Design notations

Dynamic
- Data flow diagrams (DFDs).
- State transition diagrams (STDs).
- Statecharts.
- Structure diagrams.

Static
- Entity Relationship Diagrams (ERDs).
- Class diagrams.
- Structure charts.
- Object diagrams.
Data Flow Diagrams (DFDs)

- A notation developed in conjunction with structured systems analysis/structured design (SSA/SD).

- Used primarily for pipe-and-filter styles of architecture.

- Graph–based diagrammatic notation.

- There are extensions for real-time systems that distinguish control flow from data flow.
DFDs: Diagrammatic elements

- **external entity**: A producer or consumer of information that resides outside the bounds of the system to be modeled.

- **process**: A transformation of information (a function) that resides within the bounds of the system to be modeled.

- **data object**: A data object; the arrowhead indicates the direction of data flow.

- **data store**: A repository of data that is to be stored for use by one or more processes; may be as simple as a buffer or queue or as sophisticated as a relational database.
E.g.: Level 0 for *SafeHome*

- Control panel
- User commands and data
- Display information
- Control panel display
- Sensor status
- Telephone number tones
- Telephone line
- SafeHome software
- Alarm type
- Alarm
E.g. (cont’): Level 1 (SafeHome software)
E.g. (cont’): Level 2 (*monitor sensors*)

- **sensors**
  - sensor status
- **read sensors**
  - sensor id, type
- **sensor status**
- **configuration data**
- **format for display**
  - sensor information
  - configuration information
- **assess against setup**
  - sensor id, type
  - location
  - configuration information
- **generate alarm signal**
  - alarm data
  - telephone number
- **alarm**
  - telephone number tones
- **dial phone**
  - format for display
- **telephone line**
  - alarm type
State Transition Diagrams (STDs)

- Used for capturing *state transition behavior* in cases where there is an intuitive finite collections of states.

**E.g.:** a telephone call!

- Derives from the notion of a finite state automaton.

- Graph–based diagrammatic notation.
  - Labeled nodes correspond to states.
  - Arcs correspond to transitions.
  - Arcs are labeled with events and actions (actions can cause further events to occur).

- Describes a single underlying process.

CS646: Software Design and Architectures
E.g.: Aircraft landing behavior (from text)

- **in flight**
  - **STACK** join at set height
  - **ABORT LANDING** climb to set height
  - **CLEARED TO LAND** select path; adjust flaps; lower undercarriage

- **landing approach**
  - **TOUCH DOWN** reverse engine thrust; brake

- **on runway**
  - **ABORT TAKEOFF** close throttle; brake; turn off runway
  - **PARK** taxi to stand
  - **CLEARED FOR TAKEOFF** position on runway; open throttles

- **on ground**

---

CS646: Software Design and Architectures
Statecharts

Developed by David Harel.

A generalization of STDs: States can have zero, one, two or more STDs contained within.

Related to Petri nets.

Higraph–based diagrammatic notation.
- Labeled nodes correspond to states.
- Arcs correspond to transitions.
- Arcs are labeled with events and actions (actions can cause further events to occur).

Describes one or more underlying processes.
E.g.: *Teletext television set* (from text)
E.g. (cont’d): *Teletext television set*

![Diagram of Teletext television set process](image-url)
E.g. (cont’d): *Teletext television set*

```plaintext
off

(power on)

powered

(power off)

standby

(standby)

display

(any button press)
```

CS646: Software Design and Architectures
E.g.: *Aircraft landing behavior* (from text)
E.g.: Describing more than one process
Structure Diagrams

- Used in Jackson Structured Programming.

- Used to describe several kinds of things.
  - Ordered hierarchical structure.
  - Sequential processing.

- Based on the idea of regular languages.
  - Sequencing.
  - Selection.
  - Iteration.
E.g.: Ordered hierarchical structure
E.g.: Sequential processing

- print bank statement
  - print pages
    - print page header
    - print page body
    - print page summary
  - print a/c transactions
    - print date
    - print details of transaction
    - print amount transferred
      - print in debit column
      - print in credit column
Entity Relationship Diagrams (ERDs)

Slides on this are in a separate file.
Class Diagrams

- Derived from ERDs.
- Limited to binary relationships.

Diagrammatic elements:

- Binary relationship:
  \[
  \begin{array}{c|c}
  \text{RoleName} & \text{RoleName} \\
  \text{NumberConstraint} & \text{NumberConstraint}
  \end{array}
  \]
  
- Specialization
- Generalization
- Class definition

CS646: Software Design and Architectures
E.g.: University personnel

Person
  Name: String

Student
  Year: Integer

Teacher
  Salary: Integer

Teacher
  taughtby 1..1

Graduate
  Enroll(Course)*

Course
  Name: String

Course
  takes 0..4

Teacher
  teaches 1..6
Structure Charts

- Based on the fundamental notion of a *module*.
- Used in *structured systems analysis/structured design* (SSA/SD).
- Graph–based diagrammatic notation: a structure chart is a collection of one or more node labeled *rooted directed acyclic graphs*.
  - Each graph is a process.
  - Nodes and modules are synonymous.
  - A directed edge from module M1 to module M2 captures the fact that M1 directly uses in some way the services provided by M2.

**Definitions:** The *fan-in* of a module is the count of the number of arcs directed toward the module. The *fan-out* of a module is the count of the number of arcs outgoing from the module.
E.g.: *SafeHome* (*monitor sensors*)

M1
- monitor sensors executive

M2
- acquire response info

M3
- read sensors

M4
- establish alarm conditions

M5
- alarm output controller

M6
- produce display

M7
- generate alarm signal

M8
- set up connection to phone net

M9
- generate pulses to line

CS646: Software Design and Architectures
E.g.: *SafeHome* (interact with user)

- **M10** user interaction executive
  - **M11** read user command
  - **M12** invoke command processing
    - **M13** system config. controller
    - **M17** activate/deactivate system
      - **M14** read system date
      - **M15** build configuration file
    - **M18** password proc. controller
      - **M19** read password
      - **M20** compare password with file
        - **M21** password output controller
          - **M22** process invalid message
      - **M16** monitor sensors executive
Object Diagrams

- Derived from structure charts.
- Much in common with class diagrams.

```
[ClassName]: ObjectName

AttributeName: Type
...
AttributeName: Type

MethodSignature
...
MethodSignature
```

object definition  delegation  utilization
E.g.: SafeHome (interact with user)

M10 user interaction executive

M11 read user command
M12 invoke command processing

M13 system config. controller
M14 read system date
M15 build configuration file

M16 monitor sensors executive

M17 activate/deactivate system
M18 password proc. controller
M19 read password
M20 compare password with file

M21 password output controller
M22 process invalid message