Restructuring RELAP5-3D

RELAP5 International Users Seminar

Joshua M. Hykes
Penn State University

Aug 16-18, 2006
Structure of Restructuring

- Purpose
- Description of Modular Programming
- Means of restructuring
- Algorithm for applying FOR_STRUCT
- Results
Purpose of Restructuring Effort

• Simplify the coding for increased ease of:
  – Reading & understanding
  – Maintenance
  – Development
• Reduce chance of errors

Essentially, making the developer’s job easier.
Modular Programming: Definition

Object-Oriented Programming (OOP)
• Classes
• Hierarchies

Structured (Procedural) Programming
Blocks of code with only:
• 1 entry
• 1 exit (debated)
• Sequence
• Selection
• Iteration
Structured Blocks

Sequential

- STMT 1
- STMT 2
- STMT 3

Iteration / Loop

- Until
  - Body Block

- While
  - Body Block

If / Case

- IF
  - Block T
  - Block F

- CASE
  - Block 1
  - Block N
  - Block Default

One entry
One exit
Structured Programming and RELAP: An Oxymoron?

• RELAP was largely coded without any concept of structured programming.
  – ~6200 go to statements
  – Computed go to statements
  – Arithmetic if statements
Other complications

- Pre-compiler directives
  - 0-58 unique PCD’s in source files
- Some newer (Fortran 90) coding
- Sheer size
  - Up to 4800 lines of code in source files
Road to Restructuring

- FOR_STRUCT version 2.1.1 (FS2.1.1) – a commercial software package for restructuring F77 and earlier Fortran.
- Unix shell scripts – take care of some of the extra problems
- Manual changes
  - contains subroutines
  - Moved endif’s
  - Lines over the 72 column limit
FOR_STRUCT Restructuring

- do...continue
- go to
- Computed go to
- Arithmetic if

- do...enndo
- cycle & exit
- do while...enndo
- if then...elseif...else...endif
FOR_STRUCT Failings

- Fails to completely restructure convoluted code
- Converts case statements
- Cannot restructure with pre-compiler directives
- Crashes on many of the F90 syntax changes
Abridged Algorithm

Input File

Pre-processing
• contains subroutines
• dummy variables
• #ifdef file splitting

FOR_STRUCT

Recombination

Post-processing
• undoing dummy variables
• formatting

Output File
Scripting Solutions

• Prepare coding so that FS2.1.1 can handle it
  – Example:
    • Change derived-type variables (with %) to dummy variables.
    • After processing, substitute back to original.
• Pass multiple copies through FS2.1.1 to handle pre-compiler directives
  – Turn on different options for the different runs.
• Apply FS2.1.1 iteratively.
Manual Manipulations

• contains subroutines
  – F90 feature – no new specifications required
  – Improves modularity
  – Decreases chances of FS2.1.1 errors
  – Further increases high-level understanding of code
  – Limitation: subroutine must not include `go to` statements to other sections of the code

• Nomadic `endif`
  – FS2.1.1 tends to misplace `endif` or `enddo` statements when followed by a `#endif`
Results: Progress

May 2005

- Unstructured: 493 files (89%)
- Structured: 61 files (11%)

August 2006

- Unstructured: 6 files (1%)
- Structured: 61 files (11%)
- Restructured: 536 files (88%)
Results: Complexity

- McCabe Cyclomatic Complexity - the number of independent logic paths through the code.

Mean values for the converted code:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>McCabe Cyclomatic Index</td>
<td>50.3</td>
<td>↑ 1.6</td>
</tr>
<tr>
<td>Maximum Nested Depth</td>
<td>13.7</td>
<td>↑ 5.0</td>
</tr>
<tr>
<td>Average Nested Depth</td>
<td>6.1</td>
<td>↑ 2.6</td>
</tr>
</tbody>
</table>
Cursory Conclusions

• Pre-compiler directives complicate restructuring.
• Contains subroutines increase modularity and simplify restructuring, with a few limitations.
• Restructuring has little effect on Cyclomatic complexity.
• Unix scripting is extremely powerful, but…
• There’s no substitute for blood, sweat, & tears.