TB

Full-day Tutorials
5/6/2014 8:30:00 AM

Key Test Design Techniques

Presented by:

Lee Copeland
Software Quality Engineering

Brought to you by:

340 Corporate Way, Suite 300, Orange Park, FL 32073
888-268-8770 · 904-278-0524 · sqeinfo@sqe.com · www.sqe.com

Lee Copeland

Software Quality Engineering

With more than thirty years of experience as an information systems professional at commercial and nonprofit organizations, **Lee Copeland** has held technical and managerial positions in applications development, software testing, and software process improvement. Lee has developed and taught numerous training courses on software development and testing issues, and is a well-known speaker with Software Quality Engineering. Lee presents at software conferences in the United States and abroad. He is the author of the popular reference book, *A Practitioner's Guide to Software Test Design*.

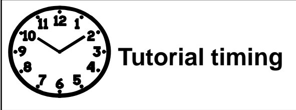
Key Test Design Techniques

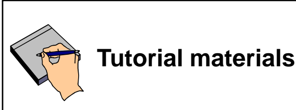
Software Quality Engineering
340 Corporate Way, Suite 300
Orange Park, Florida 32073
904.278.0707

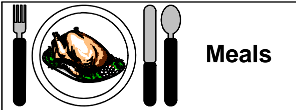
Lee Copeland
lee@sqe.com

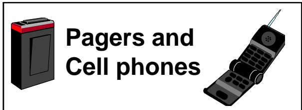
SQE ©2001-2011 Version 4.0

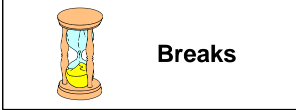
Administrivia











Tutorial Goals

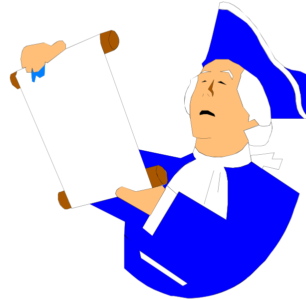
- **At the end of the tutorial you will be able to**
  - Design test cases using structured, formal, “scientific” techniques
  - Implement the “art” of test case design as well as the science
  - Understand how defect taxonomies can improve test case design

Chapter 1

Introduction

Tutorial Agenda

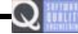
1. Introduction
2. Black Box Science
3. White Box Science
4. Black Box Art
5. Defect Taxonomies
6. Wrap-up

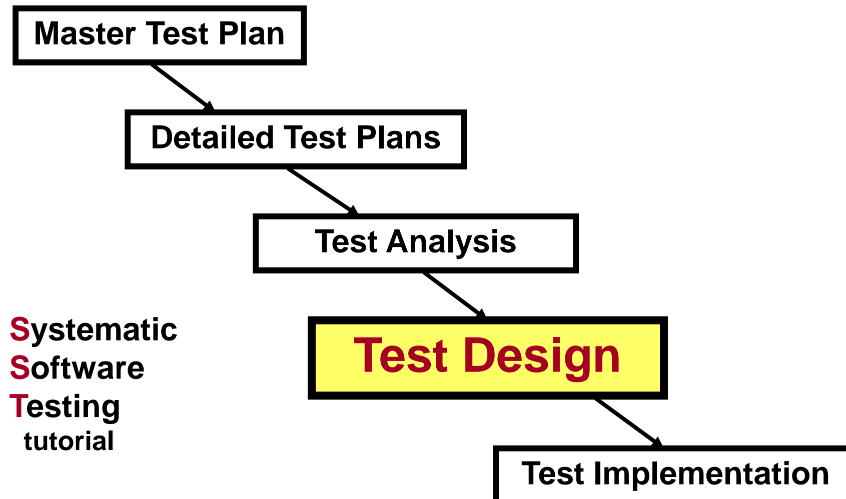


Chapter Objectives

- **At the end of this chapter you will be able to**
  - Define "testing"
  - Define the components of a test case
  - Discuss the problems of how much testing can be done

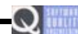
Test Design Process - Context





Systematic
Software
Testing
tutorial

What is Testing?



- **IEEE Std 610 Software Engineering Terminology: "test"**
  1. An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component
  2. To conduct an activity as in (1)
  3. A set of one or more test cases
  4. A set of one or more test procedures
  5. A set of one or more test cases and procedures

What Do We Test?

Fundamental issues	FUNCTIONS	Portability
	Usability	Reliability
	Security	Availability
	Localization	Testability
	Extensibility	Interoperability
Real-time issues	Timing	Capacity
	Throughput	Performance
System management issues	Overload	
	Backup and Recovery	
	Start-up and shut-down	
Purchased software	Operating systems	
	DBMS	
	Communications	

Current Challenges In Test Design

?

?

?

?

?

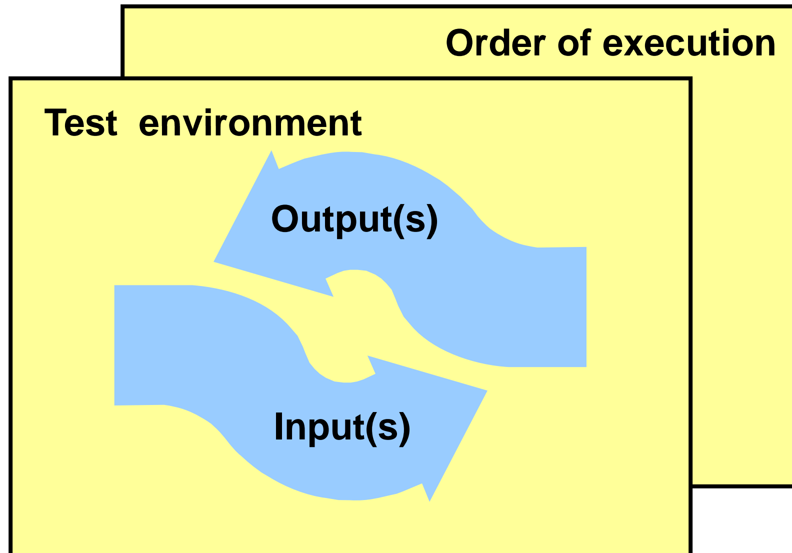
?

?

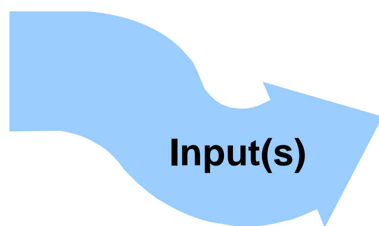
?

?

What Are Test Cases?

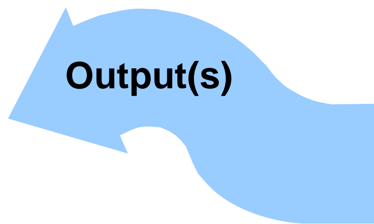


What Are Test Cases?



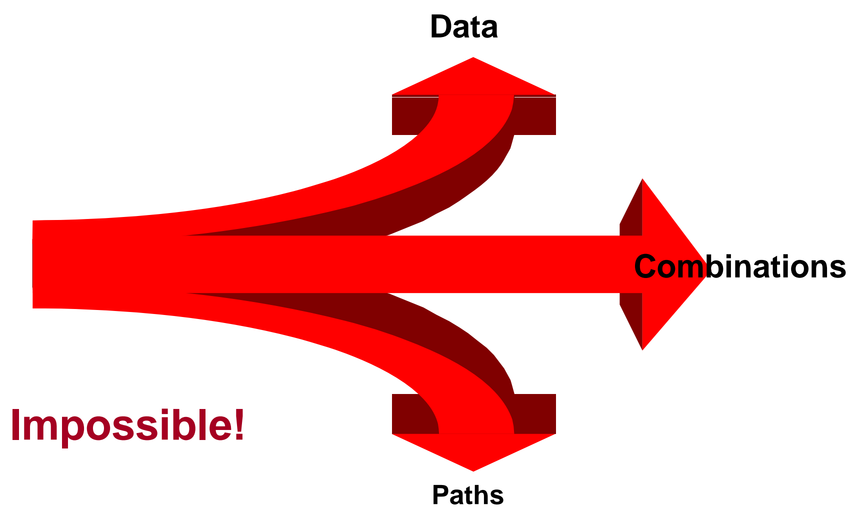
- **Inputs can be**
  - Data entered through the UI
  - Data from interfacing systems
  - Data from interfacing devices
  - Files
  - Databases
  - State
  - Environment

What Are Test Cases?

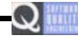


- **Outputs can be:**
  - Data to UI
  - Data to interfacing systems
  - Data to interfacing devices
  - Files
  - Databases
  - State
  - Response time

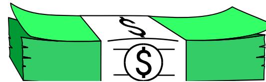
Test Everything?



Test Wisely!

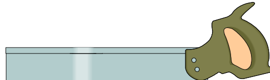


Choose wisely

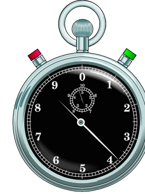


Cost effective

Risk based



Tool support



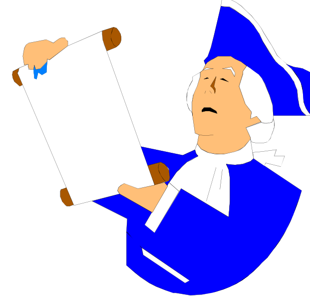
Time limited

Chapter 2

Black Box Science

Tutorial Agenda

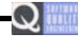
1. Introduction
2. Black Box Science
3. White Box Science
4. Black Box Art
5. Defect Taxonomies
6. Wrap-up



Objectives

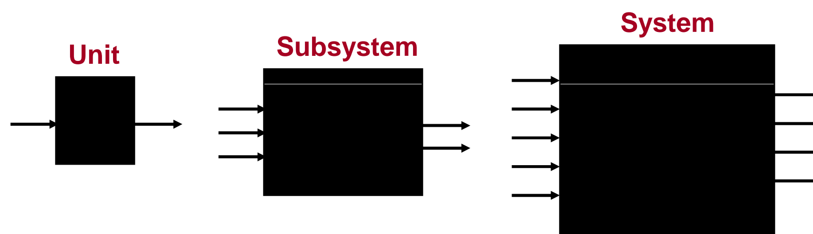
- **At the end of this chapter you will be able to**
  - Implement the Equivalence Class Partitioning and Boundary Value testing techniques
  - Create a Decision Table
  - Describe when State-Transition diagrams are useful and how to create one
  - Understand and use "All Pairs" based techniques
    - Orthogonal arrays and pairs based test cases
  - Create a Traceability Matrix
    - If one is desired or required

What is Black Box Science?



- **Black Box**
  - Focus only on the inputs and outputs
  - Ignore the internal paths, structure, and implementation
  - Can't know the percentage of code tested
- **Science**
  - Use structured approach(es)
  - Approach all data the same
  - Have a methodology based coverage measure

Black Box at Different Levels



A black box is a black box is a black box.

