Structured Design

Ideas Behind Structured Design

Structured design is a disciplined approach to design of
Computer systems and software

Comprised of following 5 aspects
1. Uses definition of problem to guide definition of solution
2. Seeks to attack complexity of contemporary problems by partitioning into
   modules and organizing modules into hierarchies
3. Uses variety of tools to render complex systems understandable
4. Offers set of strategies for developing design solution from well defined statement
   of problem
5. Offers set of criteria for evaluating quality of design

Simplifying a System

Partition into Modules

First step in controlling complexity
Partitioning into modules
Goals
• Each module should solve one well defined piece of the problem
• System should be partitioned so that function of each module is easy to
  understand
• Partitioning should be done so that connections between modules only
  introduced because of connection between pieces of problem
• Partitioning should assure that connections between modules are as
  independent as possible

Organize Modules into Hierarchies

Familiar approach
Divide and conquer
Most natural systems divided into hierarchies of stable units

Tools

Graphical Tools
Structured design uses tools
Especially graphical ones to render systems more understandable
If structured analysis precedes structured design
Analyst will present designer with structured specification
Such specification is
Output from analysis
Input to design
Example

Data and Control Flow diagrams
Decision Trees
Decision Tables
Data Dictionaries
Data Access Diagrams

Pseudo Code
Informal and flexible programming language
Not intended to be executed on a machine
Used to organize designer’s thoughts prior to implementation

Originally developed for structured programming
Can be used in structured design as well
Aides in clarifying internal
   Procedures and flow of control in modules

Example

Serial Transmit Routine

transmit
{
    select window 0
    save status info

    // transmit a character if there is a character in the buffer
    if (bufferBegin != bufferEnd)
    {
        outputBuffer <- character from transmitBuffer
        // make the buffer circular

        increment beginBuffer
        if (beginBuffer > transmitBuffer)
            beginBuffer

        clear transmit status bit
    }
}
**Structure Chart**

Structure chart illustrates partitioning of system into modules
Graphically illustrates
  - Hierarchy
  - Organization
  - Communication

Structure chart is chief structured design tool

Has many advantages
  - Graphic
  - Partitionable
  - Rigorous but flexible
  - Valuable blueprint for implementing the system
  - Helps to document system
  - Aid in maintaining and modifying the system

**Top level of structure chart**
  - Paints broad picture of the system
  - Shows major functions of system and their interfaces
  - Contains very little detailed information

**Bottom level contains more of detail**
  - May also hide such detail in descriptions of modules

Should not be something that is done once at start of design
  - Should be a living document through lifetime of product

**Elaborating a Design Solution**

Structured design offers set of strategies for
  - Developing design solution
  - From well-defined statement of problem

If problem expressed in set of data / control flow diagrams
  - Approach offers strategy for smoothly developing a good design
Evaluating a Design Solution

Structured design offers set of effectively and empirically justified criteria for evaluating quality of design solution with respect to problem being solved. Same criteria can be used when necessary to improve deficient design.

What Structured Design is Not

- Structured Design is not structured programming
  Generally agreed structured programming
  Set of reasonable methods for developing understandable and reliable source code
  Does not work well for large systems
  Has no inherent strategy for addressing complexity in large systems

Structured design and structured programming are not incompatible

- Structured design is not a means to a good problem specification
  Structured design takes as its input
  Statement of what the system is supposed to accomplish
  Without such a statement
  Cannot move forward
• Structured design is not modular programming or top down design  
  Most of work is done with modules  
  Strong focus on top down organization  
  Such ideas developed during early 70's  
  Structured design is a logical extension of such ideas

• Structured design is not solution to all world's problems  
  There are no guarantees  
  All design has  
    Difficulties  
    Pitfalls  
    False starts  
  Structured design  
    Should help in recognizing such problems early  
    Offers set of tools to help correct such problems

• Structured design does not stifle creativity  
  Truly creative designers can use  
    Organization and discipline of structured design  
    To further creativity

  Permits one to build on work of others  
    Rather than continually re-inventing the wheel

**Tools of Structured Design**

**The Structure Chart**

  Major tool used in structured design is *structure chart*  
  Used to depict overall structure of the system

