Software Quality Assurance

Module 7

Test & Evaluate

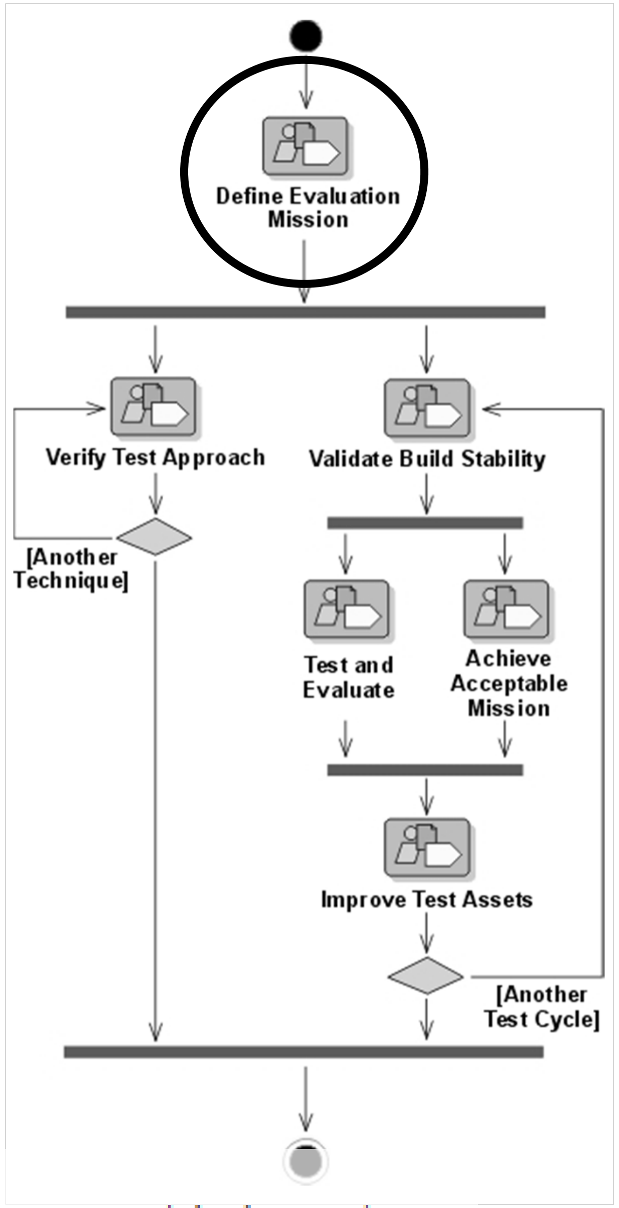
Module 7 Agenda

- Overview of the workflow: *Test and Evaluate*
- Defining test techniques—the primary types and styles of functional testing
- Individual techniques
- Using techniques together

- In the next module:
  - Analyze test failures
  - Report problems

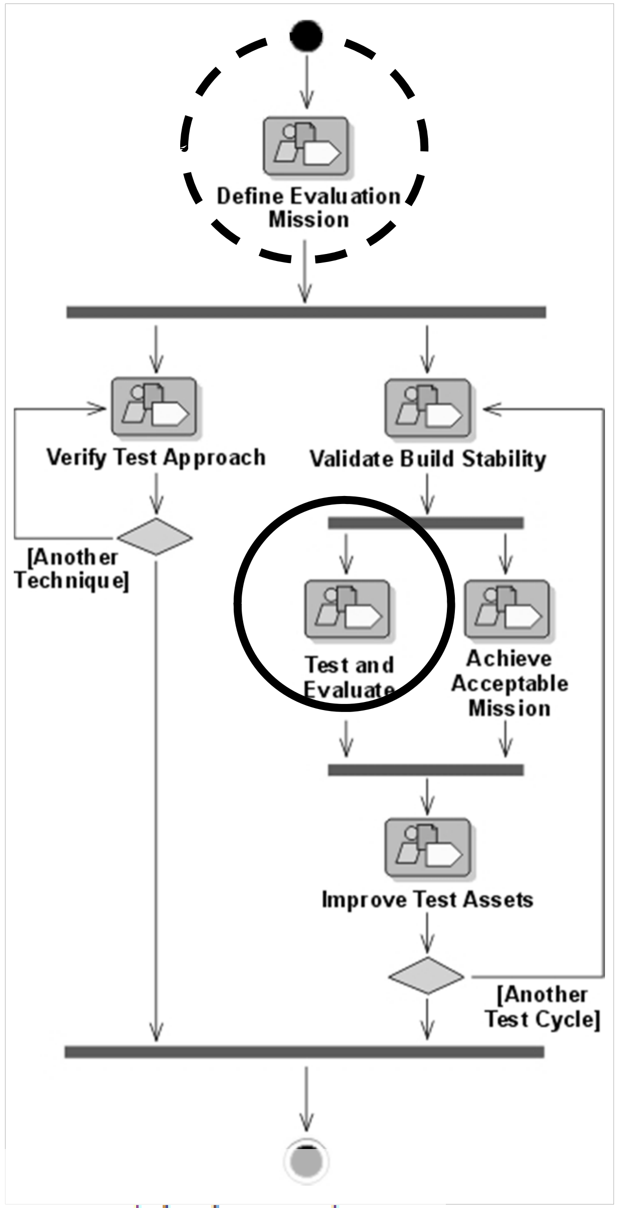
Review: Where We've Been

- In the last module, we covered the workflow detail Define Evaluation Mission
- The Mission focuses on the high-level objectives of the test team for the current iteration
  - What things should motivate us to test?
  - Why these things (and not others)?



Test and Evaluate – Part One: Test

- In this module, we drill into Test and Evaluate
- This addresses the "How?" question:
  - How will you test those things?

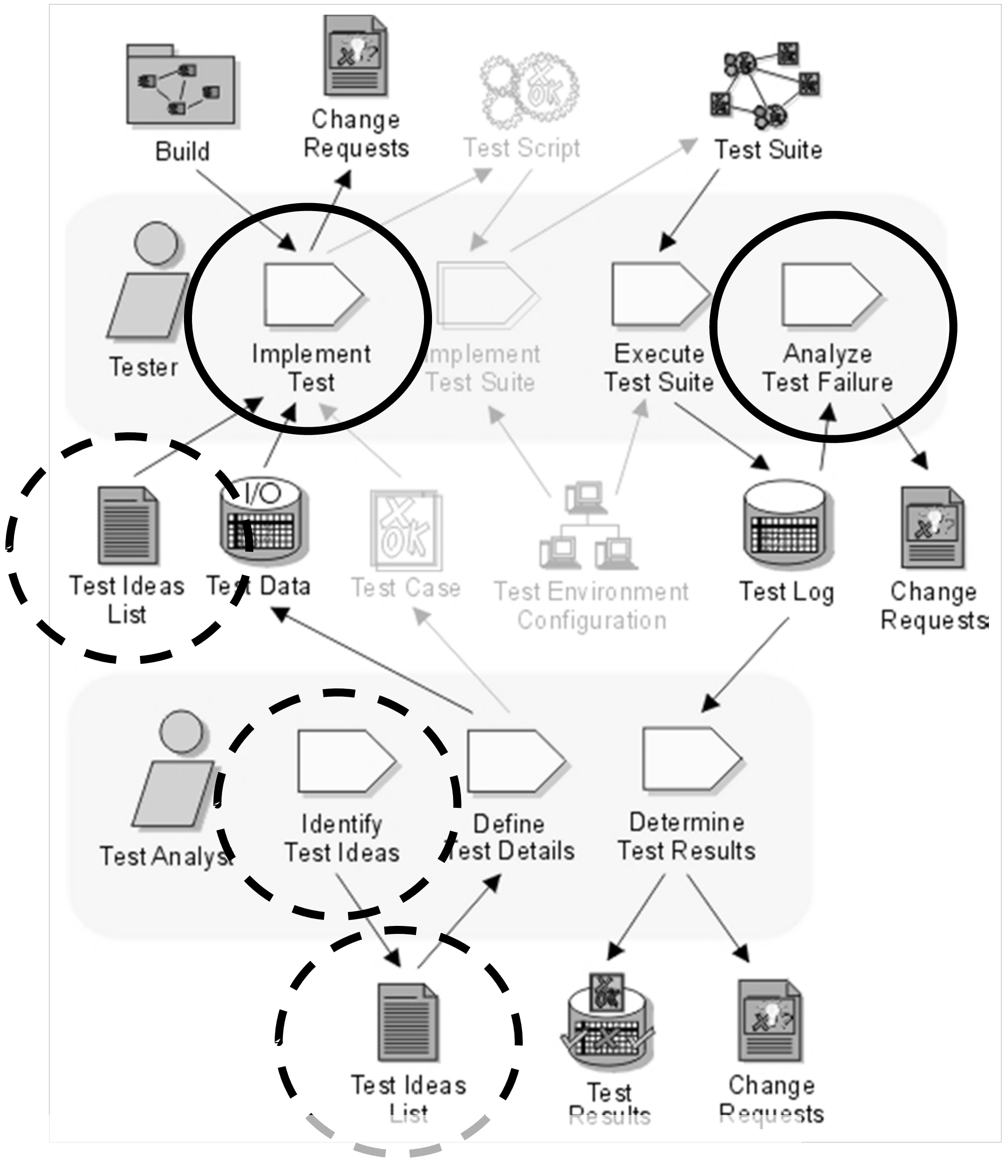


Test and Evaluate – Part One: Test

- ◆ The purpose of this workflow detail:
  - ◆ To achieve appropriate breadth and depth of the test effort to enable a sufficient evaluation of the Target Test Items — where sufficient evaluation is governed by the Test Motivators and Evaluation Mission.
- ◆ For each test cycle, this work is focused mainly on:
  - •Achieving suitable breadth and depth in the test and evaluation work
- ◆ This is the heart of the test cycle, doing the testing itself and analyzing the results.

