LECTURE 03-04.

COVERAGE-BASED TECHNIQUES. PART II

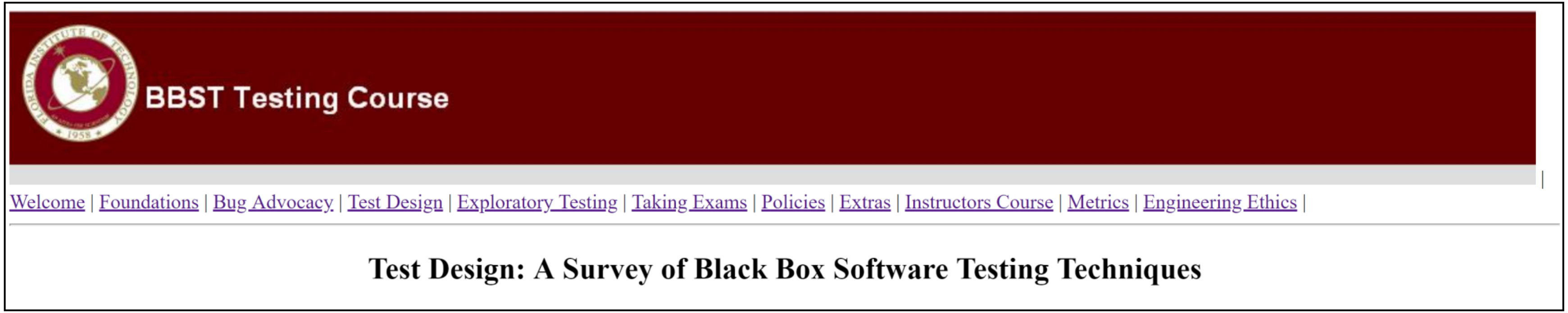
Test Design Techniques
[09-16 March 2022]

Elective Course, Spring Semester, 2021-2022

Camelia Chisăliţă-Creţu, Lecturer PhD
Babeş-Bolyai University

Acknowledgements

The course Test Design Techniques is based on the Test Design course available on the **BBST Testing Course** platform.



BBST Testing Course

Welcome | Foundations | Bug Advocacy | Test Design | Exploratory Testing | Taking Exams | Policies | Extras | Instructors Course | Metrics | Engineering Ethics |

Test Design: A Survey of Black Box Software Testing Techniques

The BBST Courses are created and developed by **Cem Kaner, J.D., Ph.D.., Professor of Software Engineering at Florida Institute of Technology.**

Contents

- **Coverage-based techniques**
  - **Part II**
    10. Multivariable testing
    11. Specification-based testing
    12. Configuration testing
    13. Logical expressions
    14. State Model-based testing
    15. User interface testing
    16. Requirements-based testing
    17. Compliance-driven testing
    18. Localization testing

COVERAGE TECHNIQUES. PART II

Multivariable testing, Specification-based testing, Configuration testing
Logical expressions
State Model-based Testing, User interface testing
Requirements-based testing, Compliance-driven testing, Localization testing

Multivariable Testing. Definition

- **Multivariable Testing allows to design and run tests for various *independent* variables.**
- Let ***a*** and ***b*** be two variables within a product; if variable **b** values are constraint to (depend on) the variables ***a*** values then ***a*** and ***b*** are **dependent variables**, otherwise they are called **independent variables**;
- E.g.:
  - in OpenOffice Writer, variables **Page Format** ∈ {A4, A5, Letter, …} and **Header Height** ∈ [0.00cm, PageFormatHeight-6.50cm] are *dependent variables*;
  - in OpenOffice Writer, variables **Page Format** ∈ {A4, A5, Letter, …} and **Number of pages** ≥ 1 are *independent variables*;
- there are several types (approaches) of multivariable testing: *mechanical*, *risk-based*, *scenario-based*.

Coverage: Multivariable testing ensures coverage to the extent that a set of tests is designed and attempt to cover it.

