Theresa Hunt

Theresa Hunt, ASQ CSQE is a lead consultant/trainer for the Westfall Team. Her specialties include software quality engineering, process definition and improvement, and testing. Prior to joining the Westfall Team Theresa was principal software quality engineer at ECC International Corp. based in Orlando, FL. The company provides a wide range of products and services used by all branches of the U.S. Department of Defense and by armed forces in 25 countries. Hunt serves as chair of the ASQ Software Division Programs Committee, is on the ASQ CSQE Exam Development Committee, and is currently continuing studies at Embry-Riddle Aeronautical University in Management of Technical Operations.

Basis Path Testing

The Westfall Team

Theresa Hunt
1660 Barton St.
Longwood, Fl 32750
phone: (407) 834-5825
fax: (407) 834-2735
Theresahunt@earthlink.net
www.westfallteam.com

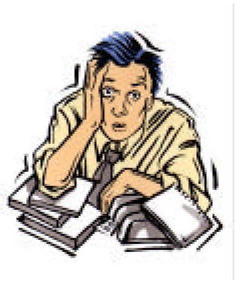
The Challenge of Testing

Slide

The Challenge of Testing

A major challenge in testing is to determine a good starting set of test cases that:

- Eliminate redundant testing
- Provide adequate test coverage
- Allow more effective testing
- Make the most of limited testing resources



Too Many Paths

There are typically many possible paths between the entry and exit of a typical software program. Every decision doubles the number of potential paths, every case statement multiplies the number of potential paths by the number of cases and every loop multiplies the number of potential paths by the number of different iteration values possible for the loop. [based on Beizer-90]

Complete path coverage of even a simple unit is extremely difficult. For example, software that includes straight line code except for a single loop that can be executed from 1 to 100 times would have 100 paths.

The Challenge

Given that testing, like all other development activities has a limited amount of resources (time, people, equipment), the challenge is to select a set of test cases that is most likely to identify as many different potential defects as possible within those limits. To do this, we must eliminate as much redundancy as possible from the testing process while still insuring adequate test coverage.

Basis Path Testing Defined

Slide

What is Basis Path Testing?

Basis path testing is a hybrid between path testing and branch testing:

Path Testing: Testing designed to execute all or selected paths through a computer program [IEEE610]

Branch Testing: Testing designed to execute each outcome of each decision point in a computer program [IEEE610]

Basis Path Testing: Testing that fulfills the requirements of branch testing & also tests all of the independent paths that could be used to construct any arbitrary path through the computer program [based on NIST]

Path	A path through the software is a sequence of instructions or statements that starts at an entry, junction, or decision and ends at another, or possibly the same, junction, decision, or exit. A path may go through several junctions, processes, or decisions, one or more times. [Beizer-90]
Independent Path	An independent path is any path through the software that introduces at least one new set of processing statements or a new condition. When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined. [Pressman-01]

Basis Path Testing Defined (cont.)

Slide

What is a Basis Path?

A *basis path* is a unique path through the software where no iterations are allowed - all possible paths through the system are linear combinations of them.

Slide

McCabe's Basis Path Testing

Steps:

1: Draw a control flow graph

2: Calculate Cyclomatic complexity

3: Choose a "basis set" of paths

4: Generate test cases to exercise each path

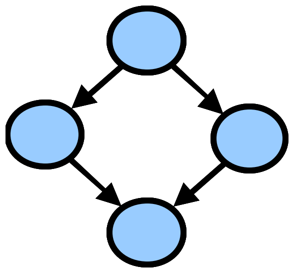
Step 1: Draw a Control Flow Graph

Slide

The Control Flow Graph

Any procedural design can be translated into a control flow graph:

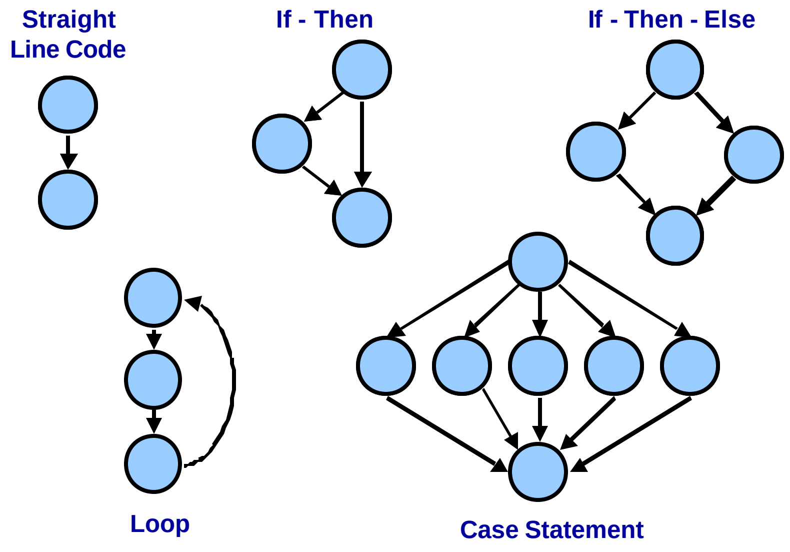
- Lines (or arrows) called *edges* represent flow of control
- Circles called *nodes* represent one or more actions
- Areas bounded by edges and nodes called *regions*
- A *predicate node* is a node containing a condition