Test and Evaluate – Part One: Test

- This module focuses on the activity *Implement Test*
- Earlier, we covered *Test-Idea Lists*, which are input here
- In the next module, we'll cover *Analyze Test Failures*, the second half of *Test and Evaluate*



Module 7 Agenda

- Overview of the workflow: *Test and Evaluate*
- **Defining test techniques**
- Individual techniques
- Using techniques together

Review: Defining the Test Approach

- In Module 6, we covered Test Approach
- A good test approach is:
  - *Diversified*
  - *Risk-focused*
  - *Product-specific*
  - *Practical*
  - *Defensible*
- The techniques you apply should follow your test approach

Discussion Exercise 7.1: Test Techniques

- There are as many as 200 published testing techniques. Many of the ideas are overlapping, but there are common themes.
- Similar sounding terms often mean different things, e.g.:
  - User testing
  - Usability testing
  - User interface testing
- What are the differences among these techniques?

Dimensions of Test Techniques

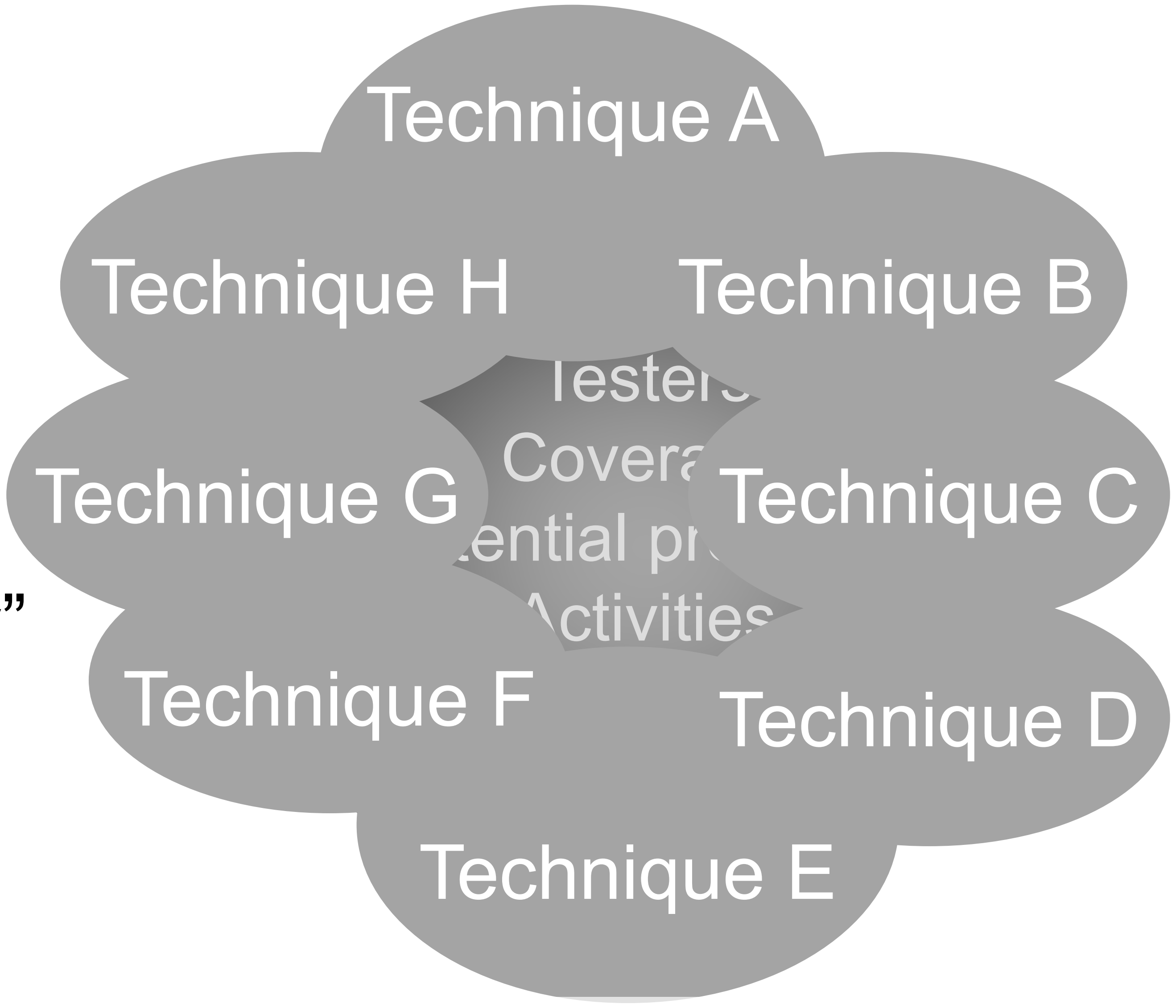
- Think of the testing you do in terms of five dimensions:
  - Testers: who does the testing.
  - Coverage: what gets tested.
  - Potential problems: why you're testing (what risk you're testing for).
  - Activities: how you test.
  - Evaluation: how to tell whether the test passed or failed.
- Test techniques often focus on one or two of these, leaving the rest to the skill and imagination of the tester.

Test Techniques—Dominant Test Approaches

- Of the 200+ published Functional Testing techniques, there are ten basic themes.
- They capture the techniques in actual practice.
- In this course, we call them:
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing

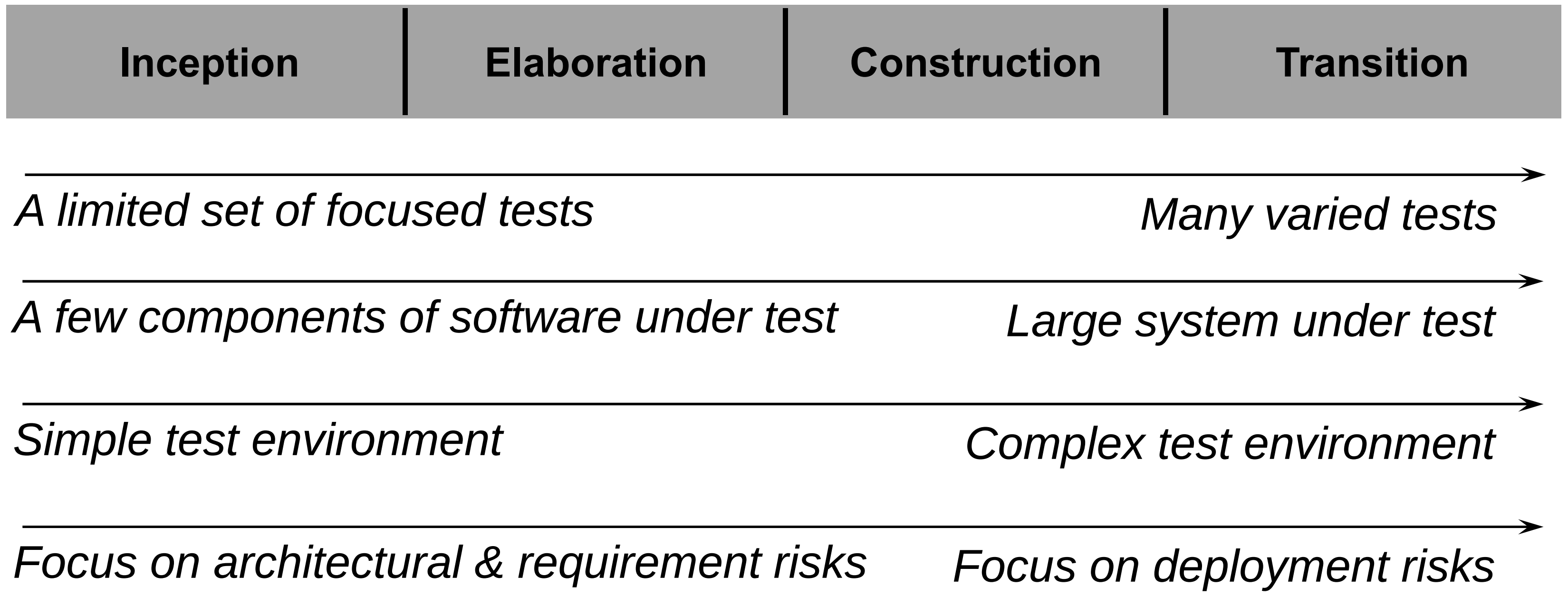
“So Which Technique Is the Best?”