Multivariable Testing. Details

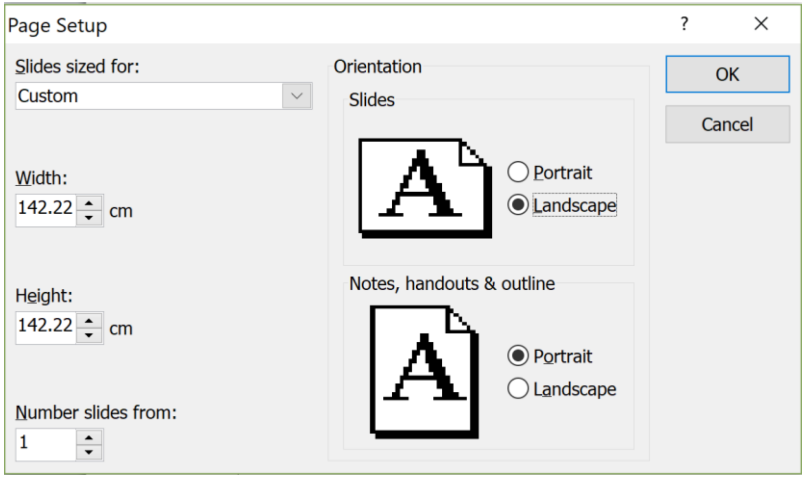
- there are several types (approaches) of multivariable testing
  - **mechanical:** the tester uses a routine **procedure** to determine a good set of tests;
    - E.g.: **random combinations** and **all-pairs**.
  - **risk-based:** the tester combines test values (values of each variable) based on **perceived risks** associated with noteworthy combinations (see **Lecture 5**);
  - **scenario-based:** the tester combines test values on the basis of **interesting stories** created for the combinations (see **Lecture 6**).

Multivariable Testing. Mechanical Approach

- **multivariable testing = combination testing**
- Combination testing consists of several coverage criteria:
  - **all singles;**
  - **all pairs;**
  - **all triples;**
  - **all N-tuples.**
- **All pairs testing** is the best-known combination testing technique;
  - it is effective for testing many **independent** variables.

Multivariable Testing. Example

- Combination testing for three *independent variables*:
  - software product **Microsoft PowerPoint**, **Page Setup** dialog:
    - **Page Width** ∈ [1.00cm, 142.22cm];
    - **Page Height** ∈ [1.00cm, 142.22cm];
    - **Number slides from…** ∈ [0, 9999].



Slides sized for: Custom

Width: 142.22 cm

Height: 142.22 cm

Number slides from: 1

Orientation

Slides
○ Portrait
● Landscape

Notes, handouts & outline
● Portrait
○ Landscape

Multivariable Testing. All Singles Coverage

- **All Singles criterion is met when each value of each variable is in at least one test.**

	Width	Height	Slide #
Test 01	1.00	1.00	0
Test 02	142.22	142.22	9999

Multivariable Testing. All Pair Coverage

- **All Pairs criterion is met when each pair of values of each pair of variables is in at least one test.**

	Width	**Height**	**Slide #**
Test 01	1.00	1.00	0
Test 02	1.00	142.22	9999
Test 03	142.22	1.00	9999
Test 04	142.22	142.22	0

Multivariable Testing. All Triples Coverage

- **All Triples criterion is met when each 3-tuple of values of each group of 3 variables is in at least one test.**

	Width	Height	Page #
Test 01	1.00	1.00	0
Test 02	1.00	1.00	9999
Test 03	1.00	142.22	0
Test 04	1.00	142.22	9999
Test 05	142.22	1.00	0
Test 06	142.22	1.00	9999
Test 07	142.22	142.22	0
Test 08	142.22	142.22	9999

Multivariable Testing. All N-Tuples Coverage

- **All N-Tuples criterion is met when every possible combination of the variables' values is in at least one test.**

	Width	Height	Page #
Test 01	1.00	1.00	0
Test 02	1.00	1.00	9999
Test 03	1.00	142.22	0
Test 04	1.00	142.22	9999
Test 05	142.22	1.00	0
Test 06	142.22	1.00	9999
Test 07	142.22	142.22	0
Test 08	142.22	142.22	9999

- For the case with 3 variables **All Triples = All N-Tuples.**

Specifications. Definition

- A specification includes any document that:
  - describes the product, and
  - drives development, sale, support, use, or purchase of the product, and
  - either
    - was created by the maker or other vendor of the product *or*
    - would be accepted by the maker or other vendor of the product as an accurate or controlling description.