Slide

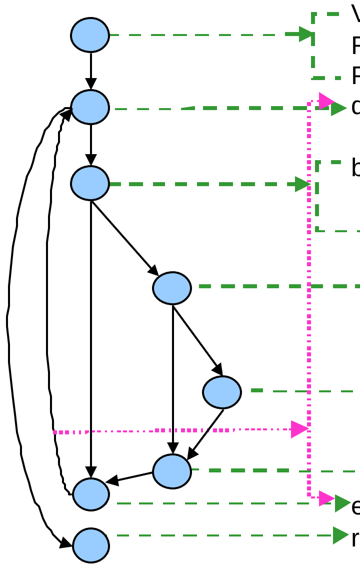
Control Flow Graph Structures

Basic control flow graph structures:



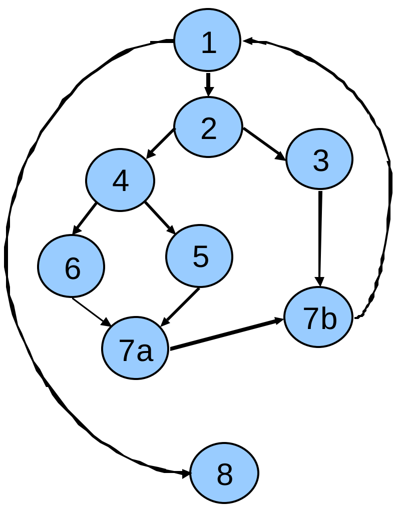
Step 1: Draw a Control Flow Graph (cont.)

Slide **Draw a Control Flow Graph - Example**



```
Procedure Validate_Pin (Valid_Pin, Return_Code)
Valid_Pin = FALSE
Return_Code = GOOD
Pin_Count = 0
do until Valid_Pin = TRUE or Pin_Count > 2 or
                  Return_Code = CANCEL
begin
    get Pin_Number (Pin_Number, Return_Code)
    if (Return_Code ≠ CANCEL)
       begin
       call Validate Pin_Number (Pin_Number, Valid_Pin)
       if (Valid_Pin = FALSE) then
            begin
            output "Invalid PIN, please re-enter PIN"
            Pin_Count = Pin_Count + 1
            end
       end
end
return (Valid_Pin, Return_Code)
```

Slide **Another Example**



```
1.    do while records remain
         read record;
 2.      if record field 1 = 0
 3.         then process record;
                store in buffer;
                increment counter;
4.       elsif record field 2 = 0
5.           then reset record;
6.           else process record;
             store in file;
7a.      endif;
       endif;
7b.    enddo;
8.   end;
```

[based on Sobey]

Step 1 The first step in basis path testing is to draw the control flow graph. As illustrated in the examples above, this can be done directly from the source code.

Step 2: Calculate Cyclomatic Complexity

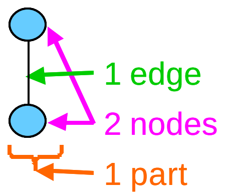
Slide

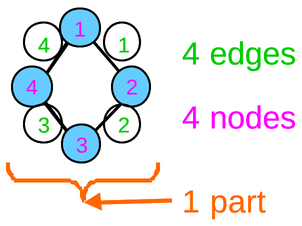
Step 2: Calculate Cyclomatic Complexity

Control flow graph:

Model: V(G) = edges - nodes + 2p

where p = number of unconnected parts of the graph





V(G) = 1 - 2 + 2x1 = 1
Straight line code always has a complexity of 1

V(G) = 4 - 4 + 2x1 = 2

Cyclomatic Complexity The second step in basis path testing is to calculate the Cyclomatic complexity from the control flow graph. McCabe's Cyclomatic Complexity is a measure of the number of linearly independent paths through the unit/component. It can therefore be used in structural testing to determine the minimum number of tests that must be executed for complete basis path coverage.

Calculation Cyclomatic Complexity is calculated from a control flow graph by subtracting the number of nodes from the number of edges and adding 2 times the number of unconnected parts of the graph. For example, straight-line code (first control flow graph above) has 1 edge and 2 nodes so its complexity is 1-2+2*1 = 1. This second graph has 4 edges and 4 nodes so its complexity is 4-4+2*1 = 2. What is the complexity of the other two graphs?

