Chapter 3 Case Studies

• Purpose:
  To illustrate how we can use architectural principles to increase understanding of software systems

Case 1: Key Word in Context

• Problem introduction:
  * Introduced in a paper of Parnas (1972)
  * Well-known and widely used teaching device in software engineering
  * "Permuted" index for the Unix Man pages

• Analysis approaches:
  * Changes in the processing algorithm
  * Changes in data representation
  * Enhancement to system function
  * Performance
  * Reuse

• Prepared by Zhiying Zhao, Fall96
SWIC Problem

- System Accepts
  - Ordered Set of Lines
  - Each Line, Ordered Set of Words
  - Each Word, Ordered Set of Chars

- Any Line can be Circular Shifted by Removing First Word and Appending to End of Line

- Output KWIC
  - Listing of All Circular Shifts of all Line in Alphabetic Order
  - Why is this Useful?

- Utilized in Unix
  - Permutated Index of Man Pages
  - Ipr Print Files
  - Print Files Ipr
  - Files Print Ipr

- Search Against First Word of Permutated Index
FIGURE 3.1  KWIC: Shared Data Solution

FIGURE 3.2  KWIC: Abstract Data Type Solution
Solution 1: Main Program/Subroutine with shared data

- Function decomposition
  - Input
  - Circular shift
  - Alphabetizer
  - Output

- Coordination:
  - Main program sequences through each function component
  - Data communicates through shared storage

- Advantages:
  - Relatively good performance
  - Data can be represented efficiently
  - Distinct computational aspects are isolated

- Drawback:
  - Reuse ability to handle changes
  - What if change happens in data storage format?
  - What if change happens in overall algorithm
  - What if enhances system functions
Solution 2: Abstract Data Types

- Decomposition:
  - Input
  - Characters
  - Circular shift
  - Alphabetic shifts
  - Output

- Coordination:
  - No longer directly shared data
  - Interfaces are provided to permit other components to access data

- Advantages:
  - Support changes in individual modules
  - Support reuse

- Drawback:
  - Difficult to enhance systems functions
  - Performance penalties when add new functions
Solution 3: Implicit Invocation

- Function decomposition:
  - Input
  - Circularshift
  - Alphabetizer
  - Output

- Coordination:
  - Components integration based on shared data
  - Computing modules access data as list or set
  - Active data model. Computations are invoked implicitly as data is modified

- Advantages:
  - Easily supports functional enhancements to the system

- Drawbacks:
  - Poor support for reuse
  - Performance. Invocations are data-driven
  - Difficult to control the processing order of the implicitly invoked modules
  - Poor support for change in data representation
FIGURE 3.3  KWIC: Implicit Invocation Solution

FIGURE 3.4  KWIC: Pipe-and-Filter Solution
Solution 4: Pipes and Filters

- Filter decomposition
  - Input filter
  - Shift filter
  - Alphabetize filter
  - Output filter

- Coordination:
  - Each filter processes the data and sends it to the next filter
  - Distributed. Each filter can run whenever it has data

- Advantage:
  - Maintains the intuitive flow of processing
  - Support reuse.
  - Easy to add function
  - Amenable to modification

- Drawback:
  - Impossible to modify the design to support interactive system
  - Performance and data exchange format
Case 2: Instrumentation Software

- Problem:
  * To develop a domain-specific architecture style for oscilloscopes
  * How to support reuse across different oscilloscope products?
  * How to improve performance while the size of software increasing?

Solution 1: An Object-Oriented Model

- Data type:
  * Waveforms, signals, measurements, trigger modes,...

- Questions:
  * There is no overall model that explained how the types fit together
  * How to partition functionality?
  * Should measurements be associated with the types of data being measured?
  * Which object should the user interface interact with?
<table>
<thead>
<tr>
<th>Shared Data</th>
<th>Abstract Data Type</th>
<th>Implicit Invocation</th>
<th>Pipe and Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change in Algorithm</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Change in Data Rep</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Change in Function</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Performance</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Reuse</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

FIGURE 3.5  KWIC Comparison of Solutions

FIGURE 3.6  Oscilloscopes: An Object-Oriented Model

FIGURE 3.7  Oscilloscopes: A Layered Model
Solution 2: A layered Model

- Model layers:
  - Core layer: represents the signal-manipulation functions that filter signals as they enter the oscilloscope
  - Second layer: represents waveform acquisition. Signals are digitized and stored internally for later processing
  - Third layer: consists of waveform manipulation, including measurement, waveform addition, Fourier transformaton
  - Fourth layer: consists of display functions, responsible for mapping digitized waveforms and measurements to visual display
  - Fifth layer: user interface

- Comment:
  - Partition the functions of an oscilloscope into well-defined group
  - Wrong model for the application domain
  - Confliction between the boundaries of abstraction and the needs for interaction among the various functions
Solution 3: A pipe-and-filter model

- Decomposition
  * Signal transformers: used to condition external signals
  * Acquisition transformers: derive digitized waveforms from these signals
  * Display transformers: convert these waveforms into visual forms

- Comments:
  * The model in which oscilloscope functions were viewed as incremental transformers of data
  * It did not isolate the functions in separate partitions
  * Well corresponded with engineer’s view of signal processing
  * Allowed the clean intermingling and substitution of hardware and software components
  * How the user should interact with it?
FIGURE 3.8 Oscilloscopes: A Pipe-and-Filter Model

FIGURE 3.9 Oscilloscopes: A Modified Pipe-and-Filter Model
Solution 4: A modified Pipe-and-filter Model

- Description:
  - Based on solution 3
  - Accounts for user inputs by associating with each filter a control interface
  - Consider filters as having a "control panel", "high-order" functions

- Advantages:
  - Solve large part of user interface problem
  - Provides a collection of settings
  - Decouples the signal-processing functions from user interface

- Drawback:
  - Poor performance
  - Different filter has different speed

- Comment:
  - Overcome these drawbacks by introducing several 'colors' of pipes
• Summary:
  * Different architectural styles have different effects on the solution to a set of problems
  * Architectural design for industrial software must be adapted to special style to meet the domain needs