Specifications. Details

- **the scope of the specification** may be:
  - to cover the *entire* product or to describe *only some part* of it, e.g., error handling;
  - to address the product from *multiple points of view* or only from *a single point of view*.

- **Questions to ask:**
  - *Do we have the right specification?*
  - *Do we have the current version?*
  - *Is the spec kept under source control?*
  - *How do we verify the version?*
  - *Is this a stable specification?*
  - *Is the product under change control?*
  - *Is the spec under change control?*

Specifications. Types

- Types of specifications:
  - **implicit specifications:** e.g., many programs do arithmetic but few include explicit specifications of the rules of arithmetic;
  - **explicit specifications: any document** that contains factual claims.

Anything that drives *people's expectations* of the product is an specification (explicit or implicit).

Specifications. Implicit Specifications

- **An implicit specification** is
  - some aspects of the product are *clearly understood*, but not described in detail in the formal specifications because:
    - they are determined by controlling **cultural or technical norms** and often described in documents completely independent of this product;
    - they are **defined among the staff**, perhaps in some other document;

- **finding documents that describe these implicit specifications is useful**:
  - rather than making an unsupported statement like *this is inappropriate* or *users won't like it*, the tester can use implicit specifications to justify his assertions.

Specifications. Implicit Specifications Examples (1)

- Examples of implicit specifications:
  - **Published style guide and UI standards;**
  - Published standards (such as C-language or IEEE Floating Point);
  - **Third party product compatibility test suites;**
  - Localization guide (probably published for localizing products on the developer's platform);
  - **Published regulations;**
  - Marketing presentations, e.g., documents that sell the concept of the product to management;
  - **Internal memos**, e.g.: project manager to engineers to describe feature definitions;
  - User manual draft (and previous version's manual);
  - **Product literature (advertisements and other promotional documents);**
  - **Sales presentations;**
  - Software change memos that come with each new internal version of the program.

Specifications. Implicit Specifications Examples (2)

- Examples of implicit specifications:
  - **Bug reports** and responses to them;
  - Look at customer call records from the previous version. What bugs were found in the field?
  - **Usability test results** and corporate responses to them;
  - **Beta test results** and corporate responses to them;
  - Third party tech support databases, magazines and web sites with:
    - **discussions of bugs in tested product**,
    - **common bugs in the current type of product niche** or on the issuing platform,
    - discussions of how some features are supposed (by some people) to work.

Specifications. Implicit Specifications Examples (3)

- Examples of implicit specifications:
  - Reverse engineer the program;
  - Look at header files, source code, database table definitions;
  - **Prototypes and lab notes** on the prototypes;
  - **Interview people**, such as
    - development lead;
    - tech writer;
    - customer service;
    - subject matter experts;
    - project manager;
    - development staff from the last version.

Specifications. Implicit Specifications Examples (4)

- Examples of implicit specifications:
  - **Specs and bug lists for all third party tools** that are used for the current product:
    - E.g.: If a company develops software for the Windows platform, the Microsoft Developer Network has lots of relevant information;
  - **The lists of compatible equipment and environments**:
    - interface specifications;
    - protocol specifications.
  - **Reference materials** that can be used as oracles for the content that comes with the program:
    - E.g.: use an atlas to check on-line geography program.

Specifications. Implicit Specifications Examples (5)

- Examples of implicit specifications:
  - Look at competing products:
    - **Similarities and differences between the benefits and features offered by the products;**
    - How the other products describe their design, capabilities and behaviors?
    - **What weaknesses to they have, what bugs do they have or were publicly fixed?**
  - Make precise comparisons with products can be emulated. If product X is supposed to work “just like” Y, X should be compared with Y thoroughly.