Tools Static analysis tools also exist for drawing control flow graphs and/or calculating Cyclomatic complexity from source code.

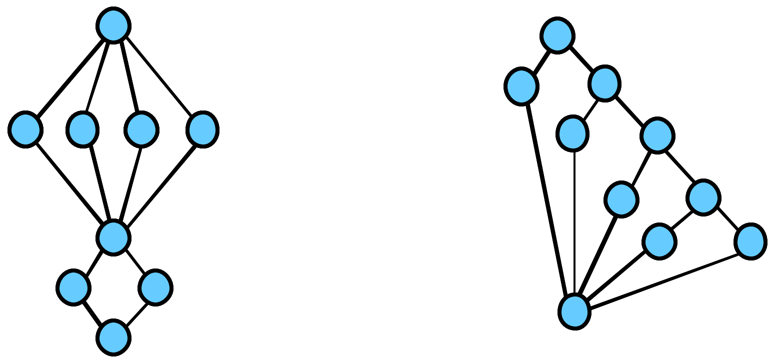
Step 2: Calculate Cyclomatic Complexity (cont.)

Slide **Cyclomatic Complexity - Exercise -**

Control flow graph:

Model: V(G) = edges - nodes + 2p

where p = number of unconnected parts of the graph



Instructions : Calculate the Cyclomatic complexity separately for each of the two control flow graphs above.

Cyclomatic complexity of first graph = ____________________

Cyclomatic complexity of second graph = __________________

Then assuming that these two control flow graphs are two unconnected parts of the same graph, calculate the Cyclomatic complexity of the combined graph.

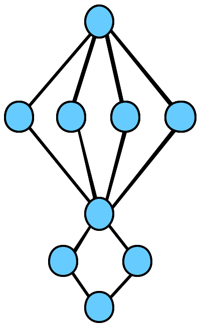
Cyclomatic complexity of combined graph = ________________

Step 2: Calculate Cyclomatic Complexity (cont.)

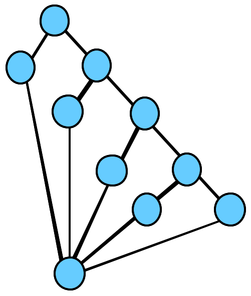
Slide

Answer to Exercise

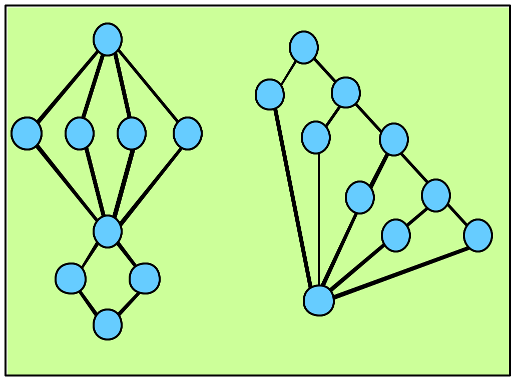
V(G) = edges - nodes + 2p



12 edges
9 nodes
12-9+2(1)=5



13 edges
10 nodes
13-10+2(1)=5



25 edges
19 nodes
25-19+2(2)=10

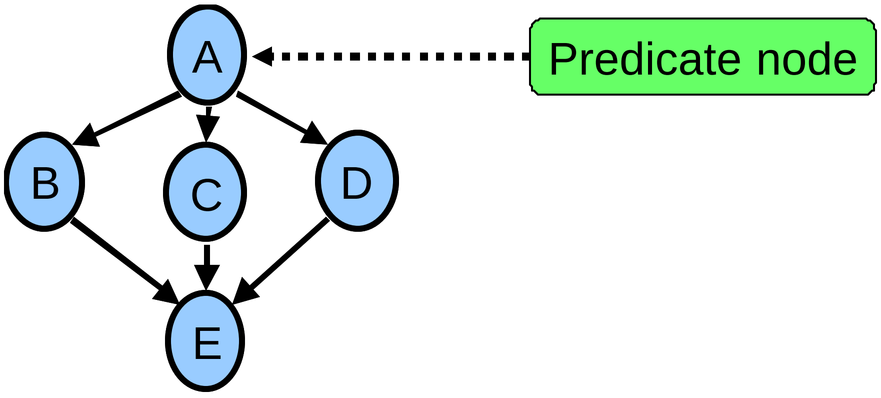
Step 2: Calculate Cyclomatic Complexity (cont.)

