IMPORTANT NOTICE TO STUDENTS

These slides are NOT to be used as a replacement for student notes. These slides are sometimes vague and incomplete on purpose to spark a class discussion.
Intent

- alphabetical list of keywords
- keywords are displayed with context
  - context = surrounding words
- also known as “permuted index”
**KWIC Example**

**Input** is a set of titles

- Clouds are white
- Ottawa is beautiful

**Output** is permuted index where each keyword is surrounded by the context in which it was used

- are white Clouds
- beautiful Ottawa is
- Clouds are white
- is beautiful Ottawa
- Ottawa is beautiful
- white Clouds are
Architectural Concerns

Parnas – 1972

- changes in the processing algorithm
  - sequential, batch, and lazy (on-demand) line shifting
- changes in data representation

Garlan, Kaiser 7 Notkin – 1992

- enhancement to system function
  - user interaction, noise words
- performance
- reuse
Solution 1 (Parnas)

Main Program/Subroutine with Shared Data

- **functional decomposition**
  - components are subroutines
- **shared memory**
Solution 1 (Parnas)

Strengths

• centralized data
  – efficient representation of data
• modular decomposition
  – easy to add new components
• performance

Weaknesses

• resistant to change
  – consider the impact of data storage format
  – difficult to enhance the overall functionality
  – reuse of component is difficult
Solution 2 (Parnas)

Abstract Data Types

- data encapsulation within objects
  - data is shared via 'interface methods'
  - data access via component interface invocation
  - no direct data access
Solution 2 (Parnas)

Advantages

- data model can change with minimal impact
- handles **individual** component design changes well
  - algorithm and data are encapsulated in individual modules
- reuse as modules interact via defined interfaces

Disadvantages

- system evolution still a problem
  - to add new or change existing features may require
    - changes to existing components
    - addition of new components
Solution 3

Implicit Invocation

- similar to solution 1
- shared data via abstract interfaces (set, list etc.)
- with some main differences
  - components are invoked implicitly
    - e.g. when a line is added
  - interaction is based on 'active data model'
calls to circular shift and alphabetizer are implicit, and are the result of inserting lines
Solution 3

Advantages

- strong evolution path
  - functional enhancements are easy
  - new components can be attached and removed

- minimal component coupling/dependency
  - data events are the source of all interactions
Solution 3

Disadvantages

- shared data model → resistant to change
- difficult to control the ordering of processing
- poor reuse since system is based on the data model
  - components cannot be reused without data model
- requires more storage capacity
  - Why?
  - IS THIS REALLY A DISADVANTAGE?
Pipes & Filters

- four filters
  - input, shift, alphabetize, output
  - data sharing is restricted by pipes
Solution 4

Advantages

- intuitive flow of processing
- reuse
- evolution
  - new filters can be easily added
  - easy to modify
Solution 4

Disadvantage

- virtually impossible to support an interactive system
- **Is this a true pipes & filters?**
  - consider the data flow
- difficult to change the data model
  - what is the LCD data unit?
- performance may suffer
  - since each filter will have to __________
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Shared Memory</th>
<th>ADT</th>
<th>Implicit Invocation</th>
<th>Pipe &amp; Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>change in algorithm</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>change in data representation</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>change in functionality</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>performance</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>reuse</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

change in algorithm == change in overall system processing mechanism

change in functionality == change in components structure