LECTURE 01.
TESTING CONCEPTS

Test Design Techniques

Elective Course, Spring Semester, 2021-2022

[23 February 2022]

Camelia Chisăliţă-Creţu, Lecturer PhD

Babeş-Bolyai University

Contents

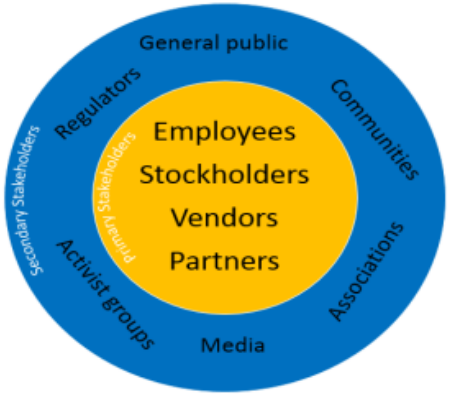
- Terminology
  - Stakeholders
    - Definition. Types
    - Exercise 1. Stakeholder Relationships
  - Computer program
    - Definition
  - Public Library Software
    - Exercise 2. Classify Stakeholders
  - Software Quality
    - Definitions
    - Exercise 3. Appropriate Software Quality
  - Software Testing
    - Definitions
- All together…
- Error, Fault, Bug, Defect, Failure
- Software error
- Test case
  - Attributes of good test cases

Stakeholders. Definition

- A stakeholder is any person affected by:
  - **the success** or **the failure** of a project, or
  - the actions or the inactions of a product, or
  - the effects of a service [BBST2010].

Stakeholders. Types

- Types of stakeholders:
  - **primary/secondary** [StakeholderMap2019]:
    - **primary stakeholder**;
    - **secondary stakeholder**;
  - **favoured/disfavoured** [GauseWeinberg2011], [KanerBach2005]:
    - **favoured stakeholder**;
    - **neutral stakeholder**;
    - **disfavoured stakeholder** ;
    - **ignored stakeholder** ;
  - **vested/ with influence** [KanerBack2005]:
    - **vested/investment-backed**;
    - **with influence**.



Exercise 1. Stakeholder Relationships

- Indicate valid relationships among the following types of stakeholders (⇒ , ⇐ , ⇏, ⇍, ⇔):
  - **favored stakeholder ? primary stakeholder;**
  - **disfavored stakeholder ? secondary stakeholder;**
  - **neutral stakeholder ? secondary stakeholder;**
  - **favored stakeholder ? neglected stakeholder;**
  - **secondary stakeholder ? disfavored stakeholder;**
  - **ignored stakeholder ? neglected stakeholder;**
  - **disfavored stakeholder ? primary stakeholder;**
  - **favored stakeholder ? vested stakeholder;**
  - **stakeholder with influence ? favored stakeholder**
  - **primary stakeholder ? vested stakeholder.**

Computer Program. Definition

- **A computer program** is
  - a **communication**
  - among several **humans** and **computers**
  - who are **distributed** over space and time,
  - that contains **instructions** that can be **executed** by a computer [BBST2010].

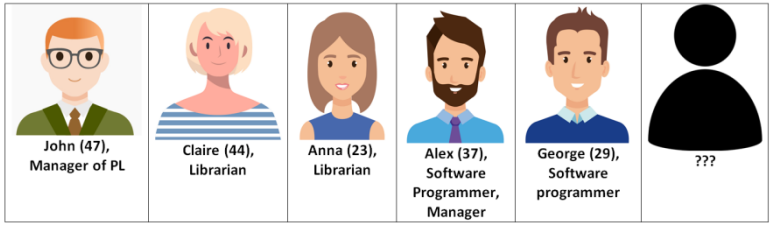
Public Library Software

- **A software for the Public Library**

 John (47), Manager of PL	 Claire (44), Librarian	 Anna (23), Librarian	 Alex (37), Software Programmer, Manager	 George (29), Software programmer

Exercise 2. Classify Stakeholders

- Classify the stakeholders following the stated categories:
  - A. primary/secondary stakeholder;
  - B. favoured/disfavoured/neutral/ignored.



- Use the table template provided below to place the stakeholders in the appropriate category. Think about the not indicated users that should be considered.