**Elements**

  Modules  
  Represent an encapsulation of a piece of functionality  
    Indicated by a box

  Pre-defined library routines identified by box with vertical lines

  Some modules may *call* other modules  
    Indicated by an arrow

  Some modules may *send data* to other modules
Indicated by an open arrow

Some modules may send flags to other modules
Indicated by an closed arrow

Communication Between Modules
We show communication between modules using such symbols

Open arrow is called data couple

Distinguished from flag because
- Data is processed
- Flag not really processed
  Signifies an event has happened
- Data relates to problem
- Flag one step removed

Example
Consider simple input output task
At high level
Identify 4 modules
  Top level task
  Transmit module
  Receive module
  Code conversion library routine
Identify data
   Received from receive module
   Transmitted by transmit routine
   Sent to and received from library routine

Identify flags
   Received data available
   Transmit task done
   Conversion task done

Can bring together in simple structure chart

Module Specification
   Structure chart shows only high level details
      Deliberately suppresses details
   Ultimately must confront details
      System must be built
   Will examine two ways to specify
      Module Interface Specification
      Specification by pseudo code
   Let's look at each

Module Interface Specification
   Method permits one to specify
      Function of module
      Without getting into too much detail
   Comprises several things
Specifies what inputs will be provided
Identifies what output is expected from module
Describes function to be carried out
    Should be stated in simple sentences
    Establish relationship between input and output
May choose to supplement with
    Tables
    Drawings
    Graphs
    Formulas

Example
Let's examine the IO task diagrammed above

*Interface Specification for IO Task*
Module: System I/O Module

Purpose: Receive data from and transmit data to outside world

Uses: Measured Data

Returns: Input Commands
         Transmission complete

Functional Details:

1. Accept measured data, translate to ASCII, transmit out.
2. When transmission complete signal done = 1.
3. Receive formatted measurement commands from outside world.
   Convert from ASCII to internal format 3.4A.
4. When command data is available, signal Input command = 1

*Specification by Pseudocode*
Pseudocode is much more detailed method to specify module
Begins to move toward how to do from what to do

Pseudocode
Informal language
    Similar to structured English
    Is a tool of the designer rather than user
Since no formal specification
    One can make it look as much like real code as desired

Much more detailed than interface specification
Consequently much less margin for error
When translating into actual code
Allows flexibility but need to be careful
   Not to turn pseudocoding into end goal

Example

Pseudocode Specification for IO Task

Module: System I/O Module
Purpose: Receive data from and transmit data to outside world
Uses: Measured Data
Returns: Input Commands
         Transmission complete

if (transmit)
   begin transmit
      get measured data
      for each character c
         c <- ascii(c)
         transmit (c)
      end for
      complete <- 1
   end begin

else if receive
   while (not end of message)
      receive (c)
      c <- decimal (c)
   end while
   inputCommand <- 1
end if
Pseudocode works as excellent programming tool
    Allows one to try out ideas at any level of detail
    Don’t need to be worried about constraints of programming language

Coupling
Now examine interdependence between modules
Called coupling
Objective
    Minimize coupling
    Want to make modules as independent as possible

Low coupling between modules
    Indicates well partitioned system

Achieved in 3 ways
    1. Eliminate unnecessary relationships
    2. Reduce the number of necessary relationships
    3. Ease tightness of necessary relationships

Reducing coupling means reducing complexity of module interconnections
Approaches
    1. Create narrow (as opposed to broad) connections
       Breadth is measure of number of interconnections between modules
       Reduce the number of pieces of data that must flow between modules
    2. Create direct vs indirect connections
       Don’t require one module to go through second to get data from third
    3. Create local rather than remote connections
       Have the connection with a second module
       Specified in parameter list
       Rather than through global data somewhere else in program
    4. Create obvious rather than obscure connections
       Express information in natural and expected way
    5. Create flexible rather than rigid connections
       Don’t hard code parameters to
       Particular memory location
       Specific data value
Cohesion

Idea related to coupling is cohesion

Coupling addresses partitioning a system
Cohesion addresses bringing things together

