Case Study on Information Hiding: The KWIC System

15-214:
Foundations of Software Engineering

Jonathan Aldrich


Some ideas from David Notkin’s CSE 503 class
Design Case Study: Key Word In Context (KWIC)

• “The KWIC [Key Word In Context] index system accepts an ordered set of lines, each line is an ordered set of words, and each word is an ordered set of characters. Any line may be "circularly shifted" by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order.”
  - Parnas, 1972

• Consider KWIC applied to the title of this slide

  Design Case Study: Case Study: Design Study: Design Case Key Word In Context (KWIC) Word In Context (KWIC) Key In Context (KWIC) Key Word Context (KWIC) Key Word In (KWIC) Key Word In Context (KWIC) Key Word In Context
KWIC Modularization #1

Master Control

Input → Circular Shift → Shifts → Alphabetize → Shifts → Output

function call

Lines

memory access

20 January 2015
KWIC Modularization #2

Master Control

Input

功能调用

Alphabetize

功能调用

Output

Line Storage

功能调用

Circular Shift

功能调用

getChar(r,w,c)

setChar(r,w,c,d)

cschar(i,w,c)

ith(i)
KWIC Observations

- Similar at run time
  - May have identical data representations, algorithms, even compiled code

- Different in code
  - Understanding
  - Documenting
  - Evolving
Effect of Change?

- Change input format
- Don’t store all lines in memory at once
- Use an encoding to save line storage space
- Store the shifts directly instead of indexing
- Amortize alphabetization over searches
Effect of Change?

- Change input format
  - Input module only
- Don’t store all lines in memory at once
  - Design #1: all modules
  - Design #2: Line Storage only
- Use an encoding to save line storage space
  - Design #1: all modules
  - Design #2: Line Storage only
- Store the shifts directly instead of indexing
  - Design #1: Circular Shift, Alphabetizer, Output
  - Design #2: Circular Shift only
- Amortize alphabetization over searches
  - Design #1: Alphabetizer, Output, and maybe Master Control
  - Design #2: Alphabetizer only
Other Factors

- **Independent Development**
  - Data formats (#1) more complex than data access interfaces (#2)
  - Easier to agree on interfaces in #2 because they are more abstract
  - More work is independent, less is shared

- **Comprehensibility**
  - Design of data formats in #1 depends on details of each module
  - More difficult to understand each module in isolation for #1
A Note on Performance

• Parnas says that if we are not careful, decomposition #2 will run slower

• He points out that a compiler can replace the function calls with inlined, efficient operations

• Lesson: don’t prematurely optimize
  • Smart compilers enable smart designs
  • Evolvability usually trumps the overhead of a function call anyway
Decomposition Criteria

• Functional decomposition
  • Break down by major processing steps

• Information hiding decomposition
  • Each module is characterized by a design decision it hides from others
  • Interfaces chosen to reveal as little as possible about this
Information Hiding
Derived from definition by Edward Berard – concept due to Parnas

- Decide what design decisions are likely to change and which are likely to be stable
- Put each design decision likely to change into a module
- Assign each module an interface that hides the decision likely to change, and exposes only stable design decisions
- Ensure that the clients of a module depend only on the stable interface, not the implementation

- Benefit: if you correctly predict what may change, and hide information properly, then each change will only affect one module
  - That’s a big if...do you believe it?
# Hiding design decisions

*Information hiding is **NOT** just about data representation*

<table>
<thead>
<tr>
<th>Decision</th>
<th>Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data representation</td>
<td></td>
</tr>
<tr>
<td>Platform</td>
<td></td>
</tr>
<tr>
<td>I/O format</td>
<td></td>
</tr>
<tr>
<td>User Interface</td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td></td>
</tr>
</tbody>
</table>