- Each has strengths and weaknesses
- Think in terms of complement
- There is no “one true way”
- Mixing techniques can improve coverage



Apply Techniques According to the LifeCycle

- Test Approach changes over the project
- Some techniques work well in early phases; others in later ones
- Align the techniques to iteration objectives


Inception
Elaboration
Construction
Transition
A limited set of focused tests
Many varied tests
A few components of software under test
Large system under test
Simple test environment
Complex test environment
Focus on architectural & requirement risks
Focus on deployment risks

Module 7 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - **Function testing**
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

At a Glance: Function Testing

Tag line	Black box unit testing
Objective	Test each function thoroughly, one at a time.
Testers	Any
Coverage	Each function and user-visible variable
Potential problems	A function does not work in isolation
Activities	Whatever works
Evaluation	Whatever works
Complexity	Simple
Harshness	Varies
SUT readiness	Any stage

Strengths & Weaknesses: Function Testing

- Representative cases
  - Spreadsheet, test each item in isolation.
  - Database, test each report in isolation
- Strengths
  - Thorough analysis of each item tested
  - Easy to do as each function is implemented
- Blind spots
  - Misses interactions
  - Misses exploration of the benefits offered by the program.

Module 7 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - **Equivalence analysis**
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

At a Glance: Equivalence Analysis (1/2)

Tag line	Partitioning, boundary analysis, domain testing
Objective	There are too many test cases to run. Use stratified sampling strategy to select a few test cases from a huge population.
Testers	Any
Coverage	All data fields, and simple combinations of data fields. Data fields include input, output, and (to the extent they can be made visible to the tester) internal and configuration variables
Potential problems	Data, configuration, error handling

At a Glance: Equivalence Analysis (2/2)

Activities	Divide the set of possible values of a field into subsets, pick values to represent each subset. Typical values will be at boundaries. More generally, the goal is to find a “best representative” for each subset, and to run tests with these representatives. Advanced approach: combine tests of several “best representatives”. Several approaches to choosing optimal small set of combinations.
Evaluation	Determined by the data
Complexity	Simple
Harshness	Designed to discover harsh single-variable tests and harsh combinations of a few variables
SUT readiness	Any stage

Strengths & Weaknesses: Equivalence Analysis

- Representative cases
  - Equivalence analysis of a simple numeric field.
  - Printer compatibility testing (multidimensional variable, doesn’t map to a simple numeric field, but stratified sampling is essential)
- Strengths
  - Find highest probability errors with a relatively small set of tests.
  - Intuitively clear approach, generalizes well
- Blind spots
  - Errors that are at boundaries or in obvious special cases.
  - The actual sets of possible values are often unknowable.

Optional Exercise 7.2: GUI Equivalence Analysis

- Pick an app that you know and some dialogs
  - MS Word and its Print, Page setup, Font format dialogs
- Select a dialog
  - Identify each field, and for each field
    - What is the type of the field (integer, real, string, ...)?
    - List the range of entries that are “valid” for the field
    - Partition the field and identify boundary conditions
    - List the entries that are almost too extreme for the field
    - List a few test cases for the field and explain why the values you chose are the most powerful representatives of their sets (for showing a bug)
    - Identify any constraints imposed on this field by other fields

Optional Exercise 7.3: Data Equivalence

- *The program reads three integer values from a card. The three values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral.*
    - From Glenford J. Myers, *The Art of Software Testing* (1979)

- Write a set of test cases that would adequately test this program.

Exercise 7.3: Myers' Answers

- Test case for a valid scalene triangle
- Test case for a valid equilateral triangle
- Three test cases for valid isosceles triangles
  - (a=b, b=c, a=c)
- One, two or three sides has zero value
- One side has a negative
- Sum of two numbers equals the third (e.g. 1,2,3)
  - Invalid b/c not a triangle (tried with 3 permutations a+b=c, a+c=b, b+c=a)
- Sum of two numbers is less than the third
  - (e.g. 1,2,4) (3 permutations)
- Non-integer
- Wrong number of values (too many, too few)

Optional Exercise 7.4: Numeric Range with Output

- The program:
  - K = I * J
  - I, J and K are integer variables
- Write a set of test cases that would adequately test this program

Equivalence Partitioning

- Test-case design by equivalence partitioning proceeds in two steps:
  - (1) identifying the equivalence classes
  - (2) defining the test cases.

Identifying the Equivalence Classes

- A set of guidelines is as follows:
  - **1. If an input condition specifies a range of values (for example,** “the item count can be from 1 to 999”), identify one valid equivalence class (1 <= item count <= 999) and two invalid equivalence classes (item count < 1 and item count > 999).
  - **2. If an input condition specifies the number of values (for** example, “one through six owners can be listed for the automobile”), identify one valid equivalence class and two invalid equivalence classes (no owners and more than six owners).

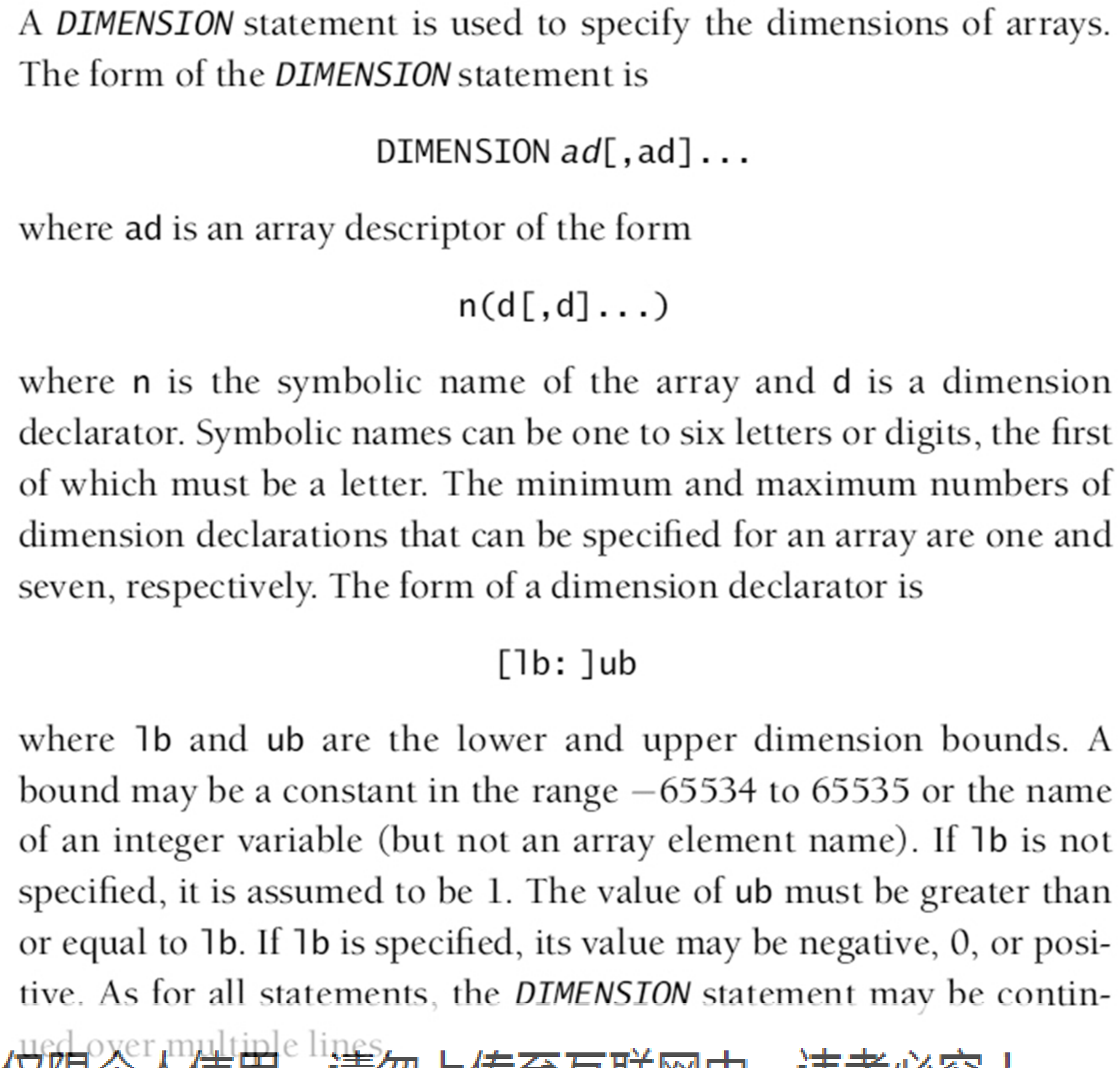
Identifying the Equivalence Classes

- A set of guidelines is as follows:
  - **3. If an input condition specifies a set of input values and there** is reason to believe that the program handles each differently (“type of vehicle must be BUS, TRUCK, TAXICAB, PASSENGER, or MOTORCYCLE”), identify a valid equivalence class for each and one invalid equivalence class (“TRAILER,” for example).
  - **4. If an input condition specifies a “must be” situation, such as** “first character of the identifier must be a letter,” identify one valid equivalence class (it is a letter) and one invalid equivalence class (it is not a letter).