Slides

Predicate Nodes Formula

One possible way of calculating V(G) is to use the predicate nodes formula:

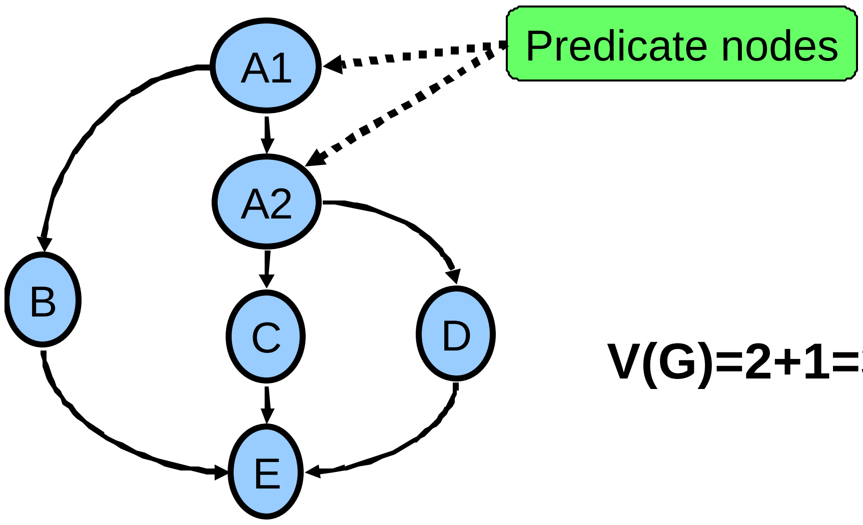
V(G) = Number of Predicate Nodes + 1



Thus for this particular graph: V(G)=1+1=2

***BUT*……..** V(G)=6-5+2(1)=3……………

A limitation on the predicate nodes formula is that it assumes that there are only two outgoing flows for each of such nodes. To adjust for this, split the predicate node into two sub-nodes:



V(G)=2+1=3

[based on Chek]

Step 2: Calculate Cyclomatic Complexity (cont.)

Slide

Uses of Cyclomatic Complexity

Cyclomatic complexity can be used to:

- Tell us how many paths to look for in basis path testing
- Help pinpoint areas of potential instability
- Indicate a unit/component's testability & understandability (maintainability)
- Provide a quantitative indication of unit/component's control flow complexity
- Indicate the effort required to test a unit/component

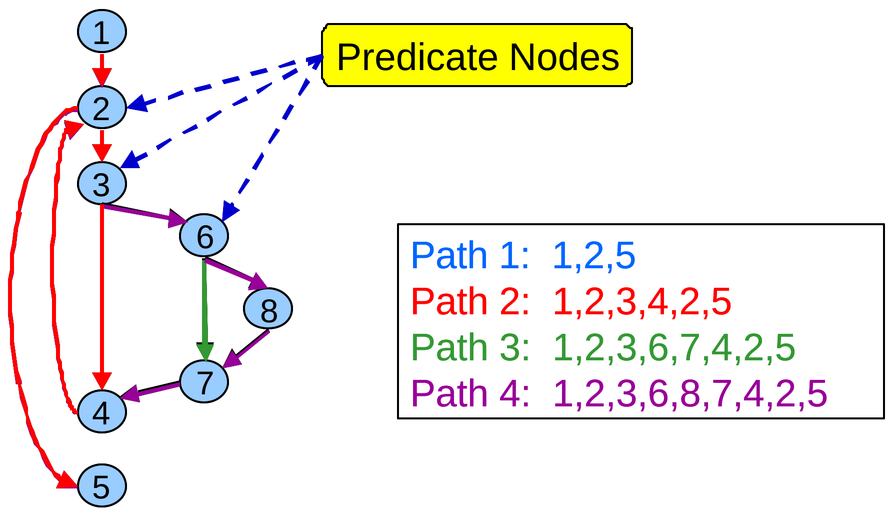
Use

Based on the unit/component's Cyclomatic complexity we can apply appropriate levels of effort on detailed inspections and testing. Those units/components having a high complexity measure need more intense inspection and test, those with lower measures are likely to have fewer defects and thus would require less stringent inspection and test. The Cyclomatic complexity also tells us how many paths to evaluate for basis path testing

Step 3: Choose a Basis Set of Paths

Slide **Step 3: Choose a Basis Set of Paths**

Using a control flow graph:



Step 1 Draw the control flow graph – for this example we will use the control flow graph we drew in the first code example.

Step 2 Calculate the Cyclomatic complexity – Remember there are actually three ways to calculate the Cyclomatic complexity of a control flow graph.

1. V(G) = edges – nodes + 2p. For this example there are 10 edges, 8 nodes and p is 1, so V(G) = 10 – 8 + 2 = 4
2. V(G) = number of regions in the control flow graph. For this example there are 3 enclosed regions plus the outside region, so V(G) = 4.
3. V(G) = number of predicate nodes + 1. A predicate node is a node with more than one edge emanating from it. For this example, nodes 2, 3 and 6 are predicate nodes, so V(G) = 3 + 1 = 4.

