**Information Hiding in KWIC**

15-413: Introduction to Software Engineering
Jonathan Aldrich

---

**Key Word In Context**

- "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

---

**KWIC Modularization #1**

- Input
- Circular Shift
- Lines
- Master Control
- Shifts
- Alphabetize
- Output

---

**KWIC Modularization #2**

- Input
- Circular Shift
- Lines
- Master Control
- Shifts
- Alphabetize
- Output

---

**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
- Avoid packing 4 characters to a word
- 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
- Avoid packing 4 characters to a word
  - Design #1: all modules
  - Design #2: Circular Shift 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

A Note on Performance

Avoid packing 4 characters to a word

Independent Development

• 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
• This is 1972!
  - But we still hear silly arguments about how (otherwise better) designs are slower
  - Smart compilers enable smart designs

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

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
• This is 1972!
  - But we still hear silly arguments about how (otherwise better) designs are slower
  - Smart compilers enable smart designs

Design Structure Matrices

15-413: Introduction to Software Engineering
Jonathan Aldrich

Design Structure Matrices

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: DSM for a design of three parameters.

- Goal: to capture dependencies in the structure of a design
- A, B, and C are design parameters
- A choice about some aspect of a design
- X means row depends on column
  - B is hierarchically dependent on A
  - If you change A, you might have to change B as well
  - Suggests you should make a decision about A first
  - B and C are interdependent
  - C and A are independent
Design Structure Matrices

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Figure 2: DSM for a proto-modular design.

- Lines show clustering into proto-modules
- Indicates several design decisions will be managed together
- True modules should be independent
  - i.e., no marks outside of its cluster
  - Not true here because B (in the B-C cluster) depends on A

KWIC Design #1

17 October 2005

Design Structure Matrices

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Figure 3: DSM for a modular design obtained by splitting.

- Interface refines the dependence as a design parameter
  - Instead of B depending on A, now A and B both depend on I
  - Serves to decouple A and B
  - Think of I as the interface of A

KWIC Design #2

17 October 2005

KWIC Design #1

17 October 2005

EDSMs: Considering Possible Changes

- Environment and Design Structure Matrices
  - Sullivan et al., ESEC/FSE 2001
- Add changes as environmental parameters
  - Note: slightly more concrete than what Sullivan et al. propose
  - Only partially controlled by designer
  - May affect each other
  - May affect design decisions in code
- What interfaces are affected?
  - Information hiding; interfaces should be stable
- What implementations are affected?
  - Information hiding hypothesis: should be local to a module

17 October 2005
Summary

- EDSMs are a structured way of thinking about the value of design
- Are design decisions isolated to a module?
- How do modules depend on interfaces?
- How are interfaces and code affected by change?