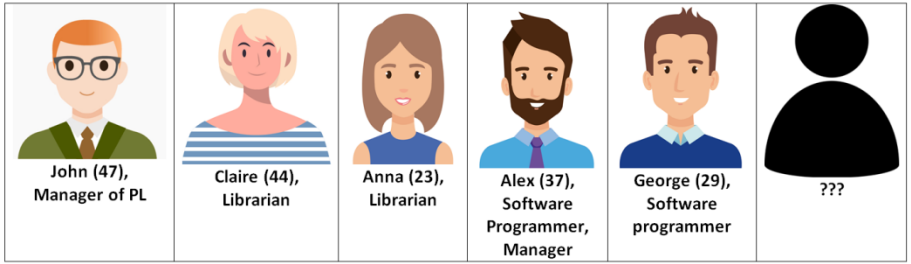
	Primary	Secondary		Favoured	Disfavoured	Neutral	Ignored
Public Library							
Software company							

Software Quality. Definitions

- **Quality is conformance to documented requirements.** [Pressman2005]
- **Quality is conformance to real user requirements (documented or not).** [Crosby1980]
- **Quality is fitness for use.** [Juran1988]
- **Quality is value to some person.** [Weinberg1992]

Exercise 3. Appropriate Software Quality

- Indicate in the table below the software quality definition that fits best the stakeholder perspective.



Software Quality Definition	Stakeholders
Pressman2005	
Crosby1980	
Juran1988	
Weinberg1992	

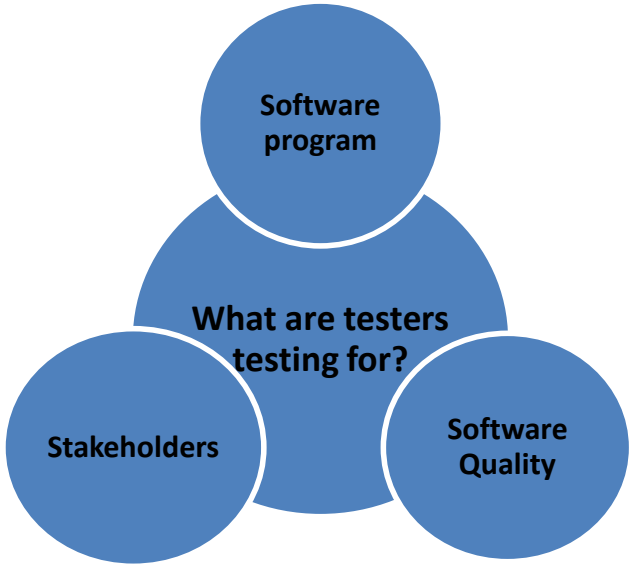
Software Testing. Definitions (1)

1. **“Testing can only reveal the presence of errors, never their absence.” [Dijkstra1969]**
2. **“Testing is the process of executing a program with the intent of finding errors.” [Myers2004]**
3. **“Testing is questioning a product in order to evaluate it.“ [KanerBack2005]**
4. **“Testing is the software behaviour observation during several executions.” [Frențiu2010]**

Software Testing. Definitions (2)

1. **Software testing is**
   - **is an empirical**
   - **technical**
   - **investigation**
   - **conducted to provide stakeholders**
   - **with information**
   - **about the quality**
   - **of the product or service under test [BBST2010].**

All together…



Testers look for different things…

…for different stakeholders.

Bug, Software Error, Failure…

- What is a BUG?

Hint…

A bug is something that *bugs* somebody.

James Bach

Error. Fault. Failure

- **Error (mistake,** *rom.* **greşeală, eroare):**
  - a **human action** that produces an incorrect result [Patton2005], e.g, typos;
- **Fault (bug, defect, software error,** *rom.* **eroare soft, defect, bug)**
  - the **consequence of an error**; [Patton2005]
  - a software system can contain faults but still never fail as long as failure triggers are not exercised;
- **Failure (***rom.* **defecţiune)**
  - a deviation of the software from its expected delivery or service; [Patton2005]
  - it occurs whenever the external behavior of a system does not conform to what was prescribed in the system specification;
  - it is the **manifestation of one or more faults**.