It's just a bigger box with more input, functionality, and output.

For Each Software Requirement

- **First let's focus on one requirement / function / story at a time**
  - Equivalence Class Partitioning
  - Boundary Value Testing
  - Combining both across multiple inputs

Equivalence Class Partitioning

- **The role of Equivalence Class Partitioning**
  - Create the minimum number of black box tests needed while still providing adequate coverage
- **Steps**
  1. Identify equivalence classes - the input ranges which are treated the same by the software
     - Valid classes: legal input ranges
     - Invalid classes: illegal or out of range input values
  2. Create a test case for each of the equivalence classes

Equivalence Class Partitioning

1. **If an input condition specifies a continuous range of values, there is one valid class and two invalid classes.**
   - **Example:** The input variable is a mortgage applicant's income. The valid range is $1000.00/mo. to $75,000.00/mo.
     - Valid class: **{1000.00 < = income < = 75,000.00}**
     - Invalid classes: **{income < 1000.00}, {income > 75,000.00}**

2. **If an input condition specifies a discrete range of permissible values, there is one valid class and two invalid classes.**
   - **Example:** The input variable is the total number of houses being purchased, from 1 to 5.
     - Valid class: **{1 < = #_Houses < = 5}**
     - Invalid classes: **{#_Houses < 1}, {#_Houses > 5}**

Equivalence Class Partitioning

3. **If an input condition specifies a set of values, there is one valid and one invalid equivalence class.**
   - Example: Types of housing are Condo, Townhouse, and Single Family
     - Valid class: **{Condo, Townhouse, Single Family}**
     - Invalid class: **{...anything else...}**

- Sets are different
  - **If each member of a set yields the same result the set is a single class**
  - **If each member generates a different result then each member is a separate class with one valid and one invalid class**

Equivalence Class Partitioning

4. **If a "must be" condition is required, there is one valid equivalence class and one invalid class.**
   – Example: The mortgage applicant must be a person.
     - Valid class: **{person}**
     - Invalid class: **{corporation, ...anything else...}**

Another Way to Annotate Partitions

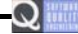
← 999.99	1000.00 to 75,000.00	75,000.01 →
Invalid	Valid	Invalid

← 0	1 to 5	6 →
Invalid	Valid	Invalid

All Else	Condo, Townhouse, Single Family
Invalid	Valid

All Else	Person
Invalid	Valid

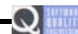
Equivalence Class Partitioning



- Steps for creating the test cases

1. **Define the equivalence classes for your rules**
2. **Write the first test case to cover as many of the valid equivalence classes from the rule set as possible (although they may be mutually exclusive)**
3. **Continue writing test cases until all of the valid equivalence classes from the rules have been covered**
4. **Write one test case for each invalid class**

Example Test Cases (EP)



Test Case	Valid Test Cases			
	Income	No.	Type	Person
1	$4,000.00	2	Condo	Bob
	Invalid Test Cases			
2	$90,000.00	1	Condo	Bob's mom
3	$800.00	2	Single Family	Leroy
4	$4,000.00	7	Townhouse	Larry
5	$2,000.00	0	Condo	Helen
6	$5,500.00	2	Office Bldg.	Sam
7	$60,000.00	2	Townhouse	ABC Incorporated

Boundary Value Testing (BV)

- **The role of Boundary Value Testing**
  - An additional black box testing approach
    - Almost always used with Equivalence Partitioning
  - Test cases at the boundary of each input
    - Includes the following
      - At the boundary (where possible or if significant)
      - Just below the boundary
      - Just above the boundary

Example Test Cases (EP/BV)

Test Case	Valid Test Cases			
	Income	No.	Type	Person
1	$1,000.00	1	Condo	Bob
2	$75,000.00	5	Single Family	Helen
3	$37,500.00	3	Townhouse	George
	Invalid Test Cases			
4	$75,000.01	1	Condo	Bob's mom
5	$999.99	2	Single Family	Leroy
6	$4,000.00	6	Townhouse	Larry
7	$2,000.00	0	Condo	Helen
8	$5,500.00	2	Office Bldg.	Sam
9	$60,000.00	2	Townhouse	ABC Incorporated

More Classes of Invalids

- **Looking at more than one data field at a time**
  - Try invalid combinations
    - All inputs are in valid range
    - But they are NOT valid together
    - Example: Singapore Airlines, Calcutta to Sydney, Wednesday (they have no flights on this route on Wednesday)
  - Try invalid orders
    - All inputs in the valid range
    - But NOT valid due to timing or order
    - Examples: using a system without logging on; asking for a refund before paying

For All Software Requirements

- **Focus on testing all of the requirements to make sure they interoperate successfully**
  - Cross-functional testing
  - Decision Tables
  - State-Transition diagrams
  - Pair based methods
    - Orthogonal Arrays
    - Combinatorial Analysis (all pairs)

Cross Functional Testing

- **Definition**
  - Test more than one function at a time
  - Possibly run these tests first
  - Why?
    - Discover design and interface errors

Cross Functional Testing

EXAMPLE	TC1	TC2	TC3	TC4	TC5
Database 2					
Mail server 2			*	*	
Remote Application 3			*		
Remote Application 2		*	*	*	
Remote Application 1				*	*
Database 1	*		*		
Mail server 1	*			*	*
User interface	*		*	*	*

Decision Table

- **Definition**
  - A table listing all possible "conditions" (inputs) and all possible "actions" (outputs)
  - There is a "rule" for each possible combination of "conditions"

- **Decision tables can be used to represent very complex business rules based on a set of defined conditions (requirements)**

Decision Table Format

	Rule 1	Rule 2	…..	Rule N
Conditions				
Condition-1				
Condition-2				
…….				
Condition-x				
Actions				
Action-1				
Action-2				
……				
Action-z				

Example Decision Table

- **From the Internal Revenue Service**

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Condition								
Single	Yes	Yes	Yes	Yes	No	No	No	No
65 or older OR blind	Yes	Yes	No	No	Yes	Yes	No	No
Unearned income > $1560	Yes	No	Yes	No	Yes	No	Yes	No
Action								
File Return	No	No	Yes	No	Yes	No	Yes	Yes

Another Decision Table

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Conditions								
Claims	0	0	1	1	2-4	2-4	5+	5+
Age	16-25	26-85	16-25	26-85	16-25	26-85	16-25	26-85
Actions								
Premium Increase	$50	$25	$100	$50	$400	$200	$0	$0
Send Warning	No	No	Yes	No	Yes	Yes	No	No
Cancel Policy	No	No	No	No	No	No	Yes	Yes

- **From an automobile insurance company**

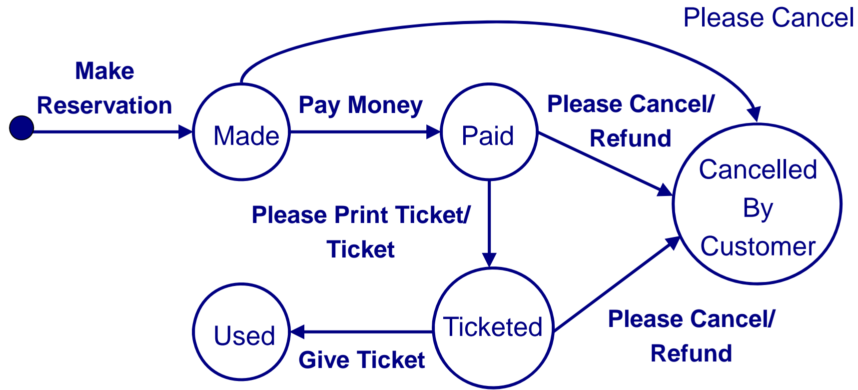
Turning Decision Tables into Test Cases

- **Create the decision table by defining condition and action combinations**
- **Strike out any rules that are "impossible"**
  - **Are they really?**
  - **How can you guarantee it?**
- **Combine columns where the values are immaterial**
  - **We don't care, or do we?**
- **Select test data to make each rule "fire"**

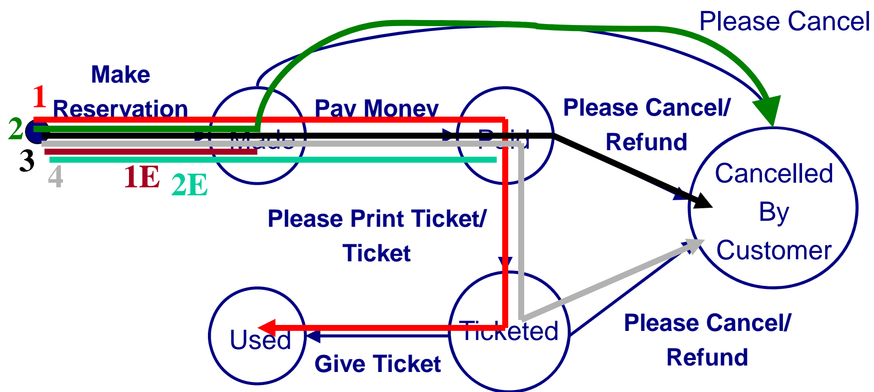
State-Transition Diagrams

- **Used for "state" machines (what happens next is dependent on what has happened before)**
  - Visual illustration of allowed sequences of functions
  - GREAT for finding correct order of valid operations
  - EXCELLENT for finding correct order of test cases
  - May be difficult to get this level of documentation in an updated and current form

A Simple State Example



Simple State Example with Tests



All Pairs Methods

- **Orthogonal Arrays and Combinatorial Analysis**
- **A device for selecting a “good” subset of all possible combinations**
  - When there are too many combinations to consider
  - When it’s risky to randomly skip testing large parts of the functionality or combinations
    - These methods test “All Pairs” (each option with every other option ONCE, but not all combinations across all options).
    - They exercise multiple pairs simultaneously
    - Requires knowledge of all legitimate combinations

Example Orthogonal Arrays

$L_4(2^3)$

	1	2	3
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

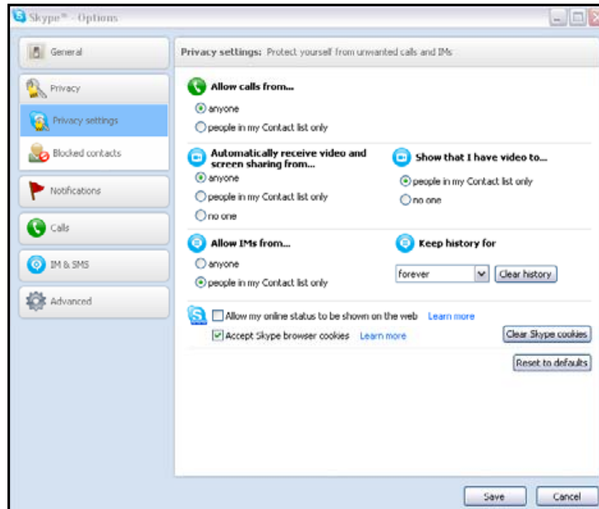
$L_9(3^4)$

	1	2	3	4
1	0	0	0	0
2	0	1	1	1
3	0	2	2	2
4	1	0	1	2
5	1	1	2	0
6	1	2	0	1
7	2	0	2	1
8	2	1	0	2
9	2	2	1	0