Specification-based Testing. Definition

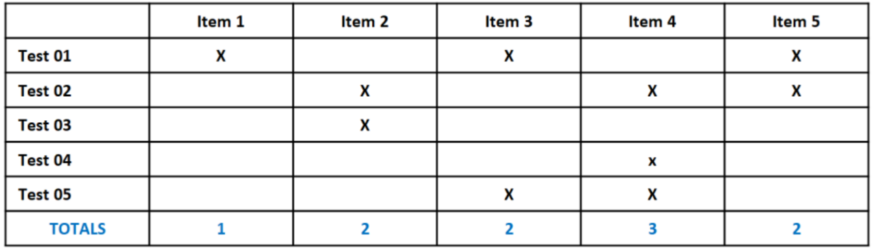
- **Spec-based Testing is focused on verifying factual claims made about the product in the specification.**
- **A factual claim** is any statement that can be shown to be true or false:
  - this often includes claim made in the *manual*, in *marketing documents* or *advertisements*, and in *technical support literature* sent to customers.

Specification-based Testing. Deriving Tests from Specs

- For every statement or fact found in the specification the tester should create:
  - at least one test that tries to check **the program behaviour if the statement is false**;
  - several tests that **vary the parameters of the statement**, e.g., test boundary conditions;
  - several tests of the **reasonable implications of the statement**;
  - several tests of this statement in **conjunction with related statements**;
  - several tests of **scenarios** that apply the statement in the process of achieving a program benefit.

How many tests should be created? **The level of depth chosen at test design should depend on the kinds of information looked for and the risks the tester tries to manage.**

Specification-based Testing. Traceability Matrix



	Item 1	Item 2	Item 3	Item 4	Item 5
Test 01	X		X		X
Test 02		X		X	X
Test 03		X			
Test 04				x	
Test 05			X	X	
TOTALS	1	2	2	3	2

- **A traceability matrix** is
  - a common tool for tracking spec-based tests;
  - useful for tracking specification coverage;
- Elements:
  - test items columns - each test item in its own column;
    - a **test item** is anything that must be tested: might be *a function, a variable, an assertion in a specification, a device that must be tested*.
  - test case rows – each test case in its own row; a cell shows that this test tests a specific test item.
- **A traceability matrix may be applied to trace tests to anything.**
- when a feature changes, the tester can quickly see which tests must be reanalyzed, probably rewritten; in general, the tester can trace back from a given item of interest to the tests that cover it;

The traceability map does not specify the tests, it just maps their coverage, i.e., how many times the tests touched the item, not how relevant the test is (depth of testing).

Specification-based Testing. Traceability Matrix Example

- E.g.: Let **App** be a software product with 5 relevant spec items;
  - the table below consists of 5 spec items;
  - the tester has designed 5 tests;
  - each of them covers/addresses one or more spec items, marked with x;
  - last row indicates how many items are run against each spec item.

	Item 1	Item 2	Item 3	Item 4	Item 5
Test 01	X		X		X
Test 02		X		X	X
Test 03		X			
Test 04				x	
Test 05			X	X	
TOTALS	1	2	2	3	2

A traceability matrix maps tests to test items. For each test item, the tester can trace back to the tests that test it.

Configuration Testing. Definition

- **Configuration Testing is**
  - **the application of multiple combinations of software and hardware to find out the optimal configurations that the system can work without any flaws or bugs.**

- **the goal is to test SUT compatibility with different devices;**
- it uses **all-singles** and **all-pairs** testing.

Coverage: Configuration testing ensures that every? configuration for the product is tested.

Configuration Testing. Coverage

- **Configuration coverage is the percentage of configuration tests the program has passed compared to the number planned to run.**

$$\text{Configuration Coverage} = \frac{\text{number of configuration tests passed}}{\text{total number of planned tests}} \times 100$$

- E.g.: if the tester needs to test the compatibility with 100 printers, and he has tested with 10 of them, then he has achieved 10% printer coverage.