We stress modularity and encapsulation
Cohesion is measure of strength of functional relatedness
Elements in a module

Goal
Create strong highly cohesive modules
Whose elements are genuinely and strongly related to one another
Conversely
Elements should not be strongly related to elements in another module

Want to maximize cohesion and minimize coupling

Let’s look at kinds of cohesion

Functional cohesion
Functionally cohesive module
Contains elements that all contribute to execution of
One and only one problem related task

Sequential Cohesion
Sequentially cohesive module
Contains elements that are involved in activity
Producing output data
That becomes input data to immediately successive task

Example
module formulate and cross validate data
uses raw data
format into raw record
cross validate fields in record
return formatted and cross validated record
end module

Communicational Cohesion
Communicational cohesive module
Contains elements that are involved in activity
Use the same input data

Example
module parse measurement command
uses raw data
find header field
find message length
find command
check parity
compute parity
return command or parity error

end module

Procedural Cohesion
Procedurally cohesive module
Contains elements that are involved in
Different and potentially unrelated activity
In which control flows from one activity to the next

Example
module read and modify record
uses output record
read input record
add parity to parity field
write output record
return

end module

Temporal Cohesion
Temporally cohesive module
Contains elements that are involved in activities
Related in time

Example
module initialize serial interface
updates wordCount, rBaudRate, tBaudRate, direction, parity
reset wordCount
set rBaudRate 9600
set tBaudRate 9600
set direction receive
set parity even
return

end module
Co- incidental Cohesion
Coincidentally cohesive module
Contains elements that are involved in activities
No meaningful relation to one another

Such cohesion – or lack of cohesion should not be used

Comparison

<table>
<thead>
<tr>
<th>Cohesion</th>
<th>Coupling</th>
<th>Cleanliness</th>
<th>Ease of Modification</th>
<th>Ease of Understanding</th>
<th>Ease of Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Sequential</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Fairly</td>
<td>Good</td>
</tr>
<tr>
<td>Communicational</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Procedural</td>
<td>Variable</td>
<td>Poor</td>
<td>Variable</td>
<td>Variable</td>
<td>Bad</td>
</tr>
<tr>
<td>Temporal</td>
<td>Poor</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Bad</td>
</tr>
<tr>
<td>Logical</td>
<td>Bad</td>
<td>Bad</td>
<td>Bad</td>
<td>Poor</td>
<td>Bad</td>
</tr>
<tr>
<td>Co- incidental</td>
<td>Bad</td>
<td>Poor</td>
<td>Bad</td>
<td>Bad</td>
<td>Bad</td>
</tr>
</tbody>
</table>

The Data / Control Diagram

Data / Control flow (DFD) diagram used to partition system
Show
Active components of system
Data and control interfaces between them
Sometimes known as bubble chart

Elements

DFD comprises four graphic elements
Data or Control flow
The Processes
Data Source / Sink
Data Store

Data or Control Flow

Similar to function call notation for structure chart
Indicated by an solid arrow for data flow

\[\text{———} \]

Indicated by an dashed arrow for control flow

\[\text{— — — — —} \]

As notation indicates
Data or control flow in direction of arrow
Processes

Processes modules or functions or tasks
Where the work is getting done

Indicated by labeled circles
Label identifies
  The name of the process
  The level in the hierarchy at which the process resides
    Level 0 - 1.0, 2.0, 3.0 etc.
    Level 1 - 1.1, 1.2, 1.3; 2.1, 2.2, 2.3; 3.1, 3.2 etc.
    Level 2 - 1.1.1, 1.1.2; 1.2.1, 1.2.2; etc.

Data Source / Sink
As name implies
  Source identifies where data originates
    For example file or input port
  Sink indicates where data goes
    For example file or output port

Drawn as labeled box with arrow to indicate direction of data flow

Data Store
Final piece is data storage
Indicates temporary store of data
  Time delayed repository of data

Represented by
  Two parallel lines or
  Two parallel lines closed on left hand end
Level 0 Communications System
Let's look at a simple example
Will draw Level 0 - top level - data flow diagram

Level 1 DFD
Would expand each of processes or tasks
In similar way