Lecture 08: Software Design and Implementation
[CMFU-203]
September 26, 2017

Sameer S Pradhan
Vassar College
Representing Software Designs
→ Viewpoints

→ Structural representations
  ➤ e.g. dependency graphs

→ Functional representations
  ➤ e.g. dataflow diagrams

→ Behavioral representations
  ➤ e.g. statecharts

→ Data Modeling representations
  ➤ e.g. entity relationship diagrams
Representing Designs

From abstractions to systems

- abstractions allow us to ignore implementation details of procedures and data structures
- for large systems we need to abstract away even more detail
- we need to represent higher level abstractions

Design representations will:

- help us to see the big picture
- allow us to communicate our designs with others
  - customers, managers, other developers, ...
  - people with varying technical expertise
- allow us to measure various quality attributes
  - completeness, consistency, complexity, ...
Viewpoints \textit{(a.k.a. “projections”)}

\textbf{A viewpoint}
\begin{itemize}
\item tells you which details you can ignore when forming an abstraction
\item defines which details are relevant and which are not
\item a viewpoint has:
  \begin{itemize}
  \item an owner (the person interested in this abstraction)
  \item a domain (the area of interest)
  \item a representation scheme
  \end{itemize}
\end{itemize}

\textbf{Example: Building a house}
\begin{itemize}
\item Useful viewpoints:
  \begin{itemize}
  \item the architect’s viewpoint (plan views, elevations, etc)
  \item the plumber’s viewpoint (routing diagrams for pipework, fittings layouts, etc)
  \item the electrician’s viewpoint (wiring diagrams, etc)
  \item the buyer’s viewpoint (artist’s impression, floorplans, etc)
  \item etc...
  \end{itemize}
\item These must all be consistent eventually!
\end{itemize}

\textbf{Viewpoints can overlap}
\begin{itemize}
\item Some aspects of a design are common to several viewpoints
\end{itemize}
Key Software Design Viewpoints

Source: Adapted from Budgen, 1994

→ Structural viewpoints
  ✷ domain: static properties (structure) of the software
  ✷ representations: structure charts, dependency graphs, etc.

→ Functional viewpoints
  ✷ domain: the tasks performed by the software, information flow
  ✷ representations: dataflow diagrams, procedural abstractions, etc.

→ Behavioral viewpoints
  ✷ domain: cause and effect within the program
  ✷ representations: state transition diagrams, statecharts, petri nets, etc.

→ Data-modeling viewpoints
  ✷ domain: the data objects and the relationships between them
  ✷ representations: entity relationship diagrams, object hierarchies

Ownership?

✦ Each of these viewpoints will be of interest to different people
  ➢ e.g. structural viewpoints are of interest to managers for planning purposes
  ➢ e.g. functional viewpoints are of interest to requirements analysts and users
Notational forms

→ Text
  - often hard to see the big picture
  - natural language is ambiguous
  - best used in small chunks (e.g. for executive summaries)

→ Diagrams
  - good for showing relationships and structure...
  - ...if they're kept simple:
    - small number of symbols (e.g. 2 types of box, 2 types of arrow)
    - must represent an abstraction (e.g. a flow chart contains nearly all the detail of code, so is not an abstraction)
    - should be easy to sketch informally!

→ Mathematical Expressions (formal specifications)
  - very precise, very concise
  - but require much training
  - cannot (yet?) express all viewpoints (e.g. timing is difficult to express)
Structural notations

See also: van Vliet 1999, section 11.1.5 and 11.2.2

→ Objects modeled
  ◦ usually program components
    ✷ compilation units,
    ✷ modules,
    ✷ procedures
    ✷ ...

→ Relationships modeled
  ◦ structural relationships between components
  ◦ static relationships only
    ✷ “calls/controls”
    ✷ “uses”
    ✷ ...

Note: structural notations deal with structure of the program, not structure of the data.