- *Is this a test technique by itself?*
  - testers that are focused on this coverage objective are likely to develop methods to make high volume configuration testing faster and easier;
  - **The optimization of the effort to achieve high coverage is the underlying technique.**

Configuration Testing. Steps to Apply

- Setup steps for configuration testing:

1. Select the **variables** to test;
2. Select the **test values** for each variable;
   - the smallest reasonable set for each variable should be selected because these numbers are multiplied;
3. **Assign 1-character abbreviations** for each value of each variable, to make the chart simple
   - this is required if the chart is created by hand;
4. Decide on **a coverage criterion**;
5. **Create the combination chart.**

Configuration Testing. Example

- **An example of configuration testing:**
  - **OS:** WinVISTA, WinXP, Win10;
  - **Printer:** HP, Epson, Lexmark;
  - **Memory:** Low, Medium, High;
  - **Processor:** 1-core, 2-core, 4-core;
  - **Graphics:** Slow, Medium, Fast;
  - **Hard drive:** 0 drives, 1 drive, 2 drives.

- **Total number of possible tests – N-tuples:**
  - **$3 * 3 * 3 * 3 * 3 * 3 = 3^6 = 729$**

Configuration Testing. Example

- OS:
  1. WinVista
  2. WinXP
  3. Win10
- Printer:
  1. HP
  2. Epson
  3. Lexmark
- Memory:
  1. Low
  2. Medium
  3. High
- Processor:
  1. 1-core
  2. 2-core
  3. 4- core
- Graphics:
  1. Slow
  2. Medium
  3. Fast
- Hard drive:
  1. 0 drives
  2. 1 drive
  3. 2 drives

Configuration Testing. All-Single Criterion

- Designed tests:
  - every value (decided to be tested)
    - of each variable
    - at least once.

	OS	Printer	Memory	Processor	Graphics	Drives
Test 01	**1**	**1**	**1**	**1**	**1**	**1**
Test 02	**2**	**2**	**2**	**2**	**2**	**2**
Test 03	**3**	**3**	**3**	**3**	**3**	**3**

Configuration Testing. All- Pairs Criterion (1)

- one variable is added to the table at a time;
- the variables are sorted such that:
  - the variable with the most values is placed in the first column;
  - the second-most values is placed in the second column;
- E.g.: 4 types of printers and for other variables there are 3 values then we would add printers first.

Configuration Testing. All- Pairs Criterion (2)

	OS	Printer	Memory	Processor	Graphics	Drives
Test 01	1	1				
Test 02	1	2				
Test 03	1	3				
Test 04	2	1				
Test 05	2	2				
Test 06	2	3				
Test 07	3	1				
Test 08	3	2				
Test 09	3	3				

Configuration Testing. All- Pairs Criterion (3)

	OS	Printer	Memory	Processor	Graphics	Drives
Test 01	1	1	1			
Test 02	1	2	2			
Test 03	1	3	3			
Test 04	2	1	2			
Test 05	2	2	3			
Test 06	2	3	1			
Test 07	3	1	3			
Test 08	3	2	1			
Test 09	3	3	2			

Configuration Testing. All- Pairs Criterion (4)

	OS	Printer	Memory	Processor	Graphics	Drives
Test 01	1	1	1	1	1	1
Test 02	1	2	2	2	2	2
Test 03	1	3	3	3	3	3
Test 04	2	1	2	3	1	2
Test 05	2	2	3	1	3	1
Test 06	2	3	1	2	2	3
Test 07	3	1	3	2	2	3
Test 08	3	2	1	3	1	3
Test 09	3	3	2	1	3	2
Test 10		1	1	2	3	2
Test 11	3	3	3	2	1	1
Test 12		2	2	1	2	3
Test 13			2	3	2	1
Test 14			3			2

Logical Expression. Definition

- **Logical Expression Testing allows to design tests for existing *decision rules*.**
- **A decision rule expresses a logical relationship.**
- E.g.: a health-insurance product with the decision rule that states:
  - **`if PERSON-AGE > 50 and`**
    - **`if PERSON-SMOKES is TRUE`**
      - **`then set OFFER-INSURANCE to FALSE.`**
- A series of such separate decisions result in the same outcome as if the user had made all those decisions at the same time; the tester can test this decision set together as one complex logical expression;
- Testers often represent decision rules (and combinations of rules) in **decision tables** – **each row in the table represents a test.**

Coverage: Logical expression testing attempts to check every decision in the program or a theoretically interesting subset.