error ---> fault ---> failure

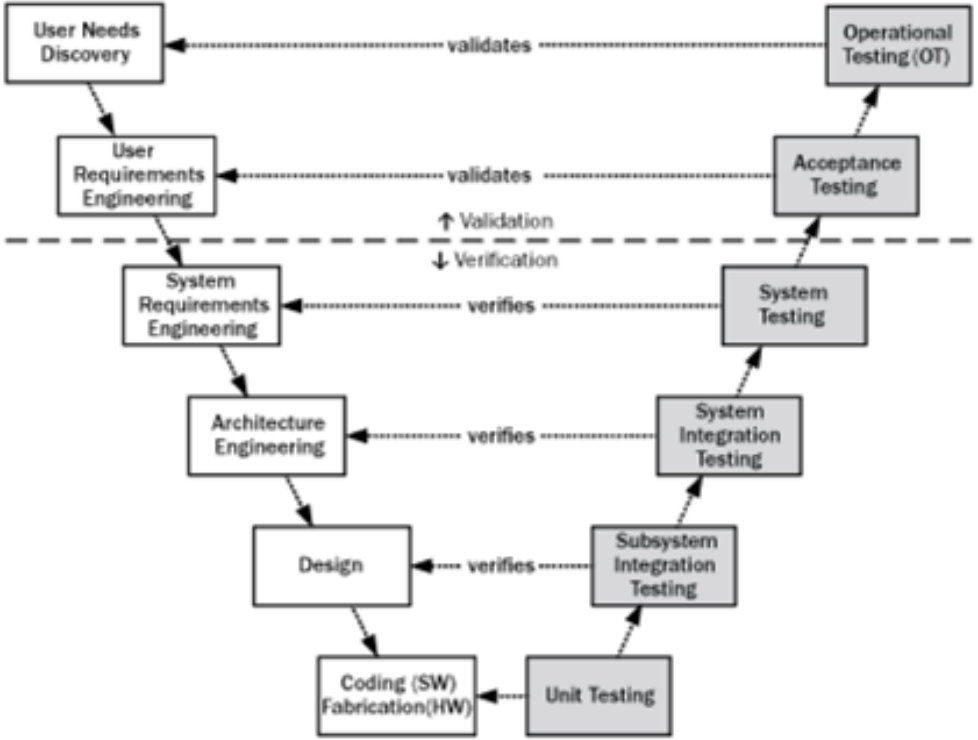
Software Error

- **A software error** is **an attribute of a software product**
  - that **reduces** its **value** to a **favored stakeholder**
  - or **increases** its **value** to a **disfavored stakeholder**
  - without a sufficiently large countervailing benefit. [BBST2010]
- E.g.: *typos*, *design issues*, *usability issues*;

> **Any threat to the value of the product to any stakeholder who matters.**
>
> **James Bach**

- Some aspects limit on purpose the quality of the product but they are not considered bugs.
  - E.g.: usage constraints specified or not in specifications;
- synonyms for bug: **variance, problem, inconsistency, error, incident, anomaly**. [Patton2005]

Testing Levels. V Model



- from: [Firesmith2015]

Test case

- **A test case** is
  - a set of **test inputs, execution conditions, and expected results** developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement." [IEEE1990]

- **A test case** is
  - **a question asked to the program.** [BBST2010]
  - we are more interested in the *informational goal*, i.e., to gain information; e.g., whether the program will pass or fail the test.

- **notation: (i , r), i ϵ D, r ϵ R**;
  - For input ***i*** is expected to achieve result ***r***. [Frentiu2010]

Test case. Attributes

- **Attributes of relevant (good) test cases:**
  - Power
  - Valid
  - Value
  - Credible
  - Representative
  - Non-redundant
  - Motivating
  - Performable
  - Reusable
  - Maintainable
  - Information value
  - Coverage
  - Easy to evaluate
  - Supports troubleshooting
  - Appropriately complex
  - Accountable
  - Affordable
  - Opportunity Cost
- **A test case has each of these attributes to some degree.**

Application Context. Details

- **Testing is performed before, during or after the product under test is released.**
  - improvement of product or process might or might not be an objective of testing;
  - types of stakeholders to test for: **project manager, marketer, customer, programmer, attorney;**

The tester should ask himself *What services wants or needs the client now, under these specific circumstances*?