$L_{18}(3^6 6^1)$

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	1	2	2	0	1	1
3	0	2	1	2	1	0	2
4	0	1	1	0	2	2	3
5	0	2	0	1	2	1	4
6	0	0	2	1	1	2	5
7	1	1	1	1	1	1	0
8	1	2	0	0	1	2	1
9	1	0	2	0	2	1	2
10	1	2	2	1	0	0	3
11	1	0	1	2	0	2	4
12	1	1	0	2	2	0	5
13	2	2	2	2	2	2	0
14	2	0	1	1	2	0	1
15	2	1	0	1	0	2	2
16	2	0	0	2	1	1	3
17	2	1	2	0	1	0	4
18	2	2	1	0	0	1	5

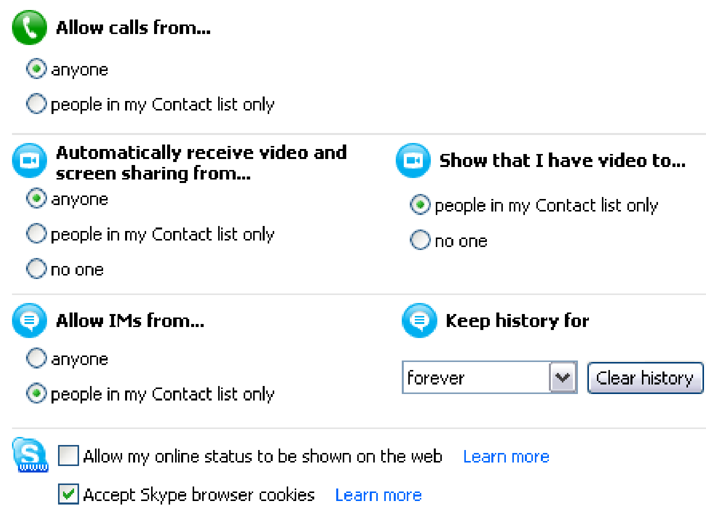
A Problem To Solve



Skype

Tools | Options | Privacy

A Problem To Solve



A Problem To Solve

- Allow calls from – anyone, contact list (2)
- Receive video from – anyone, contact list, no one (3)
- Allow IMs from – anyone, contact list (2)
- Show video to – contact list, no one (2)
- Keep history for – no history, 2 weeks, 1 month, 3 months, forever (5)
- Show online status – yes, no (2)
- Accept cookies – yes, no (2)

2 x 3 x 2 x 2 x 5 x 2 x 2 = 480 combinations

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	1	1	2	2	1
3	0	1	0	2	2	1	2
4	0	1	2	0	1	2	3
5	0	2	1	2	1	0	4
6	0	2	2	1	0	1	5
7	1	0	0	2	1	2	5
8	1	0	2	0	2	1	4
9	1	1	1	1	1	1	0
10	1	1	2	2	0	0	1
11	1	2	0	1	2	0	3
12	1	2	1	0	0	2	2
13	2	0	1	2	0	1	3
14	2	0	2	1	1	0	2
15	2	1	0	1	0	2	4
16	2	1	1	0	2	0	5
17	2	2	0	0	1	1	1
18	2	2	2	2	2	2	0

$L_{18}(3^6 6^1)$

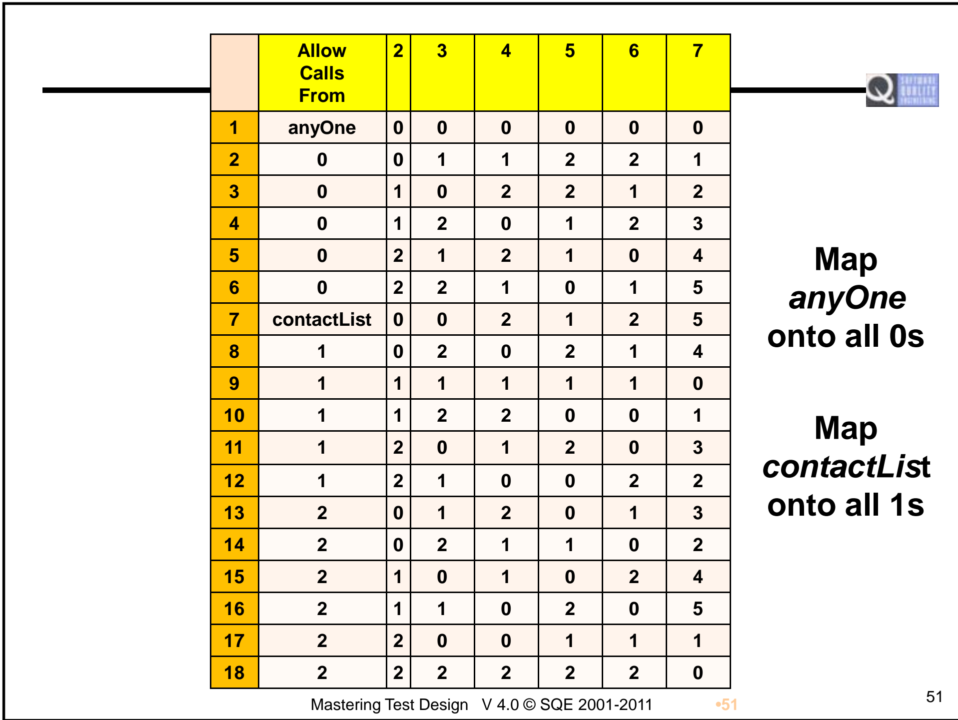
Orthogonal Arrays

- To use Orthogonal Arrays for test selection, we'll "map" our testing problem onto an OA
- (Generally) Proceed from top to bottom and left to right through the problem variables (because the first language I learned reads left to right and top to bottom)

A Problem To Solve

- Allow calls from – anyone, contact list (2)
- Receive video from – anyone, contact list, no one (3)
- Allow IMs from – anyone, contact list (2)
- Show video to – contact list, no one (2)
- Keep history for – no history, 2 weeks, 1 month, 3 months, forever (5)
- Show online status – yes, no (2)
- Accept cookies – yes, no (2)

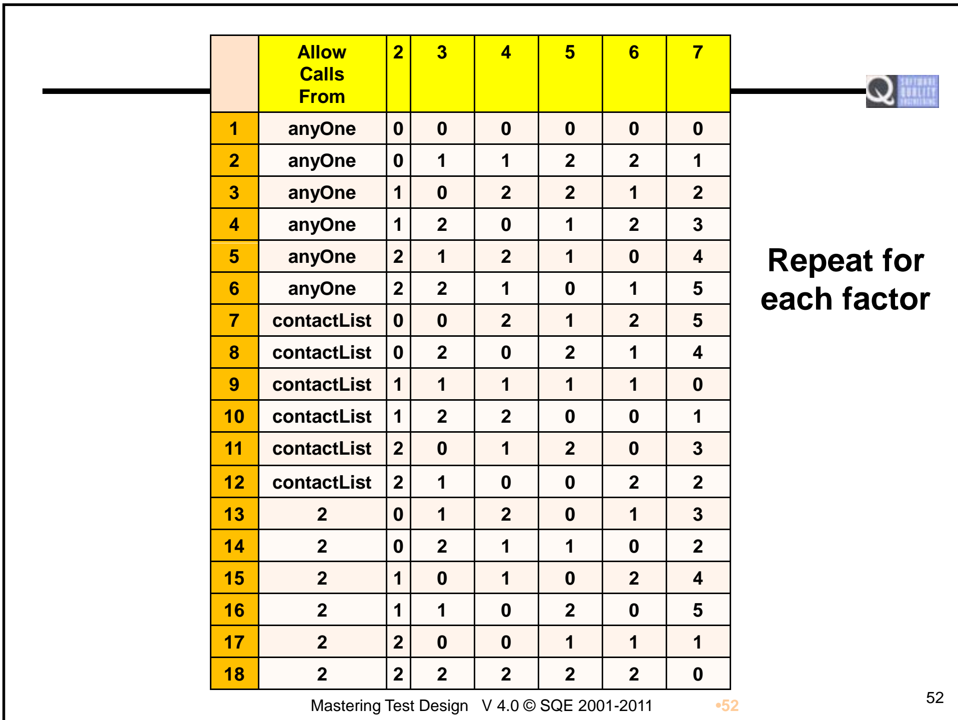
2 x 3 x 2 x 2 x 5 x 2 x 2 = 480 combinations



	Allow Calls From	2	3	4	5	6	7
1	anyOne	0	0	0	0	0	0
2	0	0	1	1	2	2	1
3	0	1	0	2	2	1	2
4	0	1	2	0	1	2	3
5	0	2	1	2	1	0	4
6	0	2	2	1	0	1	5
7	contactList	0	0	2	1	2	5
8	1	0	2	0	2	1	4
9	1	1	1	1	1	1	0
10	1	1	2	2	0	0	1
11	1	2	0	1	2	0	3
12	1	2	1	0	0	2	2
13	2	0	1	2	0	1	3
14	2	0	2	1	1	0	2
15	2	1	0	1	0	2	4
16	2	1	1	0	2	0	5
17	2	2	0	0	1	1	1
18	2	2	2	2	2	2	0

Map *anyOne* onto all 0s

Map *contactList* onto all 1s



	Allow Calls From	2	3	4	5	6	7
1	anyOne	0	0	0	0	0	0
2	anyOne	0	1	1	2	2	1
3	anyOne	1	0	2	2	1	2
4	anyOne	1	2	0	1	2	3
5	anyOne	2	1	2	1	0	4
6	anyOne	2	2	1	0	1	5
7	contactList	0	0	2	1	2	5
8	contactList	0	2	0	2	1	4
9	contactList	1	1	1	1	1	0
10	contactList	1	2	2	0	0	1
11	contactList	2	0	1	2	0	3
12	contactList	2	1	0	0	2	2
13	2	0	1	2	0	1	3
14	2	0	2	1	1	0	2
15	2	1	0	1	0	2	4
16	2	1	1	0	2	0	5
17	2	2	0	0	1	1	1
18	2	2	2	2	2	2	0

Repeat for each factor

	Allow Calls From	Receive Video From	3	4	5	6	7
1	anyOne	anyOne	0	0	0	0	0
2	anyOne	anyOne	1	1	2	2	1
3	anyOne	contactList	0	2	2	1	2
4	anyOne	contactList	2	0	1	2	3
5	anyOne	noOne	1	2	1	0	4
6	anyOne	noOne	2	1	0	1	5
7	contactList	anyOne	0	2	1	2	5
8	contactList	anyOne	2	0	2	1	4
9	contactList	contactList	1	1	1	1	0
10	contactList	contactList	2	2	0	0	1
11	contactList	noOne	0	1	2	0	3
12	contactList	noOne	1	0	0	2	2
13	2	anyOne	1	2	0	1	3
14	2	anyOne	2	1	1	0	2
15	2	contactList	0	1	0	2	4
16	2	contactList	1	0	2	0	5
17	2	noOne	0	0	1	1	1
18	2	noOne	2	2	2	2	0

