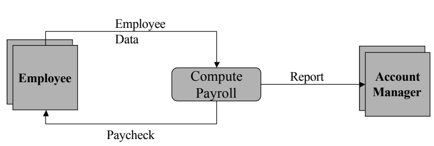


What Are We Modeling?

- 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.

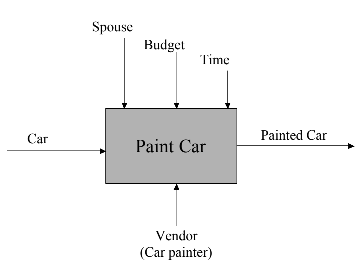
Process Frameworks

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

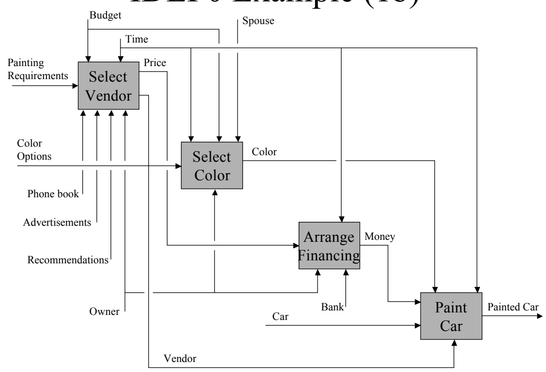
IDEF0/SADT Process Model

- 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.

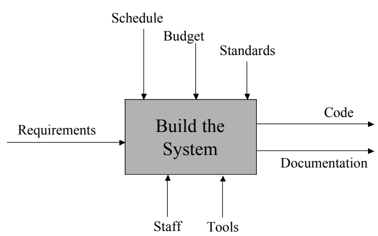
IDEF0 Example (1a)



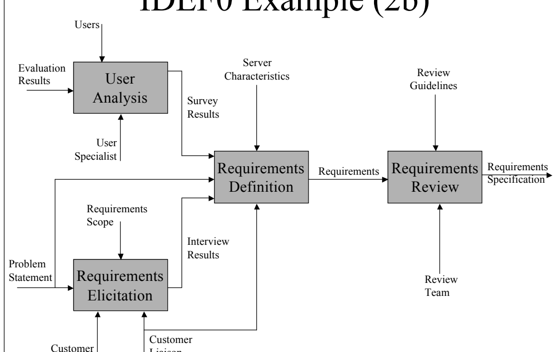
IDEF0 Example (1b)



IDEF0 Example (2a)

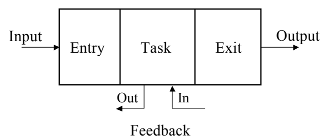


IDEF0 Example (2b)

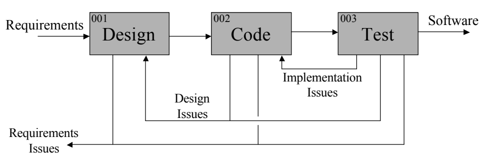


Humphrey’s Process Architecture

- 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)



Humphrey’s Architecture (3)

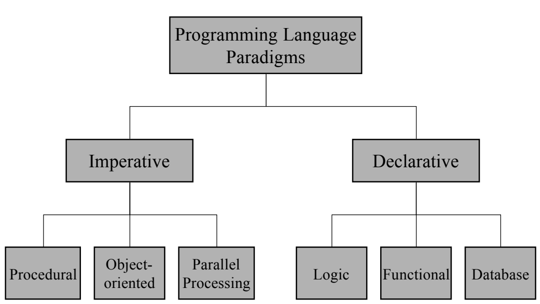
Cell	001	002	003
Entry	Approved requirements, changes, and development plan	Inspected and approved design and changes	Inspected and approved code and changes
Exit	Inspected and approved design and changes	Inspected and approved code and changes	Inspected, tested, and approved software
Feedback In	Design issues	Implementation issues	
Feedback Out	Requirements issues	Requirements and design issues	Requirements, design, and implementation issues
Task	Design	Implementation, inspection, and unit test	Testing, integration, function, system, and acceptance
Measures	Changes, errors, design, document pages	Changes, errors, design, document pages	Changes, errors, design, document pages, test suite

Programming Paradigms

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



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:

quadruple(x) = double (double (x))

double (x) = 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 = cd
append (X, dc, abcd) ?
    □  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