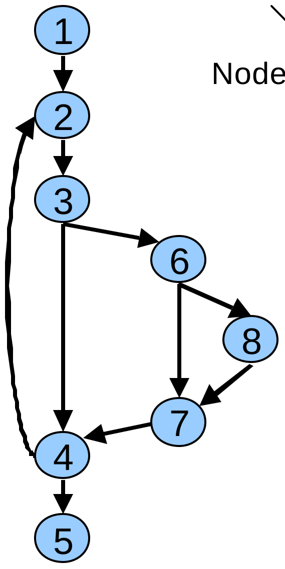
Step 3 Choose a set of basis paths – Determining the predicate nodes can help identify a set of basis paths. If test cases can be designed to cover the basis path set, it will result in complete decision (and statement) coverage of the code. Each basis path that you select must in effect test at least one new untested edge, in other words it must traverse at least one new edge. Otherwise it is considered a redundant path and does not belong in the basis set. This aids in eliminating redundant testing and ensures validity of each test case. For each subsequent basis path selected try to keep the number of new edges added as low as possible – progressing on until you have covered all possible basis paths.

Step 3: Choose a Basis Set of Paths (cont.)

Slide

Step 3: Choose a Basis Set of Paths (cont.)

The Graph (Connection) Matrix



Node \ Connected to node	1	2	3	4	5	6	7	8	
1	0	1	0	0	0	0	0	0	1 – 1 = 0
2	0	0	1	0	0	0	0	0	1 – 1 = 0
3	0	0	0	1	0	1	0	0	2 - 1 = 1
4	0	1	0	0	1	0	0	0	2 – 1 = 1
5	0	0	0	0	0	0	0	0	---
6	0	0	0	0	0	0	1	1	2 – 1 = 1
7	0	0	0	1	0	0	0	0	1 – 1 = 0
8	0	0	0	0	0	0	1	0	1 – 1 = 0

3+1=4

Graph matrix A graph matrix, also called a connection matrix:

- Is square with #sides equal to #nodes
- Has rows and columns that correspond to the nodes
- Has non-zero value weights that correspond to the edges

Note For additional information refer to the National Bureau of Standards special publication "Structured Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric. [McCabe-82]

Link Weight Can associate a number with each edge entry. In its simplest form, the link weight is 1 (a connection exists) or 0 (a connection does not exist).

Use a value of 1 (indicating that a connection exists) to calculate the Cyclomatic complexity:

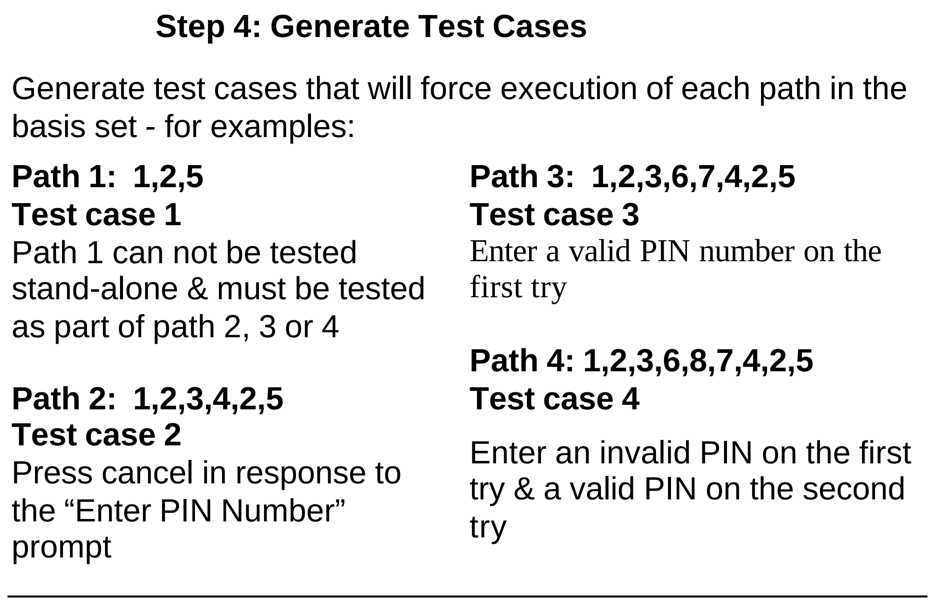
- For each row, sum column values and subtract 1
- Sum these totals and add 1

Some other interesting link weights:

- Probability that a link (edge) will be executed
- Processing time for traversal of a link
- Memory required during traversal of a link
- Resources required during traversal of a link

Step 4: Generate Test Cases

Slide



Step 4: Generate Test Cases

Generate test cases that will force execution of each path in the basis set - for examples:

Path 1: 1,2,5
Test case 1
Path 1 can not be tested stand-alone & must be tested as part of path 2, 3 or 4

Path 2: 1,2,3,4,2,5
Test case 2
Press cancel in response to the “Enter PIN Number” prompt

Path 3: 1,2,3,6,7,4,2,5
Test case 3
Enter a valid PIN number on the first try

Path 4: 1,2,3,6,8,7,4,2,5
Test case 4

Enter an invalid PIN on the first try & a valid PIN on the second try

Test Case 3 It should be noted that in order to follow path 4 this test case executes the loop a second time so the path is actually 1,2,3,6,8,7,4,2,3,6,7,4,2,5.