State Model-based Testing. Definition

- **State Model-based Testing allows to use specialized algorithms to walk the program through long paths that cover all sequences of length 2.**
- usually, the tester a program execution may be seen as moving from one state to another;
- in a given state, some inputs are valid and others are ignored or rejected; in response to a valid input, the program under test does something that it can do, which takes it to a new state;
- There are many sequences types:
  - sequence of length 2: state → transition → state;
  - sequence of length 3: state → transition → state → transition → state;
  - …
- other related technique: **Operational Modes tour**.

Coverage: State-model testing allows to run the program through <u>long paths</u> to cover all sequences of various lengths.

User Interface Testing. Definition

- **User Interface Testing checks if the *UI elements* have been implemented correctly.**
- User interface testing is **NOT** about:
  - whether the UI is *well designed* or
  - *easy to understand* or
  - *easy to work* with.
- All from above are aspects related to **usability testing.**

Coverage: user interface testing covers all the elements of the user interface (the dialogs, menus, pull-down lists, and all the other UI controls).

Requirements-based Testing. Definition

- **Requirements-based Testing focuses on testing *written* requirements.**
- requirements-based testing aims to prove, requirement by requirement, that:
  - the program satisfies every requirement in a requirements document, or that
  - some of the requirements have not been met.
- sometimes it might not be possible to say if some requirement is met by running simple tests; requirements that are easily testable are often trivial compared to the “real” requirements.
- **generally, requirements are incomplete, subject to frequent change, and often incorrect**.

Coverage: Requirements-based testing ensures written requirements are covered by tests.

Compliance-driven Testing. Definition

- **compliance** = **conformity with some regulations**;
- **Compliance-driven testing is focused on doing the set of tasks (usually the minimum set) needed to demonstrate compliance with these requirements.**
- Some products must meet externally-imposed requirements, e.g., such as regulatory requirements;

Coverage: Compliance-driven testing ensures that every task needed is done in order to demonstrate conformity/compliance.

Localization Testing. Definition

- **Localization** for software products means
  - the possibility **for a program to run in a different language, a different country, or a different culture; this means a specific set of changes are performed.**
- software publishers who will localize their software typically design the software to make localization easy;
- **Localization testing means to**
  - **create a list of the things that can be changed for localization;**
  - **test the list** to see what was actually changed, whether the changes worked, and whether anything that should not have been changed was not changed.

Coverage: Localization testing allows to test against a list of localization-related changes and risks.

Lecture Summary

- We have discussed:
  - Coverage-based techniques (Part II);
  - Coverage-based techniques attributes.

Lab Activities on week 03-04

- Tasks to achieve in week 01-02 during Lab 01:
  - **Play the game *"Testing Challenge #6 - Boundary testing"*** and **report the testing results;**
  - **Perform Tour Testing and Multivariable Testing by using an all-pair generator tool for a product at your choice.**

References

- **[Kaner2003]** Cem Kaner, An introduction to scenario testing, http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf, 2003.
- **[JonathanKohl2006]** Jonathan Kohl, *Modeling Test Heuristics*, http://www.kohl.ca/2006/modeling-test-heuristics/, 2006.
- **[DevelopSense2009]** Developer Sense, *Blog: Of Testing Tours and Dashboards,* http://www.developsense.com/blog/2009/04/of-testing-tours-and-dashboards/, 2009.
- **[Whittaker1997]** Whittaker, J.A. (1997). Stochastic software testing. Annals of Software Engineering, 4, pp. 115-131.
- **[BBST2011]** BBST – Test Design, Cem Kaner, http://www.testingeducation.org/BBST/testdesign/BBSTTestDesign2011pfinal.pdf.
- **[BBST2010]** BBST – Fundamentals of Testing, Cem Kaner, http://www.testingeducation.org/BBST/foundations/BBSTFoundationsNov2010.pdf.
- **[KanerBachPettichord2001]** Kaner, C., Bach, J., & Pettichord, B. (2001). Lessons Learned in Software Testing: Chapter 3: Test Techniques, http://media.techtarget.com/searchSoftwareQuality/downloads/Lessons_Learned_in_SW_testingCh3.pdf