Identifying the Test Cases

- The process is as follows:
- **1. Assign a unique number to each equivalence class.**
- **2. Until all valid equivalence classes have been covered by** (incorporated into) test cases, write a new test case covering as many of the uncovered valid equivalence classes as possible.
- **3. Until your test cases have covered all invalid equivalence** classes, write a test case that covers one, and only one, of the uncovered invalid equivalence classes.

Exercise 7.5: FORTRAN Dimension



A *DIMENSION* statement is used to specify the dimensions of arrays. The form of the *DIMENSION* statement is

```
DIMENSION ad[,ad]...
```

where ad is an array descriptor of the form

```
n(d[,d]...)
```

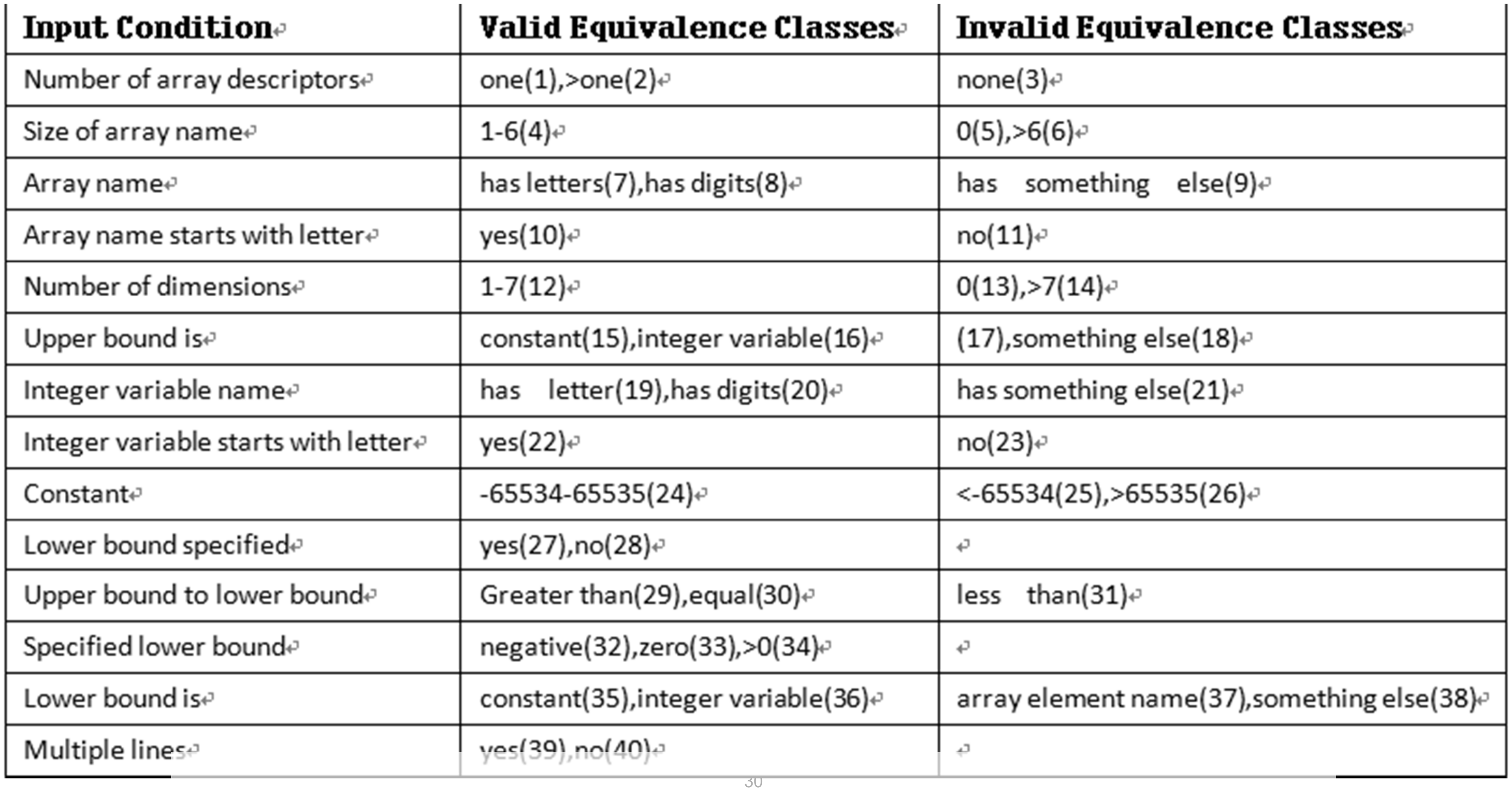
where n is the symbolic name of the array and d is a dimension declarator. Symbolic names can be one to six letters or digits, the first of which must be a letter. The minimum and maximum numbers of dimension declarations that can be specified for an array are one and seven, respectively. The form of a dimension declarator is

```
[lb: ]ub
```

where lb and ub are the lower and upper dimension bounds. A bound may be a constant in the range −65534 to 65535 or the name of an integer variable (but not an array element name). If lb is not specified, it is assumed to be 1. The value of ub must be greater than or equal to lb. If lb is specified, its value may be negative, 0, or positive. As for all statements, the *DIMENSION* statement may be continued over multiple lines.

Exercise 7.5: FORTRAN Dimension

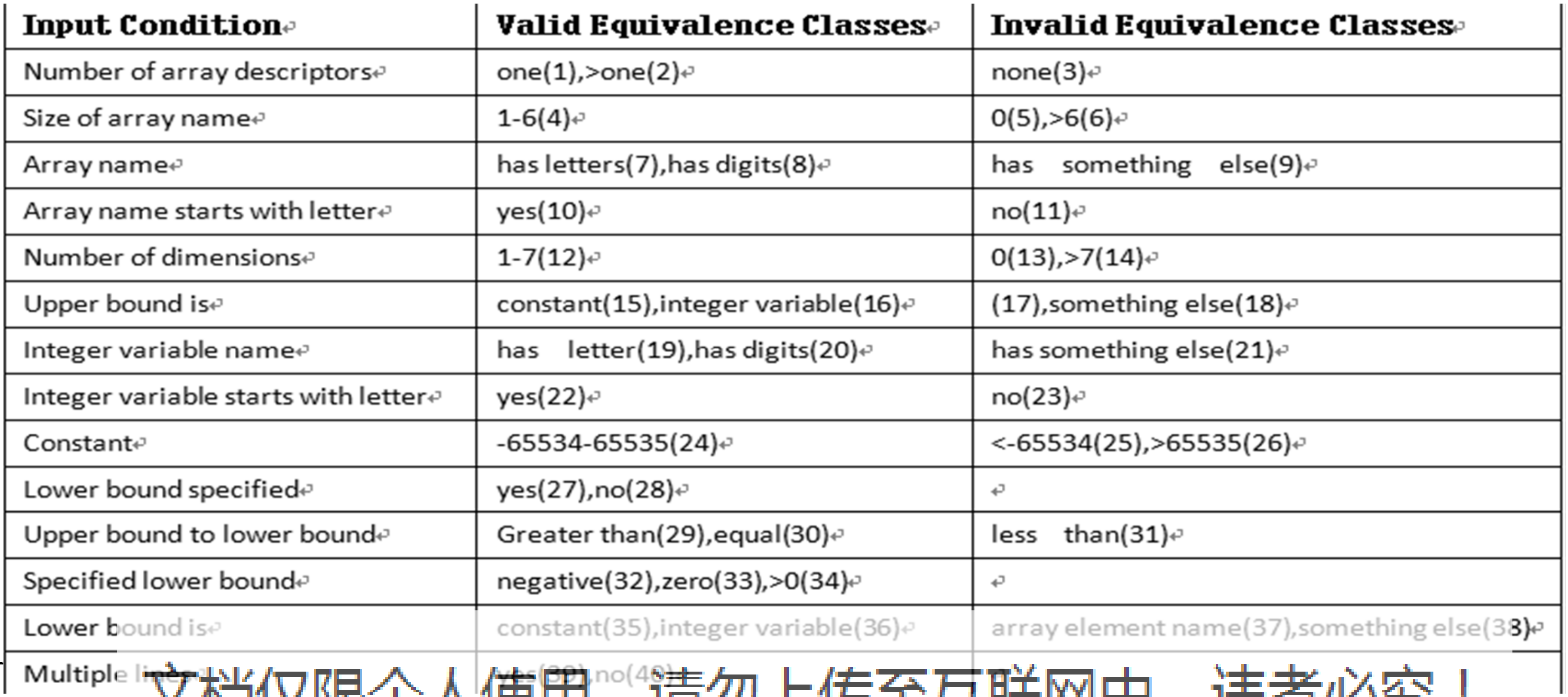
- **Equivalence Classes**



Input Condition	**Valid Equivalence Classes**	**Invalid Equivalence Classes**
Number of array descriptors	one(1),>one(2)	none(3)
Size of array name	1-6(4)	0(5),>6(6)
Array name	has letters(7),has digits(8)	has something else(9)
Array name starts with letter	yes(10)	no(11)
Number of dimensions	1-7(12)	0(13),>7(14)
Upper bound is	constant(15),integer variable(16)	(17),something else(18)
Integer variable name	has letter(19),has digits(20)	has something else(21)
Integer variable starts with letter	yes(22)	no(23)
Constant	-65534-65535(24)	<-65534(25),>65535(26)
Lower bound specified	yes(27),no(28)	
Upper bound to lower bound	Greater than(29),equal(30)	less than(31)
Specified lower bound	negative(32),zero(33),>0(34)	
Lower bound is	constant(35),integer variable(36)	array element name(37),something else(38)
Multiple lines	yes(39),no(40)	