→ Example notations
  Structure charts
    ✷ hierarchical decomposition of program

  Dependency graphs
    ✷ show the (static) control flow
The Dependency Graph

See also: van Vliet 1999, pp311-314

Notes:
- all edges must be directed
- all nodes must be labelled with the name of the procedure
- only one edge between any two nodes (no matter how many times the procedure is called)
- recursive procedures (& data abstractions) use themselves

Useful for:
- debugging, integration, measuring coupling
Functional notations

See also: van Vliet 1999, sections 11.2.1 and 11.2.2

→ Objects modeled
   ✎ Program components
      ➢ modules,
      ➢ procedures,
   ✎ Processes
      ➢ these do not necessarily correspond to components of the program

→ Relationships modeled
   ✎ information flow
   ✎ inputs and outputs
      ➢ “communicates with”.
      ➢ “sends data to”
      ➢ “received data from”

→ Example notations
   ✎ Dataflow diagrams
      ➢ show processes that transform data
   ✎ Procedural abstractions
      ➢ (although these combine structural viewpoint info too!)
   ✎ Pseudo-code
The Dataflow Diagram (DFD)

See also: van Vliet 1999, pp322-325

Key

- process
- dataflow (no control implied)
- data store
- external entity
- system boundary

→ Notes:

- every process, flow, and datastore must be labeled
- representation is hierarchical
  - each process will be represented separately as a lower level DFD
- processes are normally numbered for cross reference
- processes transform data
  - can’t have the same data flowing out of a process as flows into it
Behavioral notations

See also: van Vliet 1999, sections 9.3.2 and 12.2.2

→ Objects modeled
  ¬ Dynamic properties
    ➢ events, states, actions, conditions

→ Relationships modeled
  ¬ cause and effect
  ¬ sequencing / parallelism

→ Example notations
  ¬ State Transition Diagrams
    ➢ model the program as a finite state machine

→ Statecharts
  ¬ like an STD but with superstates and conditional transitions

→ Petri nets
  ¬ for modeling process synchronization
Statecharts

Source: Adapted from Easterbrook & Nuseibeh, 1996

→ Notes:

- all states and transitions must be labeled
- transitions may be conditional (conditions shown in brackets)
- states can be grouped into superstates:
  - transitions out of superstates may be taken from any substate
  - transitions into superstates go to the default substate
Data modelling notations

See also: van Vliet 1999, sections 9.3.1 and 12.2.1

→ Objects modeled
  - any kind of data
    - data types,
    - objects,
    - attributes of objects,
    - classes,

→ Relationships modeled
  - compositional
    - “part of”
    - “consists of”
  - classification
    - “is a kind of”

→ Example notations
  - Entity Relationship Diagrams
    - used in requirements modeling

  - Class diagrams
    - shows data abstraction hierarchy
    - Note: in OOD, is used as a structural notation for the program!!!
Entity Relationship Diagram

See also: van Vliet 1999, section 9.3.1

Key

- **film**: entity
- **age**: attribute
- **cast**: relationship
- 1-to-1
- 1-to-many
- many-to-many

→ Notes:

- relationships relate entities, not their attributes
- there is no standard way to show the cardinality of relationships
Summary

→ Viewpoints help in creating abstractions
  ✷ a viewpoint is an abstraction created for a particular purpose by a particular person
  ✷ the viewpoint tells you what information to ignore when creating the abstraction
  ✷ each viewpoint has a suitable representation scheme

→ Useful software design viewpoints:
  ✷ structural
  ✷ functional
  ✷ behavioral
  ✷ data modeling

→ But a notation is not enough...
  ✷ you need a method to tell you how to use it.
  ✷ We’ll see some sample methods later in the course.
Structured Modeling
Structured Modeling Methods

→ Basics of Structured Analysis
  - Notations used
  - Modeling Process

→ Variants
  - SADT
  - SASS
  - SSADM
  - SRD

→ Advantages and Disadvantages
Structured Analysis

→ Definition

Structured Analysis is a data-oriented approach to conceptual modeling

Common feature is the centrality of the dataflow diagram

Mainly used for information systems

variants have been adapted for real-time systems

Modeling process:

Abstract (essential functions)

Concrete (detailed model)

2. Current logical system

3. New logical system

1. Current physical system

4. New physical system

Model of current physical system only useful as basis for the logical model

Distinction between indicative and optative models is very important:

Must understand which requirements are needed to continue current functionality, and which are new with the updated system
Central Concepts

*Source: Adapted from Svoboda, 1990, p257*

- **Process (data transformation)**
  - activities that transform data
  - related by dataflows to other processes, data store, and external entities.