	Allow Calls From	Receive Video From	Allow IMs From	4	5	6	7
1	anyOne	anyOne	anyOne	0	0	0	0
2	anyOne	anyOne	contactList	1	2	2	1
3	anyOne	contactList	anyOne	2	2	1	2
4	anyOne	contactList	2	0	1	2	3
5	anyOne	noOne	contactList	2	1	0	4
6	anyOne	noOne	2	1	0	1	5
7	contactList	anyOne	anyOne	2	1	2	5
8	contactList	anyOne	2	0	2	1	4
9	contactList	contactList	contactList	1	1	1	0
10	contactList	contactList	2	2	0	0	1
11	contactList	noOne	anyOne	1	2	0	3
12	contactList	noOne	contactList	0	0	2	2
13	2	anyOne	contactList	2	0	1	3
14	2	anyOne	2	1	1	0	2
15	2	contactList	anyOne	1	0	2	4
16	2	contactList	contactList	0	2	0	5
17	2	noOne	anyOne	0	1	1	1
18	2	noOne	2	2	2	2	0

	Allow Calls From	Receive Video From	Allow IMs From	Show Video To	5	6	7
1	anyOne	anyOne	anyOne	contactList	0	0	0
2	anyOne	anyOne	contactList	noOne	2	2	1
3	anyOne	contactList	anyOne	2	2	1	2
4	anyOne	contactList	2	contactList	1	2	3
5	anyOne	noOne	contactList	2	1	0	4
6	anyOne	noOne	2	noOne	0	1	5
7	contactList	anyOne	anyOne	2	1	2	5
8	contactList	anyOne	2	contactList	2	1	4
9	contactList	contactList	contactList	noOne	1	1	0
10	contactList	contactList	2	2	0	0	1
11	contactList	noOne	anyOne	noOne	2	0	3
12	contactList	noOne	contactList	contactList	0	2	2
13	2	anyOne	contactList	2	0	1	3
14	2	anyOne	2	noOne	1	0	2
15	2	contactList	anyOne	noOne	0	2	4
16	2	contactList	contactList	contactList	2	0	5
17	2	noOne	anyOne	contactList	1	1	1
18	2	noOne	2	2	2	2	0

	Allow Calls From	Receive Video From	Allow IMs From	Show Video To	Show Online Status	6	7
1	anyOne	anyOne	anyOne	contactList	yes	0	0
2	anyOne	anyOne	contactList	noOne	2	2	1
3	anyOne	contactList	anyOne	2	2	1	2
4	anyOne	contactList	2	contactList	no	2	3
5	anyOne	noOne	contactList	2	no	0	4
6	anyOne	noOne	2	noOne	yes	1	5
7	contactList	anyOne	anyOne	2	no	2	5
8	contactList	anyOne	2	contactList	2	1	4
9	contactList	contactList	contactList	noOne	no	1	0
10	contactList	contactList	2	2	yes	0	1
11	contactList	noOne	anyOne	noOne	2	0	3
12	contactList	noOne	contactList	contactList	yes	2	2
13	2	anyOne	contactList	2	yes	1	3
14	2	anyOne	2	noOne	no	0	2
15	2	contactList	anyOne	noOne	yes	2	4
16	2	contactList	contactList	contactList	2	0	5
17	2	noOne	anyOne	contactList	no	1	1
18	2	noOne	2	2	2	2	0

	Allow Calls From	Receive Video From	Allow IMs From	Show Video To	Show Online Status	Accept Cookies	7
1	anyOne	anyOne	anyOne	contactList	yes	yes	0
2	anyOne	anyOne	contactList	noOne	2	2	1
3	anyOne	contactList	anyOne	2	2	no	2
4	anyOne	contactList	2	contactList	no	2	3
5	anyOne	noOne	contactList	2	no	yes	4
6	anyOne	noOne	2	noOne	yes	no	5
7	contactList	anyOne	anyOne	2	no	2	5
8	contactList	anyOne	2	contactList	2	no	4
9	contactList	contactList	contactList	noOne	no	no	0
10	contactList	contactList	2	2	yes	yes	1
11	contactList	noOne	anyOne	noOne	2	yes	3
12	contactList	noOne	contactList	contactList	yes	2	2
13	2	anyOne	contactList	2	yes	no	3
14	2	anyOne	2	noOne	no	yes	2
15	2	contactList	anyOne	noOne	yes	2	4
16	2	contactList	contactList	contactList	2	yes	5
17	2	noOne	anyOne	contactList	no	no	1
18	2	noOne	2	2	2	2	0

	Allow Calls From	Receive Video From	Allow IMs From	Show Video To	Show Online Status	Accept Cookies	Keep History For
1	anyOne	anyOne	anyOne	contactList	yes	yes	noHistory
2	anyOne	anyOne	contactList	noOne	2	2	2Weeks
3	anyOne	contactList	anyOne	2	2	no	1Month
4	anyOne	contactList	2	contactList	no	2	3Months
5	anyOne	noOne	contactList	2	no	yes	forEver
6	anyOne	noOne	2	noOne	yes	no	5
7	contactList	anyOne	anyOne	2	no	2	5
8	contactList	anyOne	2	contactList	2	no	forEver
9	contactList	contactList	contactList	noOne	no	no	noHistory
10	contactList	contactList	2	2	yes	yes	2Weeks
11	contactList	noOne	anyOne	noOne	2	yes	3Months
12	contactList	noOne	contactList	contactList	yes	2	1Month
13	2	anyOne	contactList	2	yes	no	3Months
14	2	anyOne	2	noOne	no	yes	1Month
15	2	contactList	anyOne	noOne	yes	2	forEver
16	2	contactList	contactList	contactList	2	yes	5
17	2	noOne	anyOne	contactList	no	no	2Weeks
18	2	noOne	2	2	2	2	noHistory

Orthogonal Arrays

- We have reduced our test cases from 480 to 18 – a substantial reduction (about 96%)
- while testing all pairs of input combinations at least once

Orthogonal Arrays

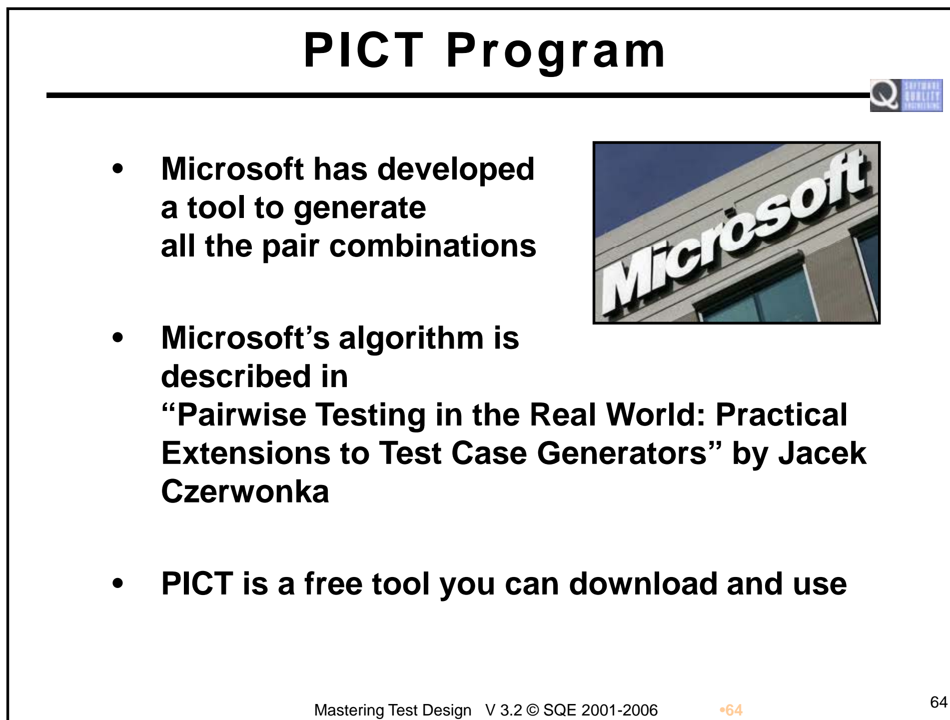
- But what should we do about the cells in the OA that have not been assigned values?
  -
  -
- But first, why do we have such cells?

	Allow Calls From	Receive Video From	Allow IMs From	Show Video To	Show Online Status	Accept Cookies	Keep History For
1	anyOne	anyOne	anyOne	contactList	yes	yes	noHistory
2	anyOne	anyOne	contactList	noOne	2	2	2Weeks
3	anyOne	contactList	anyOne	2	2	no	1Month
4	anyOne	contactList	2	contactList	no	2	3Months
5	anyOne	noOne	contactList	2	no	yes	forEver
6	anyOne	noOne	2	noOne	yes	no	5
7	contactList	anyOne	anyOne	2	no	2	5
8	contactList	anyOne	2	contactList	2	no	forEver
9	contactList	contactList	contactList	noOne	no	no	noHistory
10	contactList	contactList	2	2	yes	yes	2Weeks
11	contactList	noOne	anyOne	noOne	2	yes	3Months
12	contactList	noOne	contactList	contactList	yes	2	1Month
13	2	anyOne	contactList	2	yes	no	3Months
14	2	anyOne	2	noOne	no	yes	1Month
15	2	contactList	anyOne	noOne	yes	2	forEver
16	2	contactList	contactList	contactList	2	yes	5
17	2	noOne	anyOne	contactList	no	no	2Weeks
18	2	noOne	2	2	2	2	noHistory

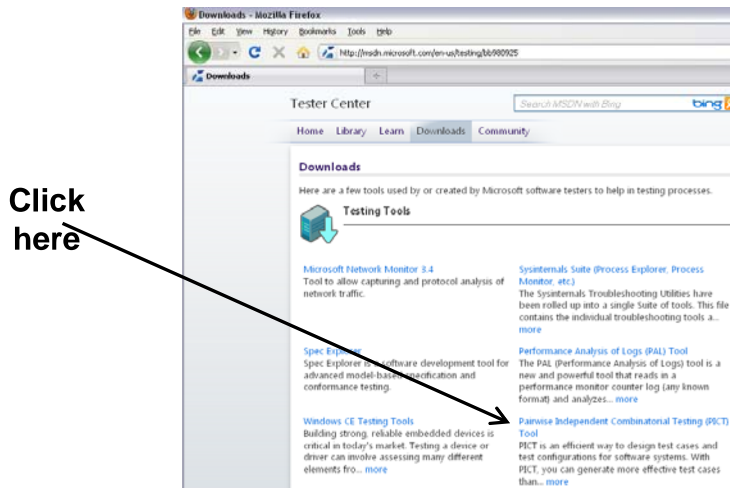
Orthogonal Arrays

- Let's fill them in with valid values:
  - Select the first value in the list
  - Alternate the values
  - Set the values according to the percentage of use (if 60% will be yes and 40% will be no, then use that percentage)
  - Set the values according to the risk that value presents
- Which is the best approach?