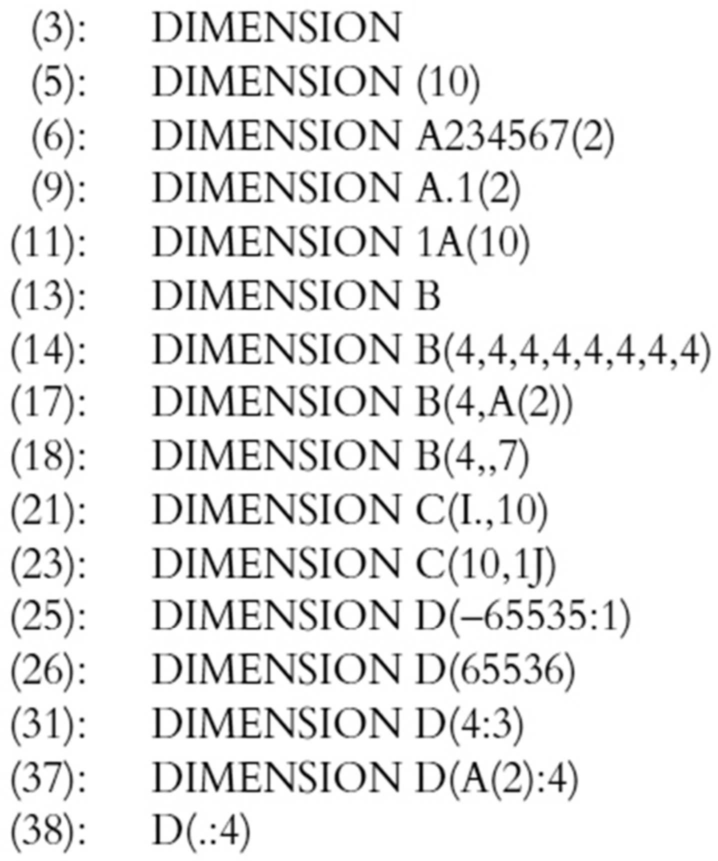
Exercise 7.5: FORTRAN Dimension

- test case
  - DIMENSION A(2)
    - covers classes 1, 4, 7, 10, 12, 15, 24, 28, 29, and 40.
  - DIMENSION A 12345 (I,9, J4XXXX,65535,1,KLM,X), BBB(−65534:100,0:1000,10:10, I:65535)
    - covers the remaining classes.

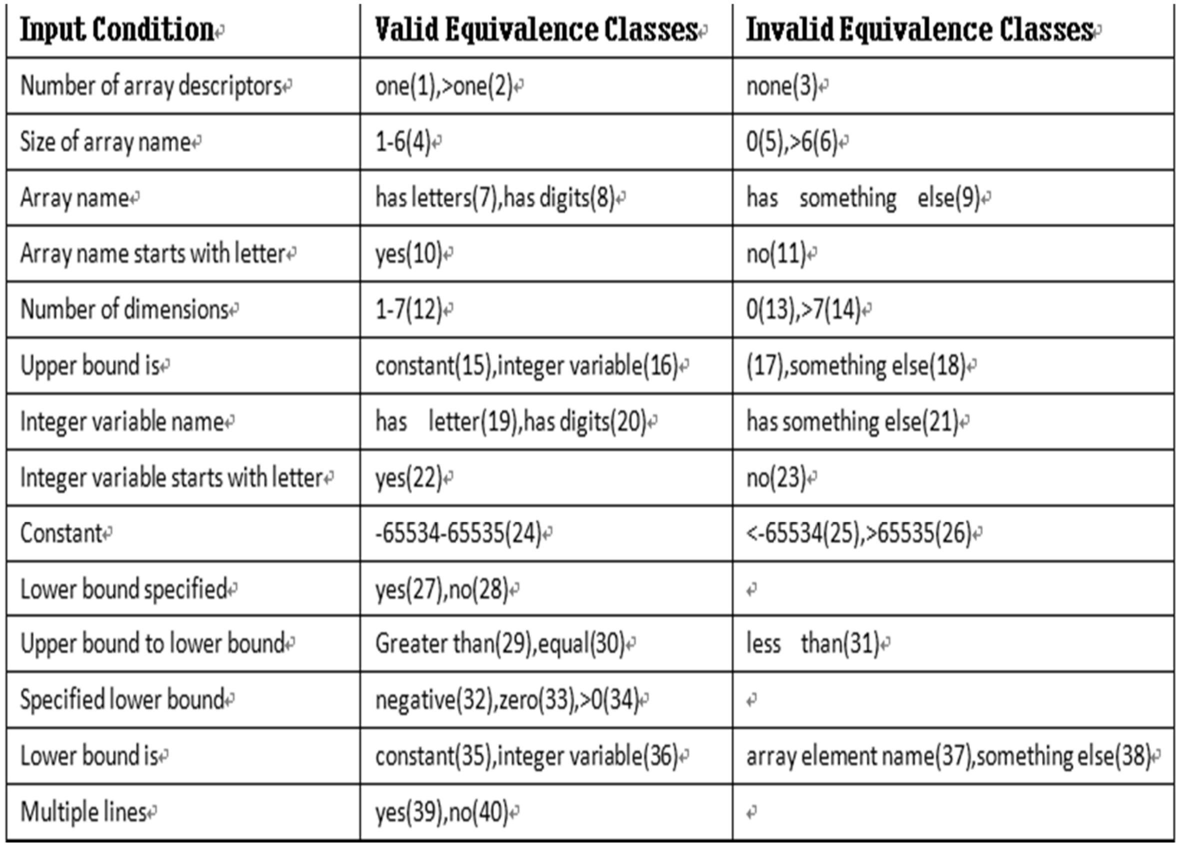


Input Condition	**Valid Equivalence Classes**	**Invalid Equivalence Classes**
Number of array descriptors	one(1),>one(2)	none(3)
Size of array name	1-6(4)	0(5),>6(6)
Array name	has letters(7),has digits(8)	has something else(9)
Array name starts with letter	yes(10)	no(11)
Number of dimensions	1-7(12)	0(13),>7(14)
Upper bound is	constant(15),integer variable(16)	(17),something else(18)
Integer variable name	has letter(19),has digits(20)	has something else(21)
Integer variable starts with letter	yes(22)	no(23)
Constant	-65534-65535(24)	<-65534(25),>65535(26)
Lower bound specified	yes(27),no(28)	
Upper bound to lower bound	Greater than(29),equal(30)	less than(31)
Specified lower bound	negative(32),zero(33),>0(34)	
Lower bound is	constant(35),integer variable(36)	array element name(37),something else(38)
Multiple lines	yes(39),no(40)	

Exercise 7.5: FORTRAN Dimension



(3):	DIMENSION
(5):	DIMENSION (10)
(6):	DIMENSION A234567(2)
(9):	DIMENSION A.1(2)
(11):	DIMENSION 1A(10)
(13):	DIMENSION B
(14):	DIMENSION B(4,4,4,4,4,4,4,4)
(17):	DIMENSION B(4,A(2))
(18):	DIMENSION B(4,,7)
(21):	DIMENSION C(I.,10)
(23):	DIMENSION C(10,1J)
(25):	DIMENSION D(–65535:1)
(26):	DIMENSION D(65536)
(31):	DIMENSION D(4:3)
(37):	DIMENSION D(A(2):4)
(38):	D(.:4)