The tester's goal should be to help the client do the best that it can be done under the given circumstances.

Application Context. Constraints

- **Factor examples that constrain the testing:**
  - Who are the **stakeholders with influence**?
  - Are there non-stakeholders with influence (e.g. regulators)?
  - What are the **goals and quality criteria** for the project?
  - What **skills and resources** (such as *time*, *money*, *tools, data, technology and testability* support) are available?
  - What's in the product?
  - **How could it fail**?
  - **Potential consequences** of potential failures?
  - Who might care about which consequence of what failure?
  - How to recognize failure?
  - How to decide what result variables to attend to?
  - How to decide what other result variables to attend to in the event of intermittent failure?
  - How to troubleshoot and simplify a failure, so as to better
    - motivate a stakeholder who might advocate for a fix?
    - enable a fixer to identify and stomp the bug more quickly?
  - How to expose, and who to expose to, undelivered benefits, unsatisfied implications, traps, and missed opportunities?

Application Context. Examples

Context 1 (Resources)

- Mature product. Time available
  - Lots of automated GUI regression test code, created in previous versions.
  - Some testers have good programming *skills* and know the regression tool's language.
  - *Time* available in the schedule for a thorough round of regression test code maintenance.

- **Suggested strategy: plan for a lot of automated GUI regression testing.**

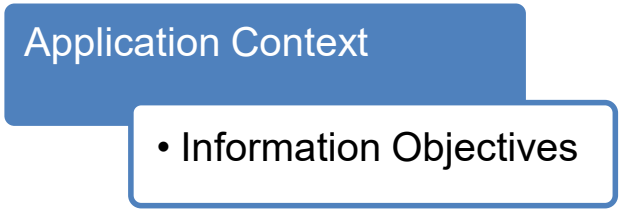
Context 2 (Resources)

- New product. Tight schedule.
  - No pre-existing tests.
  - Testers know the subject matter, the product environment, and some are excellent bug hunters.
  - *None* of the testers *are skilled* programmers.

- **Suggested strategy: plan for intensely exploratory testing: risk-focused, no automated regression, not much test documentation.**

Information Objectives

- **the information acquired during testing may have various objectives** [BBST2010]:
  - Find important **bugs**;
  - Assess the **quality** of the product;
  - Help managers assess the **progress** of the project;
  - Help managers make **release decisions**;
  - **Block premature** product releases;
  - Help **predict** and control product support **costs**;
  - Check **interoperability** with other products;
  - Find **safe scenarios** for use of the product;
  - Assess **conformance** to specifications;
  - Certify the product meets a particular **standard**;
  - Ensure the testing process meets **accountability** standards;
  - **Minimize the risk** of safety-related lawsuits;
  - Help clients **improve product quality** & testability;
  - Help clients improve their **processes**;
  - Evaluate the product for a **third party**.



Different objectives require different testing tools and strategies, that will yield different tests, test documentation and test results.

Information Objectives. Examples

Context 1:

- Mass-market software, close to release date. The test group believes **the product is too buggy** and that better-informed stakeholders would not ship it.

Information objectives:

- The testers are likely to do **bug-hunting**, **looking for important bugs** that will cause key stakeholders to reconsider **whether are willing to release the product**.

Context 2:

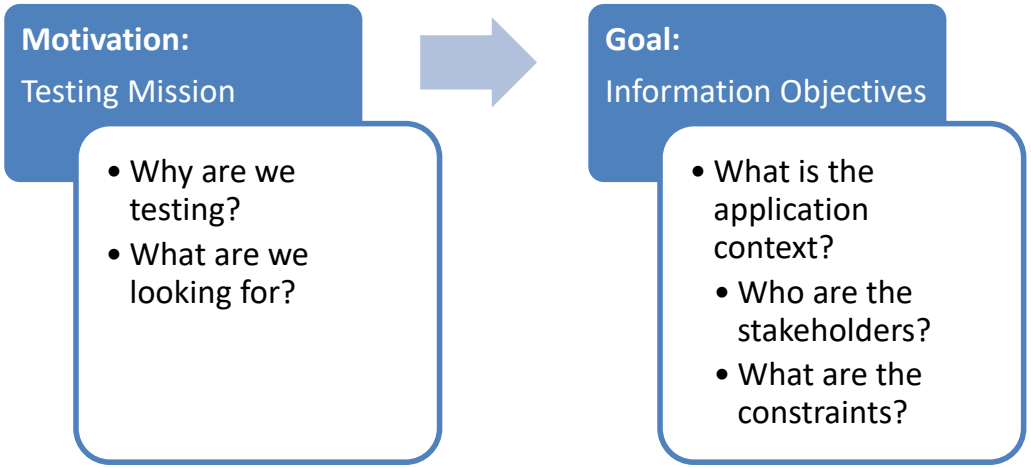
- Software fails in use and causes **serious losses**. A law firm hires testers to determine what caused the failures and when the seller found these bugs.