	Allow Calls From	Receive Video From	Allow IMs From	Show Video To	Show Online Status	Accept Cookies	Keep History For
1	anyOne	anyOne	anyOne	contactList	yes	yes	noHistory
2	anyOne	anyOne	contactList	noOne	yes	yes	2Weeks
3	anyOne	contactList	anyOne	contactList	no	no	1Month
4	anyOne	contactList	anyOne	contactList	no	no	3Months
5	anyOne	noOne	contactList	noOne	no	yes	forEver
6	anyOne	noOne	contactList	noOne	yes	no	noHistory
7	contactList	anyOne	anyOne	contactList	no	yes	2Weeks
8	contactList	anyOne	anyOne	contactList	yes	no	forEver
9	contactList	contactList	contactList	noOne	no	no	noHistory
10	contactList	contactList	contactList	noOne	yes	yes	2Weeks
11	contactList	noOne	anyOne	noOne	no	yes	3Months
12	contactList	noOne	contactList	contactList	yes	no	1Month
13	anyOne	anyOne	contactList	contactList	yes	no	3Months
14	contactList	anyOne	anyOne	noOne	no	yes	1Month
15	anyOne	contactList	anyOne	noOne	yes	yes	forEver
16	ContactList	contactList	contactList	contactList	yes	yes	1Month
17	anyOne	noOne	anyOne	contactList	no	no	2Weeks
18	contactList	noOne	contactList	noOne	no	no	noHistory



PICT Program



•http://msdn.microsoft.com/en-us/testing/bb980925

PICT Program

- … and download pict33.msi (a Microsoft Installer Package)
- Save it
- Execute it to install. (It installs in the Program Files folder)
- PICTHelp.html has the instructions (They are very helpful)

PICT Program

- **Prepare the input by entering the data into Notepad**

```
SkypeExampleForPICT - Notepad
AllowCalls: Anyone, ContactList
ReceiveVideo: Anyone, ContactList, NoOne
AllowIMs: Anyone, ContactList
ShowVideo: ContactList, NoOne
KeepHistory: NoHistry, TwoWeeks, OneMonth, ThreeMonths, Forever
ShowStatus: Yes, No
AcceptCookies: Yes, No
```


PICT Program

- **Run pict.exe**
- **Specify it's input and output. For example:**
  - **pict SkypeExampleForPICT.txt > SkypeTestCasesFromPICT.txt**

```
Command Prompt
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Lee Copeland.SQE-1D6952E4469>cd\

C:\>cd downloads

C:\DOWNLOADS>pict SkypeExampleForPICT.txt > SkypeTestCasesFromPICT.txt

C:\DOWNLOADS>
```

PICT Program

- **Open the output with Excel to make it look nice**

PICT Program

AllowCalls	ReceiveVideo	AllowIMs	ShowVideo	KeepHistory	ShowStatus	AcceptCookies
ContactList	ContactList	Anyone	ContactList	OneMonth	Yes	Yes
Anyone	Anyone	ContactList	NoOne	ThreeMonths	No	No
ContactList	NoOne	Anyone	NoOne	Forever	No	Yes
Anyone	ContactList	ContactList	NoOne	NoHistry	Yes	No
ContactList	NoOne	ContactList	ContactList	TwoWeeks	Yes	No
Anyone	NoOne	Anyone	ContactList	ThreeMonths	Yes	Yes
ContactList	Anyone	Anyone	ContactList	NoHistry	No	Yes
Anyone	ContactList	ContactList	ContactList	Forever	Yes	No
Anyone	NoOne	ContactList	NoOne	OneMonth	No	No
ContactList	Anyone	Anyone	NoOne	Forever	Yes	No
Anyone	Anyone	Anyone	NoOne	TwoWeeks	No	Yes
ContactList	ContactList	ContactList	NoOne	ThreeMonths	No	Yes
Anyone	Anyone	Anyone	ContactList	OneMonth	No	No
Anyone	ContactList	ContactList	NoOne	TwoWeeks	No	Yes
ContactList	NoOne	ContactList	ContactList	NoHistry	Yes	No

PICT Program

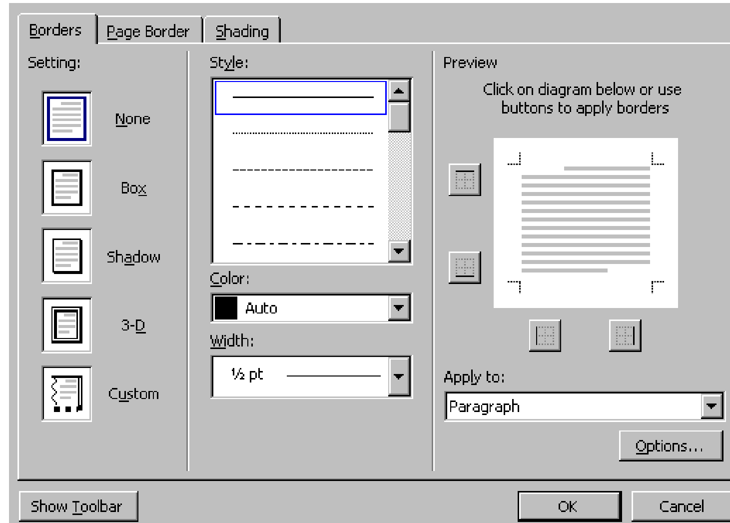
- **Note the good news – we can test all pairs of the 7 inputs in only 16 test cases (rather than the total combinations of 480)**

PICT Program

- **PICT has a nice feature the other approaches don't have – it allows for selection constraints to be specified**

```
Type:           Single, Spanned, Striped, Mirror, RAID-5
Size:           10, 100, 1000, 10000, 40000
Format method:  Quick, Slow
File system:    FAT, FAT32, NTFS
Cluster size:   512, 1024, 2048, 4096, 8192, 16384
Compression:    On, Off

# There are limitations on volume size

IF [File system] = "FAT"   THEN [Size] <= 4096;
IF [File system] = "FAT32" THEN [Size] <= 32000;

# And not all file systems support compression

IF [File system] <> "NTFS" or
  ([File system] =  "NTFS" and [Cluster size] > 4096)
THEN [Compression] = "Off";
```

Another Example



Another Example

The borders and shading dialog box :

- **5 settings**
- **5 styles (showing)**
- **9 colors**
- **9 widths**

5 x 5 x 9 x 9 = 2025 combinations

An $L_{81}(9^{10})$ array is the smallest array that meets our needs

(There is an $L_{81}(3^{24}9^{4})$ array that would work)

Sources

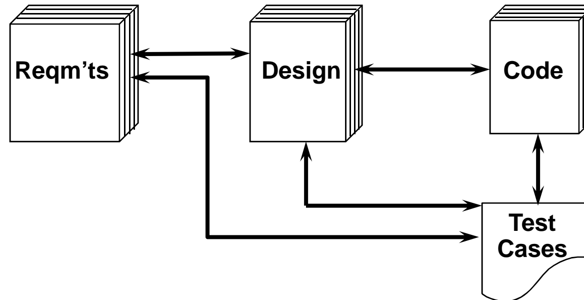
- An excellent book is ***Quality Engineering Using Robust Design*** **by Madhav S. Phadke**
- **For a catalog of orthogonal arrays, see www.research.att.com/~njas/oadir/index.html**
- **A nice tool is rdExpert from www.phadkeassociates.com**
- **A very robust tool is AETG from Telcordia at aetgweb.argreenhouse.com**

Trace Matrix

- **A Trace Matrix can be used to track test cases during development and execution**
  - Identification of uniquely "testable" requirements
    - Eliminate wishes, hopes, dreams, and other un-testable statements
    - Eliminate duplicates
    - Assign a unique ID for each requirement and design objective
    - Determine its priority if possible
  - Verification of implementation of all requirements

Trace Matrix - Context

Track everything!



Trace Matrix - Example

Trace Number	Requirement		Design	
	Sect.	Summary	Sect.	Summary
001	1.1	99.99% availability	1.6 1.7 6.1 15.3	Dual processors Hot back-up On-line diagnostics On-line performance monitors
N/A	1.2	Hardware selected will be available in less than 30 days	N/A	N/A
N/A	1.3	Transactions will not be lost (included with 1.4)	N/A	N/A
002	1.4	100% of all transactions input will be processed (includes 1.3)	2.2 2.3 2.4	Transaction processing Transaction logs Retransmission request for missing transactions

Techniques vs. Levels of Test

Method	Unit	Integ.	Sys	Sys to Sys	Web
Equivalence class partitioning	√	√	√	√	√
Boundary value	√	√	√	√	√
Invalid combinations and processes	√	√	√	√	√
Cross functional testing		√	√	√	√
Decision table	√	√	√	√	√
State transition diagrams	√	√	√	√	√
Orthogonal Arrays and All Pairs	√	√	√	√	√
Trace Matrix	√	√	√	√	√

Risk / Priority Based

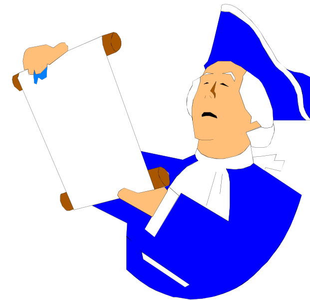
- **Add more test cases for higher risk or priority**
  - Some of the factors in risk
    - High use software
    - Safety criticality
    - Criticality to operations
    - “Key” features such as security, monetary expenditures
    - Others?
  - Including risk in the planning from the beginning can help when there isn’t time to run all of the tests
  - Risk is a subjective judgment call
  - Test higher risk areas early and more thoroughly

Chapter 3

White Box Science

Tutorial Agenda

1. Introduction
2. Black Box Science
3. ► White Box Science
4. Black Box Art
5. Defect Taxonomies
6. Wrap-up

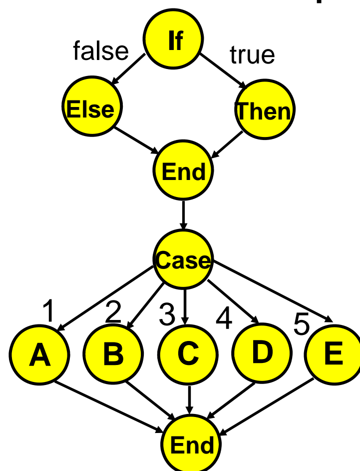


Objectives

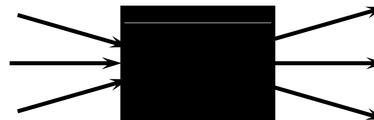
- **At the end of this chapter you will be able to**
  - Describe the differences between white and black box tests
  - Define coverage possibilities for unit, integration, and system level white box tests

What is White Box?