Input Condition	Valid Equivalence Classes	Invalid Equivalence Classes
Number of array descriptors	one(1),>one(2)	none(3)
Size of array name	1-6(4)	0(5),>6(6)
Array name	has letters(7),has digits(8)	has something else(9)
Array name starts with letter	yes(10)	no(11)
Number of dimensions	1-7(12)	0(13),>7(14)
Upper bound is	constant(15),integer variable(16)	(17),something else(18)
Integer variable name	has letter(19),has digits(20)	has something else(21)
Integer variable starts with letter	yes(22)	no(23)
Constant	-65534-65535(24)	<-65534(25),>65535(26)
Lower bound specified	yes(27),no(28)	
Upper bound to lower bound	Greater than(29),equal(30)	less than(31)
Specified lower bound	negative(32),zero(33),>0(34)	
Lower bound is	constant(35),integer variable(36)	array element name(37),something else(38)
Multiple lines	yes(39),no(40)	

Module 7 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - **Specification-based testing**
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

At a Glance: Specification-Based Testing

Tag line	Verify every claim
Objective	Check conformance with every statement in every spec, requirements document, etc.
Testers	Any
Coverage	Documented reqts, features, etc.
Potential problems	Mismatch of implementation to spec
Activities	Write & execute tests based on the spec’s. Review and manage docs & traceability
Evaluation	Does behavior match the spec?
Complexity	Depends on the spec
Harshness	Depends on the spec
SUT readiness	As soon as modules are available

Strengths & Weaknesses: Spec-Based Testing

- Representative cases
  - Traceability matrix, tracks test cases associated with each specification item.
  - User documentation testing
- Strengths
  - Critical defense against warranty claims, fraud charges, loss of credibility with customers.
  - Effective for managing scope / expectations of regulatory-driven testing
  - Reduces support costs / customer complaints by ensuring that no false or misleading representations are made to customers.
- Blind spots
  - Any issues not in the specs or treated badly in the specs /documentation.

Traceability Tool for Specification-Based Testing

The Traceability Matrix

	Stmt 1	Stmt 2	Stmt 3	Stmt 4	Stmt 5
Test 1	X	X	X		
Test 2		X		X	
Test 3	X		X	X	
Test 4			X	X	
Test 5				X	X
Test 6	X				X

Optional Exercise 7.6: What “Specs” Can You Use?

- Challenge:
  - Getting information in the absence of a spec
  - What substitutes are available?
- Example:
  - The user manual – think of this as a commercial warranty for what your product does.
- What other “specs” can you/should you be using to test?

Exercise 7.6—Specification-Based Testing

- Here are some ideas for sources that you can consult when specifications are incomplete or incorrect.
  - Software change memos that come with new builds of the program
  - User manual draft (and previous version’s manual)
  - Product literature
  - Published style guide and UI standards

Exercise 7.6—Specification-Based Testing(cont.)

- Published standards (such as C-language)
- 3rd party product compatibility test suites
- Published regulations
- Internal memos (e.g. project mgr. to engineers, describing the feature definitions)
- Marketing presentations, selling the concept of the product to management
- Bug reports (responses to them)
- Reverse engineer the program.
- Interview people, such as
  - development lead, tech writer, customer service, subject matter experts, project manager
- Look at header files, source code, database table definitions
- Specs and bug lists for all 3rd party tools that you use

Exercise 7.6—Specification-Based Testing(cont.)

- Prototypes, and lab notes on the prototypes
- Interview development staff from the last version.
- Look at customer call records from the previous version. What bugs were found in the field?
- Usability test results
- Beta test results
- Localization guide (probably one that is published, for localizing products on your platform.)
- Get lists of compatible equipment and environments from Marketing (in theory, at least.)
- ……

Module 7 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - **Risk-based testing**
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

Definitions—Risk-Based Testing

- Three key meanings:
  1. **Find errors** (risk-based approach to the technical tasks of testing)
  2. **Manage the process of finding errors** (risk-based test management)
  3. **Manage the testing project and the risk posed by (and to) testing in its relationship to the overall project** (risk-based project management)
- We'll look primarily at risk-based testing (#1), proceeding later to risk-based test management.
- The project management risks are very important, but out of scope for this class.

At a Glance: Risk-Based Testing

Tag line	Find big bugs first
Objective	Define, prioritize, refine tests in terms of the relative risk of issues we could test for
Testers	Any
Coverage	By identified risk
Potential problems	Identifiable risks
Activities	Use qualities of service, risk heuristics and bug patterns to identify risks
Evaluation	Varies
Complexity	Any
Harshness	Harsh
SUT readiness	Any stage

Strengths & Weaknesses: Risk-Based Testing

- Representative cases
  - Equivalence class analysis, reformulated.
  - Test in order of frequency of use.
  - Stress tests, error handling tests, security tests.
  - Sample from predicted-bugs list.
- Strengths
  - Optimal prioritization (if we get the risk list right)
  - High power tests
- Blind spots
  - Risks not identified or that are surprisingly more likely.
  - Some “risk-driven” testers seem to operate subjectively.
    - How will I know what coverage I’ve reached?
    - Do I know that I haven’t missed something critical?

Workbook Page—Risks in Qualities of Service

- **Quality Categories:**
  - Accessibility
  - Capability
  - Compatibility
  - Concurrency
  - Efficiency
  - Localizability
  - Maintainability
  - Performance
  - Portability
  - Recoverability
  - Installability and uninstallability -- Conformance to standards
  - Reliability -- Scalability -- Security
  - Supportability -- Testability -- Usability

Each quality category is a risk category, as in:
“the risk of unreliability.”

Workbook Page—Heuristics to Find Risks (1/2)

- **Risk Heuristics: Where to look for errors**
  - **New things**: newer features may fail.
  - **New technology:** new concepts lead to new mistakes.
  - **Learning Curve:** mistakes due to ignorance.
  - **Changed things**: changes may break old code.
  - **Late change:** rushed decisions, rushed or demoralized staff lead to mistakes.
  - **Rushed work:** some tasks or projects are chronically underfunded and all aspects of work quality suffer.

Workbook Page—Heuristics to Find Risks (2/2)

- Risk Heuristics: Where to look for errors
  - *Complexity*: complex code may be buggy.
  - *Bugginess*: features with many known bugs may also have many unknown bugs.
  - *Dependencies*: failures may trigger other failures.
  - *Untestability*: risk of slow, inefficient testing.
  - *Little unit testing*: programmers find and fix most of their own bugs. Shortcutting here is a risk.
  - *Little system testing so far*: untested software may fail.
  - *Previous reliance on narrow testing strategies*: (e.g. regression, function tests), can yield a backlog of errors surviving across versions.

Workbook Page—Bug Patterns As a Source of Risks

- *Testing Computer Software* laid out a set of 480 common defects. To use these:
  - Find a defect in the list
  - Ask whether the software under test could have this defect
  - If it is theoretically possible that the program could have the defect, ask how you could find the bug if it was there.
  - Ask how plausible it is that this bug could be in the program and how serious the failure would be if it was there.
  - If appropriate, design a test or series of tests for bugs of this type.
- Use the web: www.bugnet.com

Workbook Page—Risk-Based Test Management

- Project risk management involves
  - Identification of the different risks to the project (issues that might cause the project to fail or to fall behind schedule that cost too much or dissatisfy customers or other stakeholders)
  - Analysis of the potential costs associated with each risk
  - Development of plans and actions to reduce the likelihood of the risk or the magnitude of the harm
  - Continuous assessment or monitoring of the risks (or the actions taken to manage them)
- Useful material free at http://seir.sei.cmu.edu
- http://www.coyotevalley.com (Brian Lawrence)
- Good paper by Stale Amland, Risk Based Testing and Metrics, in appendix.

Optional Exercise 7.7: Risk-Based Testing

- You are testing Amazon.com

  (Or pick another familiar application)
- First brainstorm:
  - What are the functional areas of the app?
- Then evaluate risks:
  - What are some of the ways that each of these could fail?
  - How likely do you think they are to fail? Why?
  - How serious would each of the failure types be?

Module 7 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - **Stress testing**
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

At a Glance: Stress Testing

Tag line	Overwhelm the product
Objective	Learn what failure at extremes tells about changes needed in the program’s handling of normal cases
Testers	Specialists
Coverage	Limited
Potential problems	Error handling weaknesses
Activities	Specialized
Evaluation	Varies
Complexity	Varies
Harshness	Extreme
SUT readiness	Late stage

Strengths & Weaknesses: Stress Testing

- Representative cases
  - Buffer overflow bugs
  - High volumes of data, device connections, long transaction chains
  - Low memory conditions, device failures, viruses, other crises
  - Extreme load
- Strengths
  - Expose weaknesses that will arise in the field.
  - Expose security risks.
- Blind spots
  - Weaknesses that are not made more visible by stress.