Basis Path Testing During Integration

Slide

Basis Path Testing During Integration

Basis path testing can also be applied to integration testing when units/components are integrated together

McCabe’s Design Predicate approach:

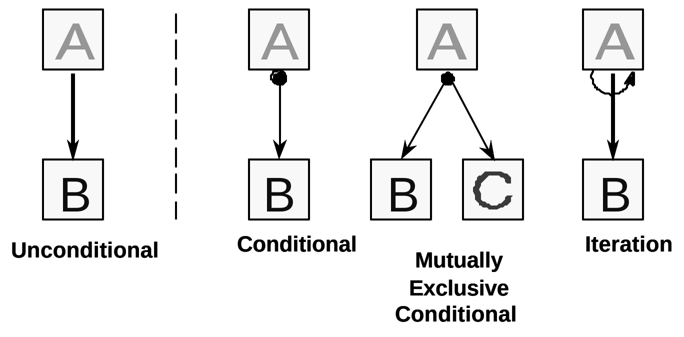
- Draw a “structure chart”
- Calculate integration complexity
- Select tests to exercise every “type” of interaction, not every combination

This approach can be used to predict the integration test effort before coding begins

Slide

Types of Interactions

Types of unit/component interactions:



Unconditional Unit/component A always calls unit/component B. A calling tree always has an integration complexity of at least one. The integration complexity is NOT incremented for each occurrence of an unconditional interaction.

Conditional Unit/component A calls unit/component B only if certain conditions are met. The integration complexity is incremented by one for each occurrence of a conditional interaction in the structure chart.

Mutually Exclusive Unit/component A calls either unit/component B or unit/component C (but not both) based upon certain conditions being met. The integration complexity is incremented by one for each occurrence of a mutually exclusive conditional interaction in the structure chart.

Iterative Unit/component A calls unit/component B one or more times based upon certain conditions being met. The integration complexity is incremented by one for each occurrence of an iterative interaction in the structure chart.

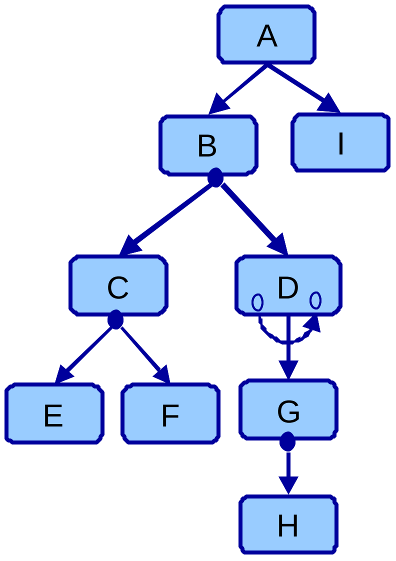
Basis Path Testing During Integration (cont.)

Slide **Integration Basis Path Test Cases**

Basis path testing can also be applied to integration testing when units/components are integrated together:

Complexity: 5

Basis set of paths:
1. A,B,C,E & then A calls I
2. A,B,C,F & then A calls I
3. A,B,D,G (only once) & then A calls I
4. A,B,D,G (more than once) & then A calls I
5. A,B,D,G,H & then A calls I



Basis Path Testing Basis path testing is a white-box testing technique that identifies test cases based on the flows or logical paths that can be taken through the software. However, basis path testing can also be applied to integration testing when software units/components are integrated together. The use of the technique quantifies the integration effort involved as well as the design-level complexity. A basis path is a unique path through the software where no iterations are allowed.

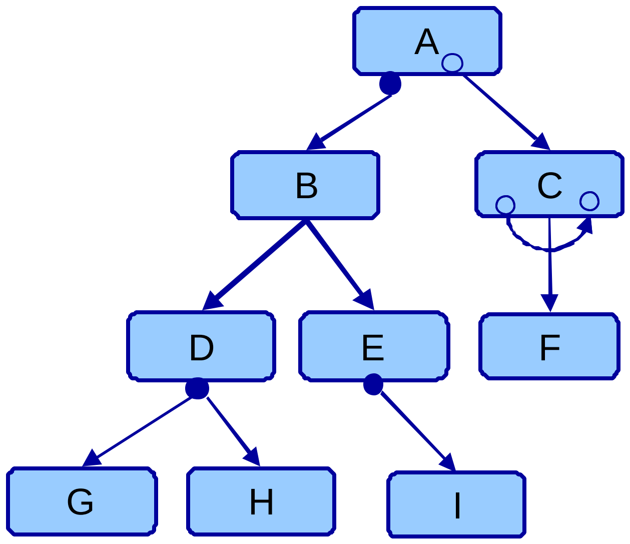
Basis Path Testing During Integration – Exercise