White box: Structure/path



Black box: Behavior



The Art of Software Testing
Glenford Myers

White Box Definitions

- **The focus of White Box testing**
  - Examine paths in the implementation
  - Make sure that each statement, decision branch, or path is tested with at least one test case
  - Definitions of coverage can vary by level of test
  - Desirable to use tools to analyze and track coverage

Understanding Coverage

- **What does "coverage" mean?**
  - NOT all possible combinations of data values or paths can be tested
  - Coverage is a way of defining how many of the potential paths were actually exercised by the tests
  - Coverage goals can vary by risk, trust, and level of test

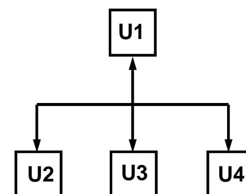
Coverage for a Unit

- **Selection: SLOC**
- **Possible strategies**
  - Statement coverage (SLOC)
  - Decision coverage (branches)
  - Paths
- **A code Sample**

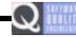
```
a=1
b=2
c=3
If (a==2)
   {n=n+2;}
Else
   {n=n/2;}
p=q/r
If (b/c>3)
   {z=x+y;}
```

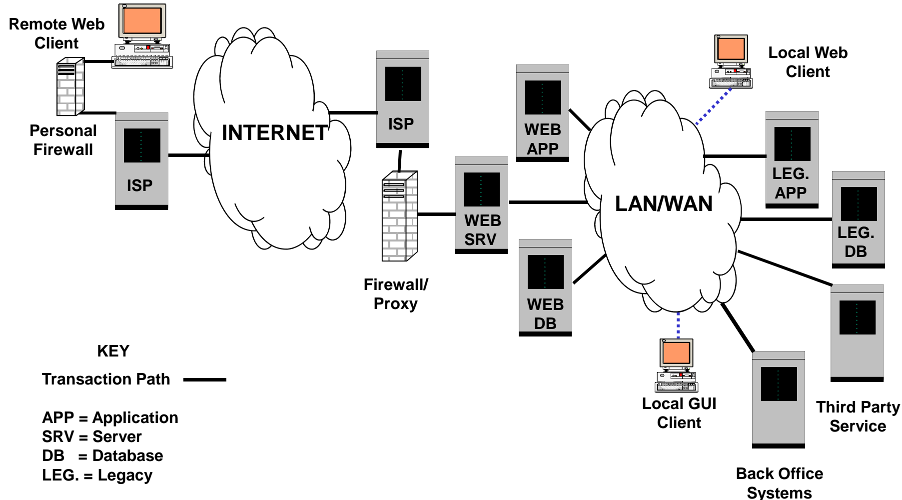

Coverage for Integration

- **Selection: Unit**
- **Possible strategies**
  - Each unit (already should have been done)
  - All choices (branches) between units
  - All paths
  - Can continue to use statement coverage measure (using a coverage tool)



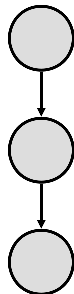
Coverage for Systems



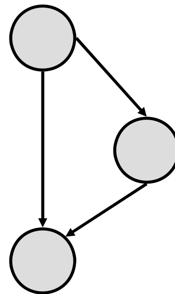


General Control Flow Concepts

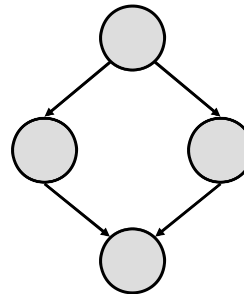
Sequence



If Then

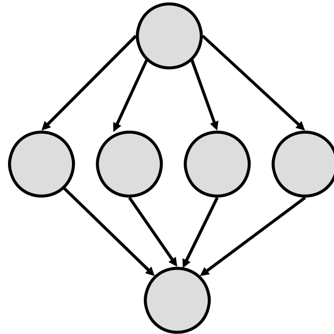


If Then Else

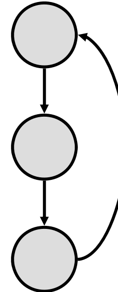


General Control Flow Concepts

CASE



Iteration

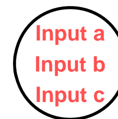


Applying Control Flow to Code

A simple code segment

```
Input a;
Input b;
Input c;
If (a==2)
   {n=n+2;}
Else
   {n=n/2;}
p=q/r
If (b/c>3)
   {z=x+y;}
```

Begin at the top

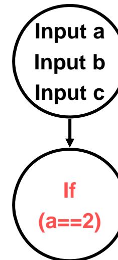


Applying Control Flow to Code

A simple code segment

```
Input a;
Input b;
Input c;
If (a==2)
   {n=n+2;}
Else
   {n=n/2;}
p=q/r
If (b/c>3)
   {z=x+y;}
```

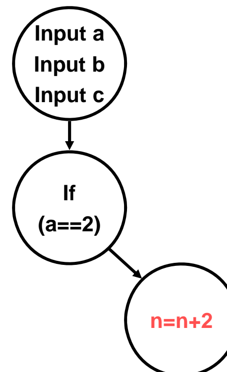
Continue the graph



Applying Control Flow to Code

A simple code segment

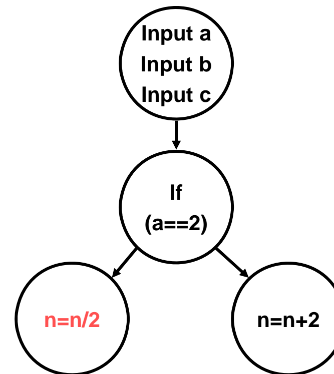
```
Input a;
Input b;
Input c;
If (a==2)
   {n=n+2;}
Else
   {n=n/2;}
p=q/r
If (b/c>3)
   {z=x+y;}
```



Applying Control Flow to Code

A simple code segment

```
Input a;
Input b;
Input c;
If (a==2)
   {n=n+2;}
Else
   {n=n/2;}
p=q/r
If (b/c>3)
   {z=x+y;}
```

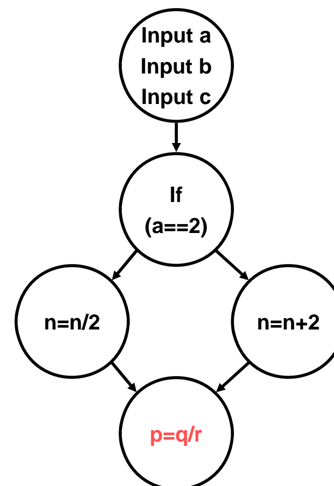


 © SQE 2001-2011

Applying Control Flow to Code

A simple code segment

```
Input a;
Input b;
Input c;
If (a==2)
   {n=n+2;}
Else
   {n=n/2;}
p=q/r
If (b/c>3)
   {z=x+y;}
```

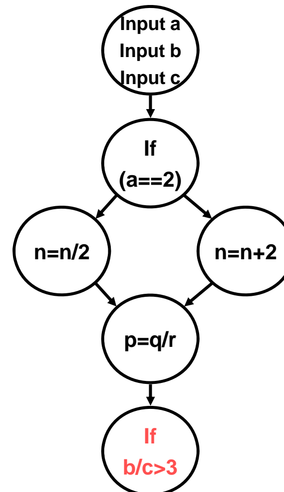


 © SQE 2001-2011

Applying Control Flow to Code

A simple code segment

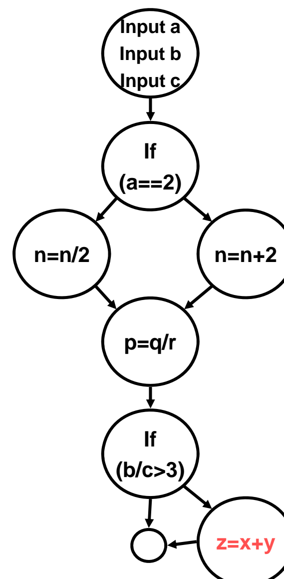
```
Input a;
Input b;
Input c;
If (a==2)
   {n=n+2;}
Else
   {n=n/2;}
p=q/r
If (b/c>3)
   {z=x+y;}
```



Applying Control Flow to Code

A simple code segment

```
Input a;
Input b;
Input c;
If (a==2)
   {n=n+2;}
Else
   {n=n/2;}
p=q/r
If (b/c>3)
   {z=x+y;}
```

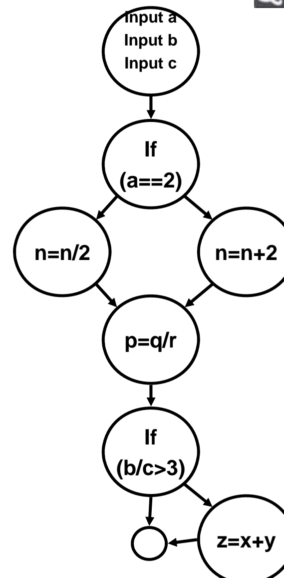


Understanding Unit Testing

- **When unit testing a module, the key question is how many tests are required to reasonably and fully exercise the code?**
  - What is the minimum number of tests that can be run to fully exercise the structure of the code?
  - How many paths are there?
  - How do I determine/create these tests?
- **There are three approaches to answering these questions, all require a flow graph**

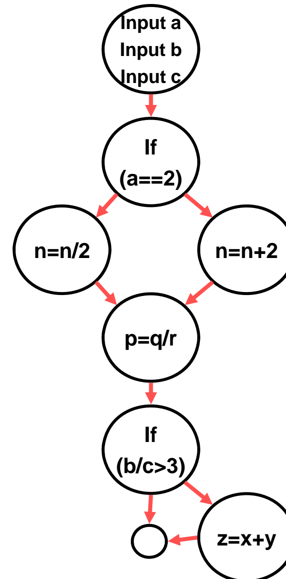
1st Method to Determine Tests

- **McCabe's technique for calculating units**
  - Convert the code to a flow graph
  - Compute the paths by e-n+2(p)
  - This is called Cyclomatic Complexity
    - The number of paths to test
    - All decision options are tested
    - Each statement is covered at least once



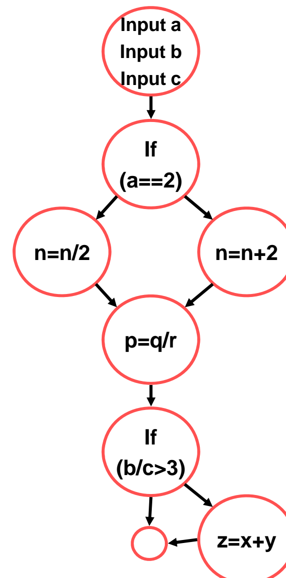
How Many Tests?