Module 7 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - **Regression testing**
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

At a Glance: Regression Testing

Tag line	Automated testing after changes
Objective	Detect unforeseen consequences of change
Testers	Varies
Coverage	Varies
Potential problems	Side effects of changes Unsuccessful bug fixes
Activities	Create automated test suites and run against every (major) build
Complexity	Varies
Evaluation	Varies
Harshness	Varies
SUT readiness	For unit – early; for GUI - late

Strengths & Weaknesses—Regression Testing

- Representative cases
  - Bug regression, old fix regression, general functional regression
  - Automated GUI regression test suites
- Strengths
  - Cheap to execute
  - Configuration testing
  - Regulator friendly
- Blind spots
  - “Immunization curve”
  - Anything not covered in the regression suite
  - Cost of maintaining the regression suite

Module 7 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - **Exploratory testing**
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

At a Glance: Exploratory Testing

Tag line	Simultaneous learning, planning, and testing
Objective	Simultaneously learn about the product and about the test strategies to reveal the product and its defects
Testers	Explorers
Coverage	Hard to assess
Potential problems	Everything unforeseen by planned testing techniques
Activities	Learn, plan, and test at the same time
Evaluation	Varies
Complexity	Varies
Harshness	Varies
SUT readiness	Medium to late: use cases must work

Strengths & Weaknesses: Exploratory Testing

- Representative cases
  - Skilled exploratory testing of the full product
  - Rapid testing & emergency testing (including thrown-over-the-wall test-it-today)
  - Troubleshooting / follow-up testing of defects.
- Strengths
  - Customer-focused, risk-focused
  - Responsive to changing circumstances
  - Finds bugs that are otherwise missed
- Blind spots
  - The less we know, the more we risk missing.
  - Limited by each tester's weaknesses (can mitigate this with careful management)
  - This is skilled work, juniors aren't very good at it.

Module 7 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - **User testing**
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

At a Glance: User Testing

Tag line	Strive for realism Let’s try real humans (for a change)
Objective	Identify failures in the overall human/machine/software system.
Testers	Users
Coverage	Very hard to measure
Potential problems	Items that will be missed by anyone other than an actual user
Activities	Directed by user
Evaluation	User’s assessment, with guidance
Complexity	Varies
Harshness	Limited
SUT readiness	Late; has to be fully operable

Strengths & Weaknesses—User Testing

- Representative cases
  - Beta testing
  - In-house lab using a stratified sample of target market
  - Usability testing
- Strengths
  - Expose design issues
  - Find areas with high error rates
  - Can be monitored with flight recorders
  - Can use in-house tests focus on controversial areas
- Blind spots
  - Coverage not assured
  - Weak test cases
  - Beta test technical results are mixed
  - Must distinguish marketing betas from technical betas

User (Usability) Testing

- Some considerations:
  - 1. Has each user interface been tailored to the intelligence, educational background, and environmental pressures of the end user?
  - 2. Are the outputs of the program meaningful, non-insulting to the user, and devoid of computer gibberish?
  - 3. Are the error diagnostics, such as error messages, straightforward, or does the user need a PhD in computer science to comprehend them?

User (Usability) Testing

- Some considerations:
  - 4. Does the total set of user interfaces exhibit considerable conceptual integrity, an underlying consistency, and uniformity of syntax, conventions, semantics, format, style, and abbreviations?
  - 5. Where accuracy is vital, such as in an online banking system, is sufficient redundancy present in the input?
  - 6. Does the system contain an excessive number of options, or options that are unlikely to be used?

User (Usability) Testing

- Some considerations:
  - 7. Does the system return some type of immediate acknowledgment to all inputs?
  - 8. Is the program easy to use?
  - 9. Is the design conducive to user accuracy?
  - 10. Are the user actions easily repeated in later sessions?
  - 11. Did the user feel confident while navigating the various paths or menu choices?
  - 12. Did the software live up to its design promise?

User (Usability) Testing

- Test User Selection
  - software targeted for a **specific end-user type** or industry should be tested by what could be described as expert users.
  - software with a more general target market—mobile device software, for example, or general-purpose Web pages—might better be tested by users selected randomly.

User (Usability) Testing

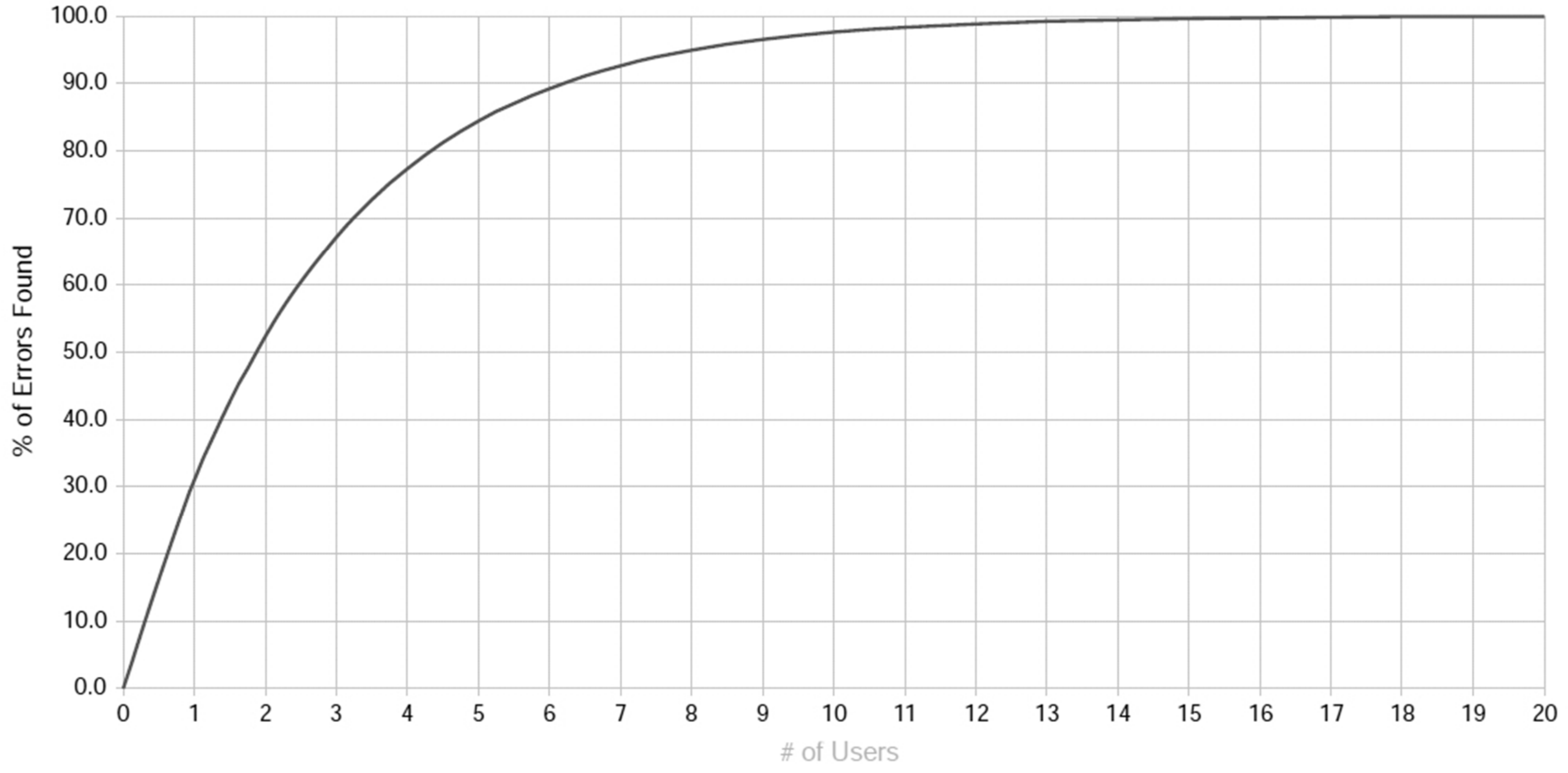
- How Many Users Do You Need?

$$E = 100 \times (1 - (1 - L)^{\wedge} n)$$

where: E = percent of errors found
n = number of testers
L = percent of usability problems found by a tester

User (Usability) Testing

- How Many Users Do You Need?
- Using the equation with L =31%



Module 7 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - **Scenario testing**
  - Stochastic or Random testing
- Using techniques together

At a Glance: Scenario Testing

Tag line	Instantiation of a use case Do something useful, interesting, and complex
Objective	Challenging cases to reflect real use
Testers	Any
Coverage	Whatever stories touch
Potential problems	Complex interactions that happen in real use by experienced users
Activities	Interview stakeholders & write screenplays, then implement tests
Evaluation	Any
Complexity	High
Harshness	Varies
SUT readiness	Late. Requires stable, integrated functionality.

Strengths & Weaknesses: Scenario Testing

- Representative cases
  - Use cases, or sequences involving combinations of use cases.
  - Appraise product against business rules, customer data, competitors’ output
  - Hans Buwalda’s “soap opera testing.”
- Strengths
  - Complex, realistic events. Can handle (help with) situations that are too complex to model.
  - Exposes failures that occur (develop) over time
- Blind spots
  - Single function failures can make this test inefficient.
  - Must think carefully to achieve good coverage.