Slide **Integration Basis Path Testing - Exercise**

Instructions:

1.Calculate the integration complexity for the calling tree

2.Select a set of basis paths to test the calling tree



Integration Complexity = __________

Basis Paths:

Basis Path Testing During Integration - Answer

Slide

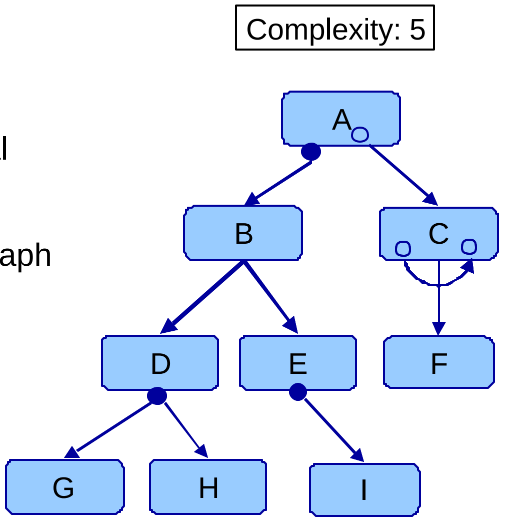
Integration Basis Path Testing - Answer

Interaction types:

- 2 conditional
- 1 mutually exclusive conditional
- 1 iteration
- 1 simply because we have a graph

Basis set of paths:

1. A,C,F (only once)
2. A,C,F (more than once)
3. A,B,D,G & then B calls E & then A calls C
4. A,B,D,H & then B calls E & then A calls C,F
5. A,B,D,H & then B calls E, I & then A calls C,F



Answer - Integration Basis Path Testing (cont.)

Slide

And a "What If"

Interaction types:

3 conditional

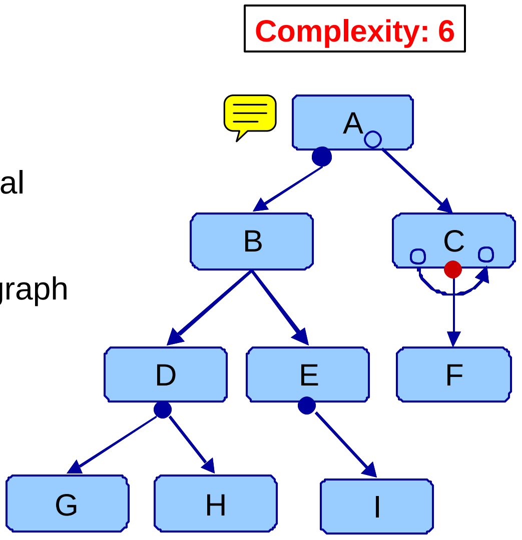
1 mutually exclusive conditional

1 iteration

1 simply because we have a graph

Basis set of paths:

1. A,C,F (only once)
2. A,C,F (more than once)
3. A,B,D,G & then B calls E & then A calls C
4. A,B,D,H & then B calls E & then A calls C,F
5. A,B,D,H & then B calls E, I & then A calls C,F
6. **A,C**



Addition If the call from unit C to unit F is both conditional and iterative, you add one to the integration complexity for the iteration and one for the conditional. This would increase the integration complexity of this diagram to 6. The additional basis path would be A,C.

Conclusion

Slide **Conclusion**

Benefits of basis path testing:

- Defines the number of independent paths thus the number of test cases needed to ensure:
  - ♦ Every statement will be executed at least one time
  - ♦ Every condition will be executed on its true & false sides
- Focuses attention on program logic
- Facilitates analytical versus arbitrary test case design

References

Beizer-90	Boris Beizer, *Software Testing Techniques*, Van Nostrand Reinhold., New York, 1990, ISBN 0-442-20672-0.
GSAM	Department of the Air Force Software Technology Support Center, *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems Command and Control Systems Management Information Systems, Version 3.0*, May 2000, Hill Air Force Base, Utah 84056-5205, https://www.stsc.hill.af.mil
IEEE-610	IEEE Standards Software Engineering, Volume 1, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std. 610-1990 , The Institute of Electrical and Electronics Engineers, 1999, ISBN 0-7381-1559-2.
McCabe-82	Thomas J. McCabe, Structured Testing: *A Software Testing Methodology Using the Cyclomatic Complexity Metric*, NBS Special Publication, National Bureau of Standards, 1982.
Pressman-01	Roger Pressman, *Software Engineering, A practitioner's Approach, Fifth Edition*, McGraw Hill, Boston, 2001, ISBN 0-07-365578-3.