- Compute the paths by e-n+2(p)
- e = 9



How Many Tests?

- Compute the paths by e-n+2(p)
- n = 8



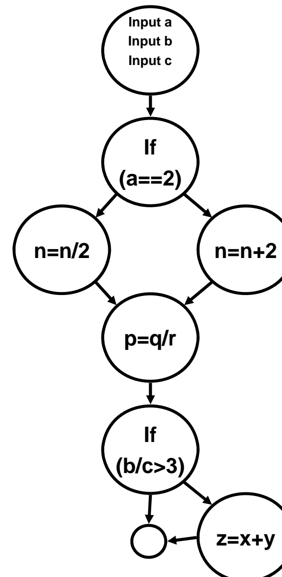
How Many Tests?

- **Compute the paths by e-n+2(p)**

  9-8+2(1) = 3

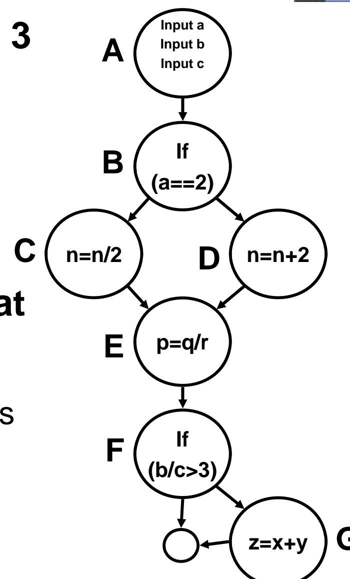
  - There are no called modules, so (p) is set to 1

- **There are 3 paths that will cover the code**
  - 100% decision coverage
  - 100% statement coverage
  - Does not guarantee 100% path coverage



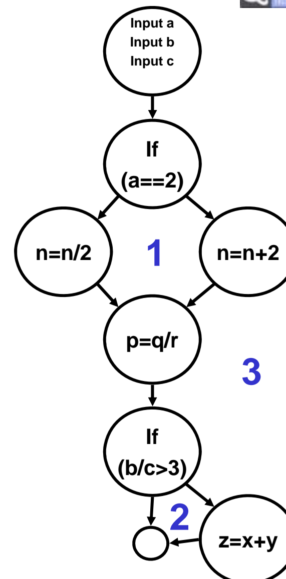
What are the Paths (tests)?

- **Cyclomatic Complexity is 3**
- **Paths are:**
  - **ABCEF**
  - **ABDEF**
  - **ABDEFG**
- **Test cases should cover at least three paths**
  - There is more than one “basis set” of paths



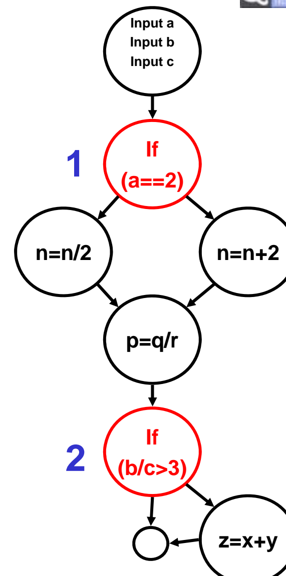
2nd Method to Determine Tests

- **The next approach is to manually calculate the regions of the graph**
  - Convert the code to a flow graph
  - Count the regions of the flow graph (including the exterior)
    - An area enclosed by lines (edges) is a region
    - The external area is always a region
- **This can be expressed as**
  - Regions + 1



3rd Method to Determine Tests

- **The third approach is to manually calculate the decisions in the graph**
  - Convert the code to a flow graph
  - Count the decision nodes in the flow graph
- **This can be expressed as**
  - Decisions + 1



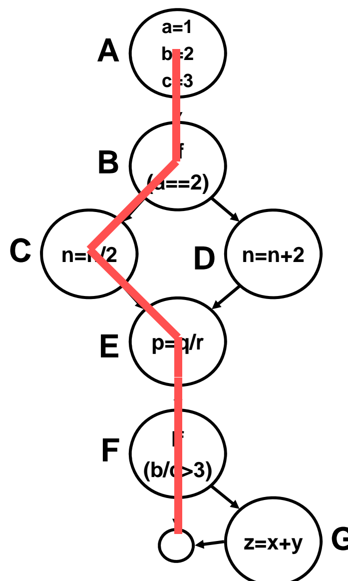
Number of Paths (Tests)

- **You will notice that all three methods generate the same result**
  - CC 9-8+2(1) = 3
  - Regions 2+1 = 3
  - Decisions 2+1 = 3

- **Regions and decisions require the construction of a graph or manual analysis of the code**
- **Cyclomatic complexity can be calculated using a tool, or done manually**

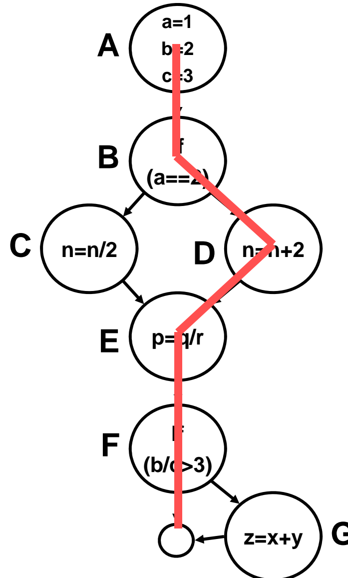
So, What are the Paths (Tests)?

- **Path one:**
  - ABCEF



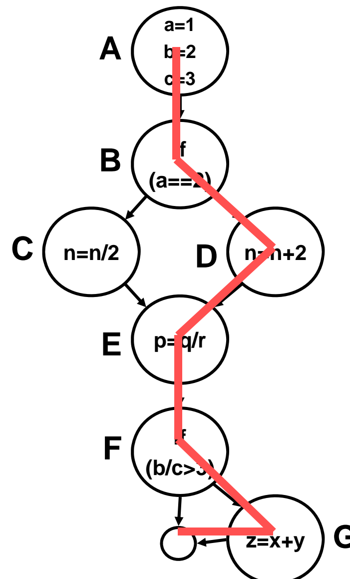
So, What are the Paths (Tests)?

- **Path two:**
  - ABDEF



So, What are the Paths (Tests)?

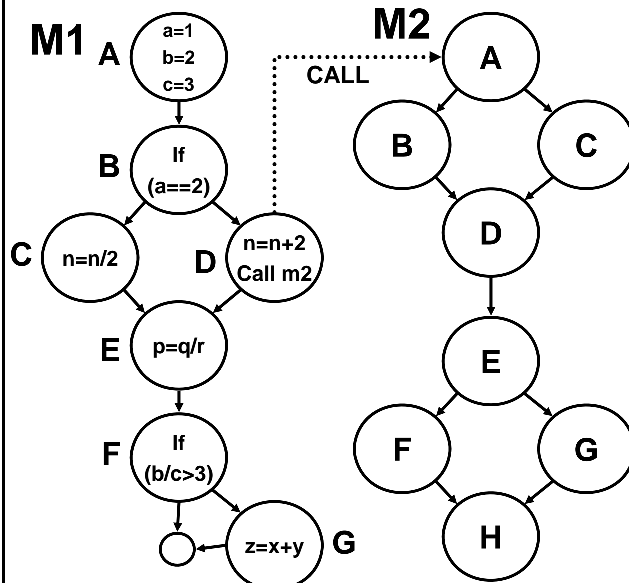
- **Path three:**
  - ABDEFG



What about (P) ?

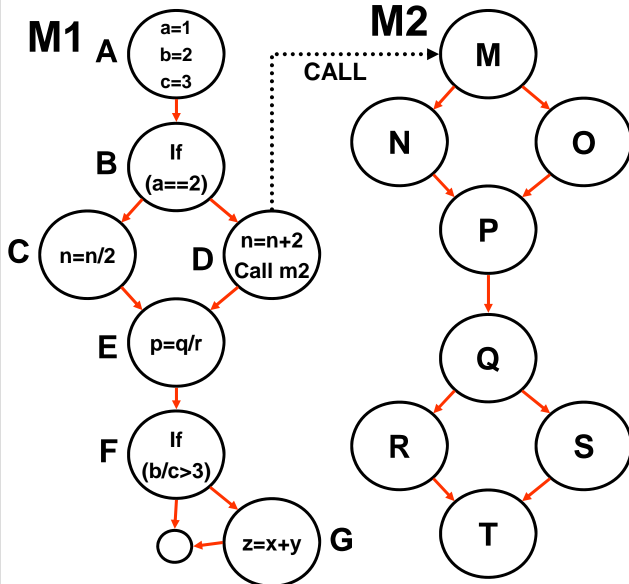
- **If there are called programs the computation must account for those modules as well**
- **The change is in how the edges (E) and nodes (N) are determined.**
  - P represents the number of unconnected parts of the graph

What about (P)?



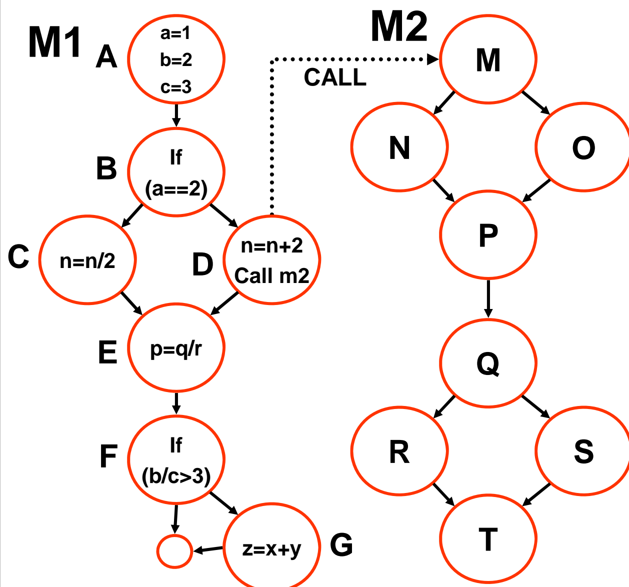
- **For multiple graphs**
  - Module 2 is called by module 1

What about (P)?