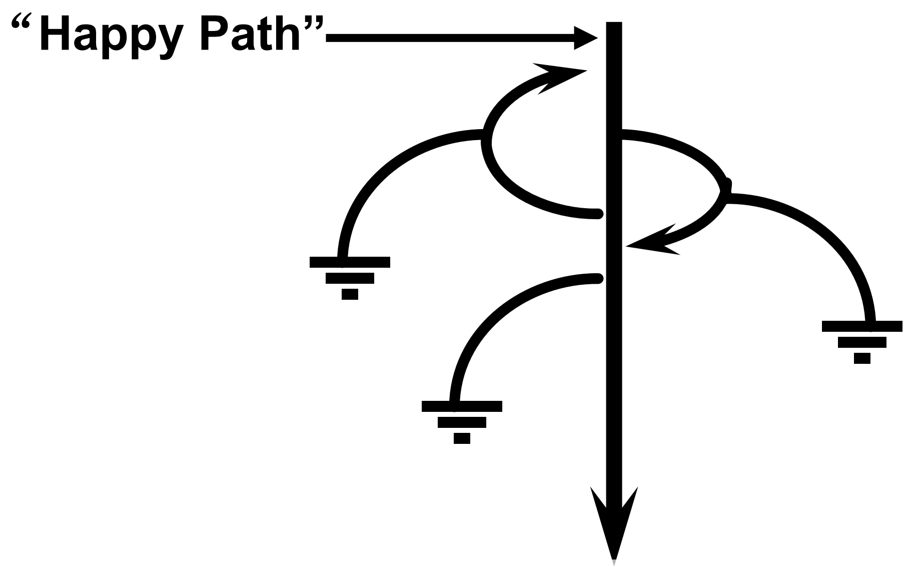
Workbook Page—Test Scenarios From Use Cases

Outline of a Use Case

Has one normal, *basic flow* (“Happy Path”)

Several *alternative flows*

- Regular variants
- Odd cases
- **Exceptional flows** handling error situations



Workbook Page—Scenarios Without Use Cases

- Sometimes, we develop test scenarios independently of use cases. The ideal scenario has four characteristics:
  - It is realistic (e.g. it comes from actual customer or competitor situations).
  - It is easy (and fast) to determine whether a test passed or failed.
  - The test is complex. That is, it uses several features and functions.
  - There is a stakeholder who has influence and will protest if the program doesn’t pass this scenario.

Workbook Page—Soap Operas

- A Soap Opera is a scenario based on real-life client/customer experience.
- Exaggerate every aspect of it. For example:
  - For each variable, substitute a more extreme value
  - If a scenario can include a repeating element, repeat it lots of times
  - Make the environment less hospitable to the case (increase or decrease memory, printer resolution, video resolution, etc.)
- Create a real-life story that combines all of the elements into a test case narrative.

Optional Exercise 7.8: Soap Operas for Testing

1. Pick a familiar product
2. Define a scope of the test
3. Identify with the business environment
4. Include elements that would make things difficult
5. Tell the story

Module 7 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - **Stochastic or Random testing**
- Using techniques together

At a Glance: Stochastic or Random Testing (1/2)

Tag line	Monkey testing High-volume testing with new cases all the time
Objective	Have the computer create, execute, and evaluate huge numbers of tests. The individual tests are not all that powerful, nor all that compelling. The power of the approach lies in the large number of tests. These broaden the sample, and they may test the program over a long period of time, giving us insight into longer term issues.

At a Glance: Stochastic or Random Testing (2/2)

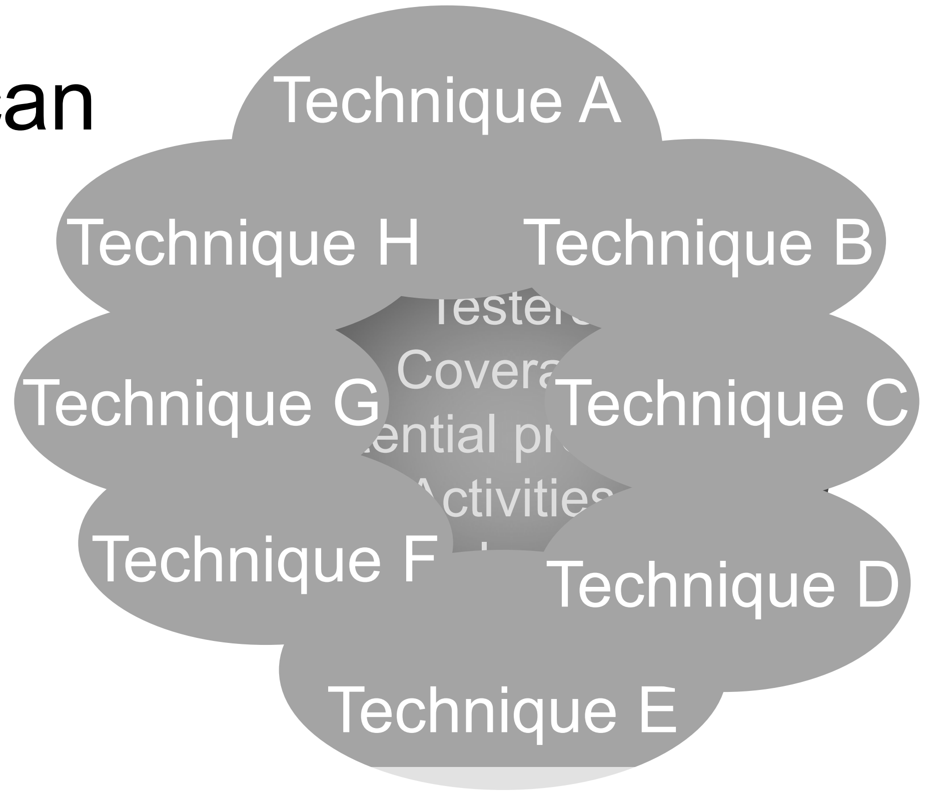
Testers	Machines
Coverage	Broad but shallow. Problems with stateful apps.
Potential problems	Crashes and exceptions
Activities	Focus on test generation
Evaluation	Generic, state-based
Complexity	Complex to generate, but individual tests are simple
Harshness	Weak individual tests, but huge numbers of them
SUT readiness	Any

Module 7 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- Individual techniques
- **Using techniques together**

Combining Techniques (Revisited)

- A test approach should be diversified
- Applying opposite techniques can improve coverage
- Often one technique can extend another



Applying Opposite Techniques to Boost Coverage

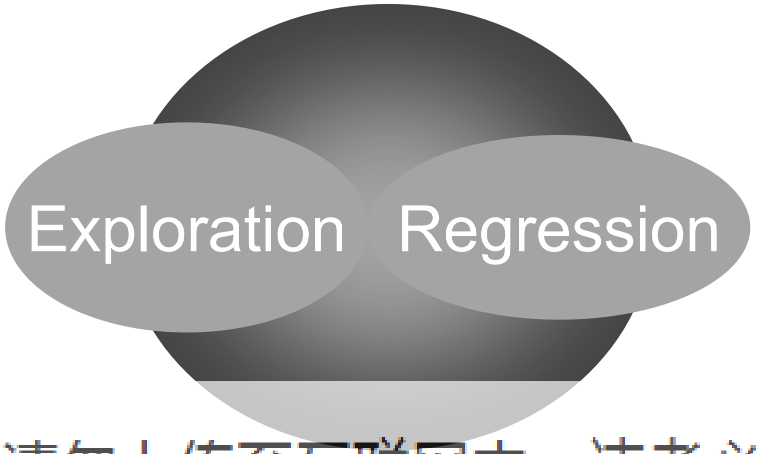
<u>Contrast these two techniques</u>

Regression

- Inputs:
  - Old test cases and analyses leading to new test cases
- Outputs:
  - Archival test cases, preferably well documented, and bug reports
- Better for:
  - Reuse across multi-version products

Exploration

- Inputs:
  - models or other analyses that yield new tests
- Outputs
  - scribbles and bug reports
- Better for:
  - Find new bugs, scout new areas, risks, or ideas



Applying Complementary Techniques Together

- Regression testing alone suffers fatigue
  - The bugs get fixed and new runs add little info
- Symptom of weak coverage
- Combine automation w/ suitable variance
  - E.g. Risk-based equivalence analysis
- Coverage of the combination can beat sum of the parts

Equivalence
Risk-based
Regression

How To Adopt New Techniques

1. Answer these questions:
   - What techniques do you use in your test approach now?
   - What is its greatest shortcoming?
   - What **one** technique could you add to make the greatest improvement, consistent with a good test approach:
     - Risk-focused?
     - Product-specific?
     - Practical?
     - Defensible?
2. Apply that additional technique until proficient
3. Iterate

Discussion 7.9: Which Techniques Should You Use

1. Break out into workgroups
2. For your team, answer the questions on the previous slide
3. Present your findings

Optional Review Exercise 7.10: Characterize Testing Techniques

	Testers	Coverage	Problems / Risks	Activities	Evaluation
Function testing					
Equivalence analysis					
Specification-based testing					
Risk-based testing					
Stress testing					
Regression testing					
Exploratory testing					
User testing					
Scenario testing					
Stochastic testing					