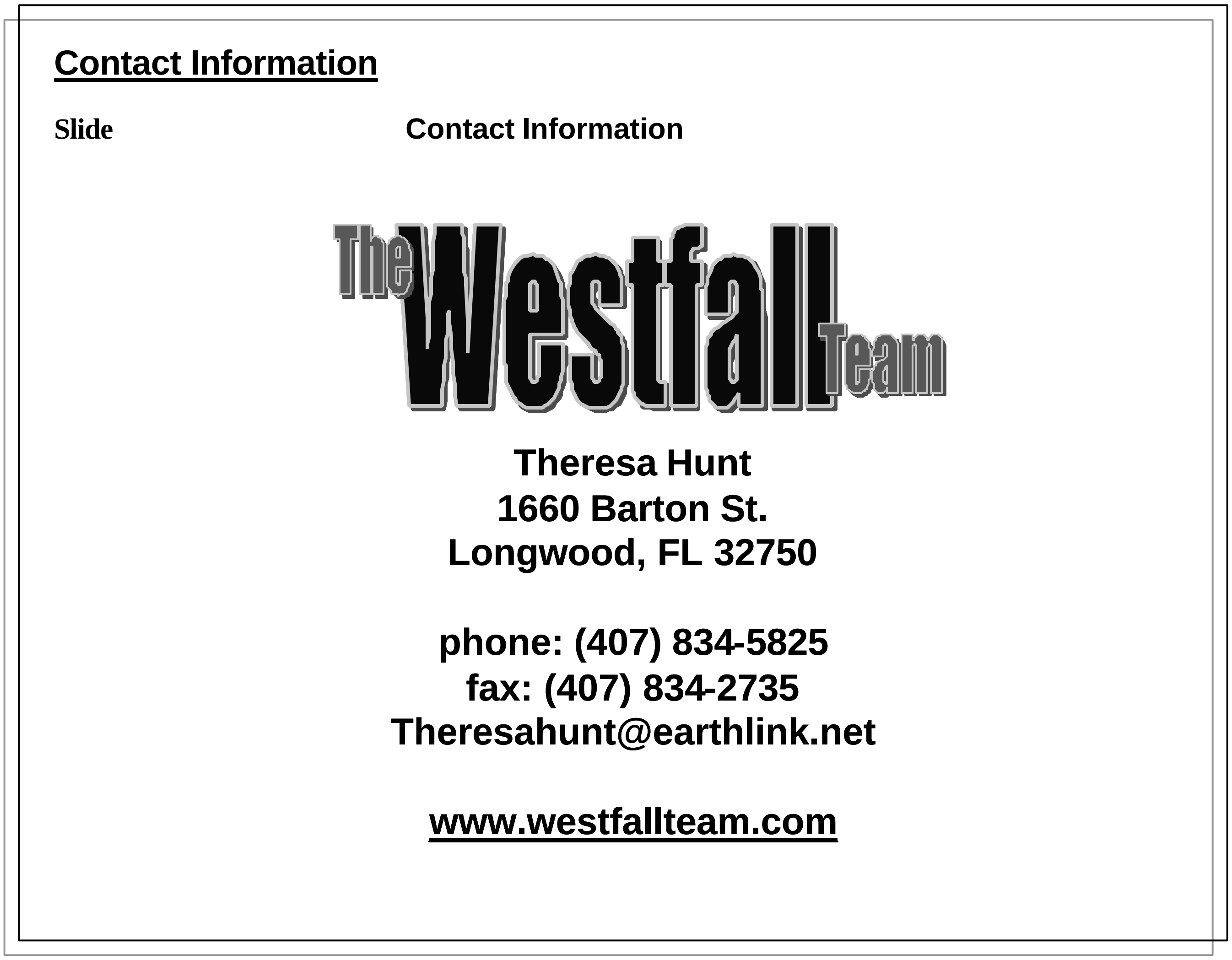
On-Line Resources (note: The use of all on-line resources is subject to applicable copyright laws and may change with time and publication status. The net locations given below should also be considered dynamic, but are accurate at the completion of this paper.)

Unpublished notes on basis path testing are available at:

Check Yang	http://www.chekyang.com
Dr. A.J. Sobey	http://louisa.levels.unisa.edu.au/se1/testing-notes/test01_3.htm
Joseph Poole	NISTIR 5737 - http://hissa.nist.gov/publications/nistir5737/

Contact Information

Slide **Contact Information**



Theresa Hunt
1660 Barton St.
Longwood, FL 32750

phone: (407) 834-5825
fax: (407) 834-2735
Theresahunt@earthlink.net

www.westfallteam.com