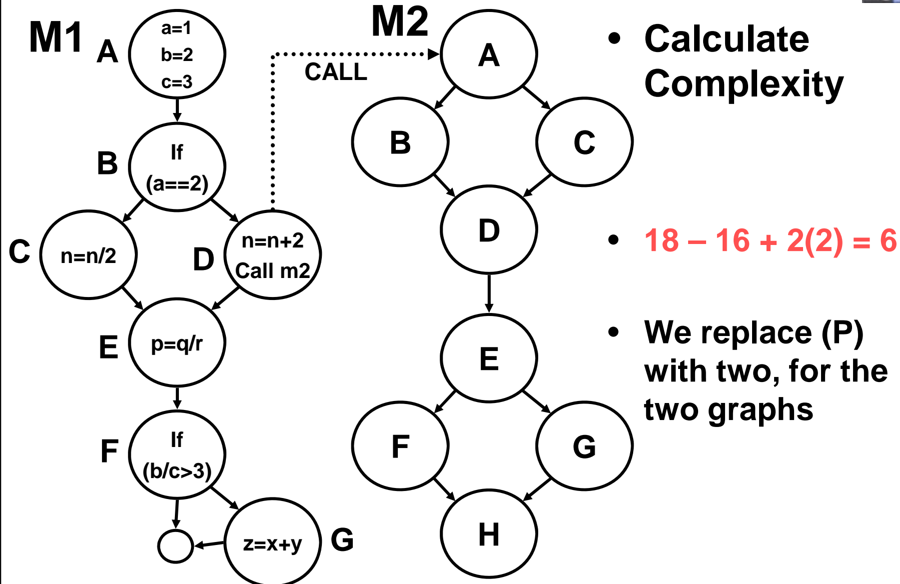
- **Sum the total edges**
  - Module 1
    - 9 edges
  - Module 2
    - 9 edges
  - Total
    - 18 edges

What about (P)?



- **Sum the total nodes**
  - Module 1
    - 8 nodes
  - Module 2
    - 8 nodes
  - Total
    - 16 nodes

Calculate the Total Complexity



- Calculate Complexity
- 18 – 16 + 2(2) = 6
- We replace (P) with two, for the two graphs

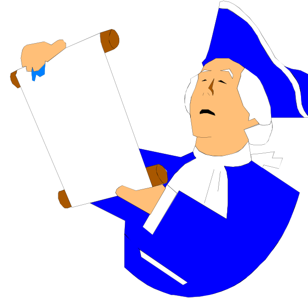
Chapter 4

Black Box

Art

Tutorial Agenda

1. Introduction
2. Black Box Science
3. White Box Science
4. Black Box Art
5. Defect Taxonomies
6. Wrap-up



Objectives

- **At the end of this chapter you will be able to**
  - Implement creative test case selection techniques
  - Evaluate the use of exploratory testing

Why Science is Not Enough

- **Limits of science**
  - Programmers are logical thinkers, so they catch many of the “logical’ defects
  - Real users are NOT necessarily logical
  - Real environmental circumstances are often illogical
  - Can’t test enough combinations with just scientific techniques

Hunches and Guessing

- **Talent based testing**
  - Based on the implementation
  - Some individuals can walk up to a system and “break” it
  - Good to have multi-dimensional teams doing the testing
  - Experienced testers can just “know” what is likely to break

Bet I can break it!!!

Group Insights / Examples

- **Experience can be shared with examples**
  - "Best practices" examples of test cases posted on an internal web site
    - "Best" is a relative term; what's good on one system/application may not work on another
  - Examples of good tests based on type of element being tested
    - Compiled by experienced personnel
    - Available to everyone
    - Can be a standard, guidelines, or suggestions

Exploratory Testing

"The classical approach to test design is like playing '20 Questions' by writing out all the questions in advance."

- James Bach

Exploratory Testing

"Exploratory Testing, as I practice it, usually proceeds according to a conscious plan. But not a rigorous plan … it is not scripted in detail."

"To the extent that the next test we do is influenced by the result of the last test we did, we are doing exploratory testing. We become more exploratory when we can't tell what tests should be run in advance of the test cycle."

- James Bach

Exploratory Testing

- **Test cases themselves are NOT pre-planned**
  - Exploratory testing can be concurrent with product development and test execution
  - Based on implicit and explicit (if they exist) specifications as well as the "as built" product
  - Starts with a conjecture as to correct behavior, followed by exploration for evidence that it works/does not work
  - Based on some kind of mental model
  - "Try it and see if it works" (James Bach www.satisfice.com is an excellent resource for more on this topic)

Exploratory Testing

- **Basic steps to exploratory testing**

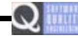
1. Identify the purpose of the product
2. Identify functions
3. Identify areas of potential instability
4. Test each function and record problems
5. Design and record a consistency verification test

Create Creative Invalids

- **Break the rules in unexpected ways**
  - No logic
  - Strange
    - Order
    - Combinations
    - Exception conditions
  - The “nobody would ever do that” tests

- **Boris Beizer: “It helps to have some devious testers.”**

Class Exercise



- **Pick your technique**
  - Data in the range 0.0-1.0 should be processed using algorithm A, data between 1.0 and 10.0 using algorithm B, and data between 10.0 and 100.0 using algorithm C.
  - We've got very few written requirements for this system. We do, however, know something about how the system is to operate, its behavior, and functions.

Class Exercise

- **Pick your technique** ***(continued)***
  - If condition 1 (C1) is true and condition 2 (C2) is true, then action 1 (A1) should be taken. If C1 is true and C2 is false, then A2 is the proper response. If C1 is false and C2 is true, then A3 is appropriate. If C1 is false and C2 is false, then A4 is the correct action.

Class Exercise

- **Pick your technique** ***(continued)***
  - In state 1 (S1) events 1 (E1) and 2 (E2) are valid while event 3 (E3) is not. In state 2 (S2) events 1 (E1) and 3 (E3) are valid while event 2 (E2) is not. In state 3 (S3) events 2 (E2) and 3 (E3) are valid while event 1 (E1) is not. If the system is in S1 and E2 occurs, then action 1 (A1) should be taken. If the system is in S3 and E3 occurs, then action 2 (A2) should be taken.

Class Exercise

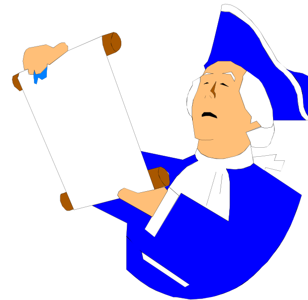
- **Pick your technique** ***(continued)***
  - I've got so many combinations to test I just can't do them all. Please help!

Chapter 5

Defect Taxonomies

Tutorial Agenda

1. Introduction
2. Black Box Science
3. White Box Science
4. Black box Art
5. ► Defect Taxonomies
6. Wrap-up



Using a Taxonomy

- A taxonomy is a classification of things into ordered groups or categories that indicate natural, hierarchical relationships.
  - In software test design we are concerned with taxonomies of defects, ordered lists of common defects we expect to encounter in our testing.

Sample Taxonomies

- One of the first defect taxonomies was defined by Boris Beizer in *Software Testing Techniques*.
- The classic book *Testing Computer Software* by Cem Kaner contains a detailed taxonomy of more than 400 types of defects.
- Robert Binder notes that many defects in the object-oriented paradigm are problems using encapsulation, inheritance, polymorphism, message sequencing, and state-transitions.

Sample Taxonomies

- James Whittaker's book *How to Break Software* is a tester's delight. Proponents of exploratory testing exhort us to "explore." Whittaker tells us specifically "where to explore." Not only does he identify areas in which faults tend to occur, but he defines specific testing attacks to locate these faults.
- Giri Vijayaraghavan has chosen a much narrower focus—the eCommerce shopping cart.

Example Taxonomy

1XXX	Requirements
11XX	**Requirements incorrect**
12XX	**Requirements logic**
13XX	**Requirements, completeness**
14XX	**Verifiability**
15XX	**Presentation, documentation**
16XX	**Requirements changes**
2XXX	Features And Functionality
21XX	**Feature/function correctness**
22XX	**Feature completeness**
23XX	**Functional case completeness**
24XX	**Domain bugs**
25XX	**User messages and diagnostics**
26XX	**Exception conditions mishandled**

A Portion of Beizer's Taxonomy

Example Taxonomy *(continued)*

3XXX	Structural Bugs
31XX	**Control flow and sequencing**
32XX	**Processing**
4XXX	Data
41XX	**Data definition and structure**
42XX	**Data access and handling**
5XXX	Implementation and Coding
51XX	**Coding and typographical**
52XX	**Style and standards violations**
53XX	**Documentation**
6XXX	Integration
61XX	**Internal interfaces**
62XX	**External interfaces, timing, throughput**
7XXX	System and Software Architecture

A Portion of Beizer's Taxonomy

Example Taxonomy *(continued)*

71XX	**O/S call and use**
72XX	**Software architecture**
73XX	**Recovery and accountability**
74XX	**Performance**
75XX	**Incorrect diagnostics, exceptions**
76XX	**Partitions, overlays**
77XX	**System generation, environment**
8XXX	Test Definition and Execution
81XX	**Test design bugs**
82XX	**Test execution bugs**
83XX	**Test documentation**
84XX	**Test case completeness**

A Portion of Beizer's Taxonomy

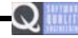
Taxonomy Sources

- Beizer, Boris (1990). *Software Testing Techniques* (2nd Edition). Van Nostrand Reinhold.
- Binder, Robert V. (2000). *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley.
- Kaner, Cem, Jack Falk, and Hung Quoc Nguyen (1999). *Testing Computer Software* (2nd Edition). John Wiley & Sons.
- Whittaker, James A. (2003). *How To Break Software: A Practical Guide to Testing*. Addison Wesley.
- Vijayaraghavan, Giri and Cem Kaner. “Bugs in your shopping cart” www.testingeducation.org/articles/BISC_Final.pdf

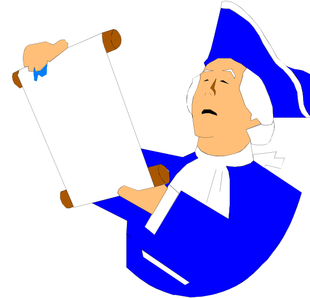
Chapter 6

Wrap Up

Tutorial Agenda



1. Introduction
2. Black Box Science
3. White Box Science
4. Black Box Art
5. Defect Taxonomies
6. ► Wrap-up



Current Challenges

?

?

?

?

?

?

?

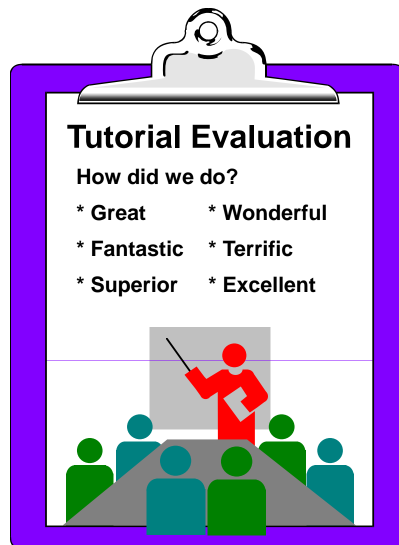
?

?

Tutorial Goals Achieved

- You are now able to
  - Design test cases using structured techniques
  - Implement the “art” of test case design as well as the “science”
  - Understand how defect taxonomies can improve test case design

Tutorial Evaluations



Thank You

- On behalf of Software Quality Engineering, thank you for attending this tutorial.
- If you have further needs for training or consulting, please think first of SQE.
- If I can be of further assistance, please let me know. My email is lee@sqe.com

Good Luck!