- **Data flow**
  - indicate passage of data from output of one entity to input of another
  - represent a data group or data element

- **Data store**
  - a place where data is held for later use
  - Data stores are passive: no transformations are performed on the data

- **External entity**
  - An activity outside the target system
  - Acts as source or destination for dataflows that cross the system boundary
  - External entities cannot interact directly with data stores

- **Data group**
  - A cluster of data represented as a single dataflow
  - Consists of lower level data groups, or individual elements

- **Data element**
  - a basic unit of data
Modeling tools

*Source: Adapted from Svonoda, 1990, p258-263*

→ **Data flow diagram**
  - Context diagram (“Level 0”)
    - whole system as a single process
  - intermediate level DFDs decompose each process
  - functional primitives are processes that cannot be decomposed further

→ **Data dictionary**
  - Defines each data element and data group
  - Use of BNF to define structure of data groups

→ **Primitive Process Specification**
  - Each functional primitive has a “mini-spec”
  - these define its essential procedural steps
  - Expressed in English narrative, or some form of pseudo-code

→ **Structured Walkthrough**
Example Data Dictionary

Mailing Label =
   customer_name +
   customer_address

customer_name =
   customer_last_name +
   customer_first_name +
   customer_middle_initial

customer_address =
   local_address +
   community_address + zip_code

local_address =
   house_number + street_name +
   (apt_number)

community address =
   city_name + [state_name | province_name]

Example Mini-Spec

FOR EACH Shipped-order-detail
   GET customer-name + customer-address
   FOR EACH part-shipped
      GET retail-price
      MULTIPLY retail-price by quantity-shipped
      TO OBTAIN total-this-order
      CALCULATE shipping-and-handling
      ADD shipping-and-handling TO total-this-order
      TO OBTAIN total-this-invoice
   PRINT invoice
Alternative Process Model: SRD

Source: Adapted from Davis, 1990, p72-75

1. Define a user-level DFD
   - interview each relevant individual in the current organization
     - actually a role, rather than an individual
   - Identify the inputs and outputs for that individual
   - Draw an ‘entity diagram’ showing these inputs and outputs

2. Define a combined user-level DFD
   - Merge all alike bubbles to create a single diagram
   - Resolve inconsistencies between perspective

3. Define the application-level DFD
   - Draw the system boundary on the combined user-level DFD
   - Then collapse everything within the boundary into a single process

4. Define the application-level functions
   - label the inputs and outputs to show the order of processing for each function
     - i.e. for function A, label the flows that take part in A as A1, A2, A3,...
Later developments

→ Later work recognized that:
  - development of both current physical and current logical models is overkill
  - top down development doesn’t always work well for complex systems
  - entity-relationship diagrams are useful for capturing complex data

→ Structured Analysis / Real Time (SA/RT)
  - Developed by Ward and Mellor in the mid-80’s
  - Extends structured analysis for real-time systems
    - Adds control flow, state diagrams, and entity-relationship models

→ Modern Structured Analysis
  - Captured by Yourdon in his 1989 book
  - Uses two models: the environmental model and the behavioral model
    - together these comprise the essential model
  - Includes plenty of advice culled from many years experience with structured analysis
Real-time extensions

Source: Adapted from Svoboda, 1990, p269

Control line conditions

Line tension
Enable
Enable
Tension off
Enable
Tension of

material inlet

Line tension
Enable
Enable

Monitor Tension

Current tension

Controlling tension

Tension settings table

Inlet control

Control flow (continuous)
Control flow (discrete)
Control Transformation

Control Store

KEY
Evaluation of SA techniques

Source: Adapted from Davis, 1990, p174

→ Advantages

❖ Facilitate communication.
❖ Notations are easy to learn, and don’t require software expertise
❖ Clear definition of system boundary
❖ Use of abstraction and partitioning
❖ Automated tool support
  ➢ e.g. CASE tools provide automated consistency checking

→ Disadvantages

❖ Little use of projection
  ➢ even SRD’s ‘perspectives’ are not really projection
❖ Confusion between modeling the problem and modeling the solution
  ➢ most of these techniques arose as design techniques
❖ These approaches model the system, but not its application domain
❖ Timing & control issues are completely invisible
  ➢ Although extensions such as Ward-Mellor attempt to address this