Information objectives:

- The testers will not do general bug-hunting. They are likely to determine **how** (and in how many ways) **they can replicate specific failures** and **they will study corporate quality records** to **ensure the testing process meets accountability standards**.

Testing Mission. Details

- typically, the testing mission is performed to achieve (to obtain) the primary information objectives;
  - if there are too many objectives, the mission (the testing activity) is fragmented, and probably will not achieve the needed outcome;
  - **less objectives increase mission awareness and focus on the testing to be done.**



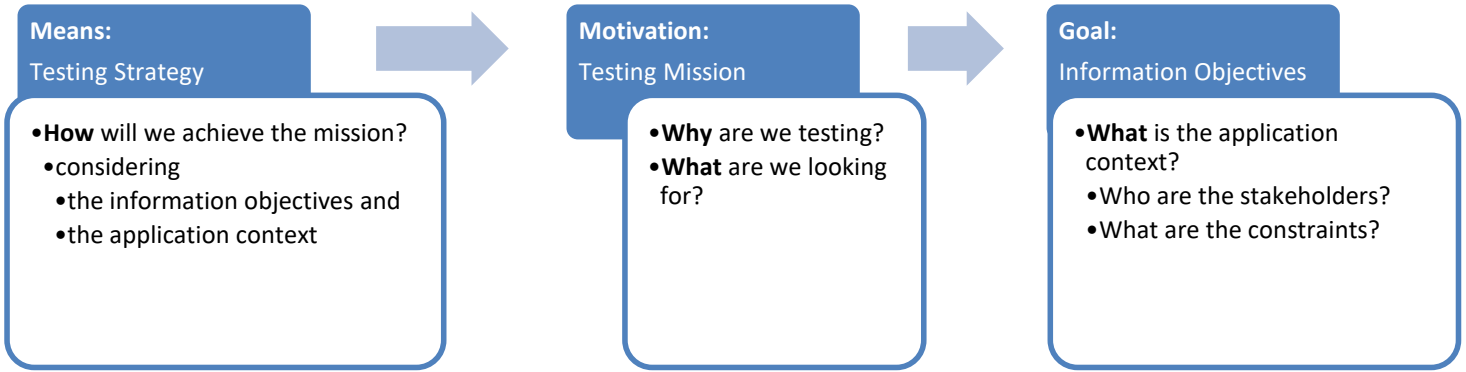
Testing Mission. Example

- generally, **the testing mission changes over the course of a project**;
- E.g.: A 6-month development project, with first code delivery to test in month 2.
  - month 2 to 5 may be ***bug-hunting***:
    - harsh tests in areas of highest risk;
    - exploratory scans for unanticipated areas of risk;
  - month 6 may be ***helping the project manager determine whether the product is ready to ship***:
    - status and quality assessments, less testing;
    - tests include coverage-oriented surveys.

Testing mission should be explicit. Testers should be worried if they are of trying to achieve several missions at the same time.

Testing Strategy. Definition

- **A testing strategy** is
  - the *guiding framework* for deciding what tests (**what test techniques**) are best suited to some project,
  - given the project's **objectives** and **constraints** (the application context);
  - and the **informational objectives** of the addressed testing. [BBST2010]
- **A testing strategy** is
  - **a generalized description of the test process**, usually at the product or organizational level. [ISTQB2018]
  - **the set of ideas that guide the test design process;**



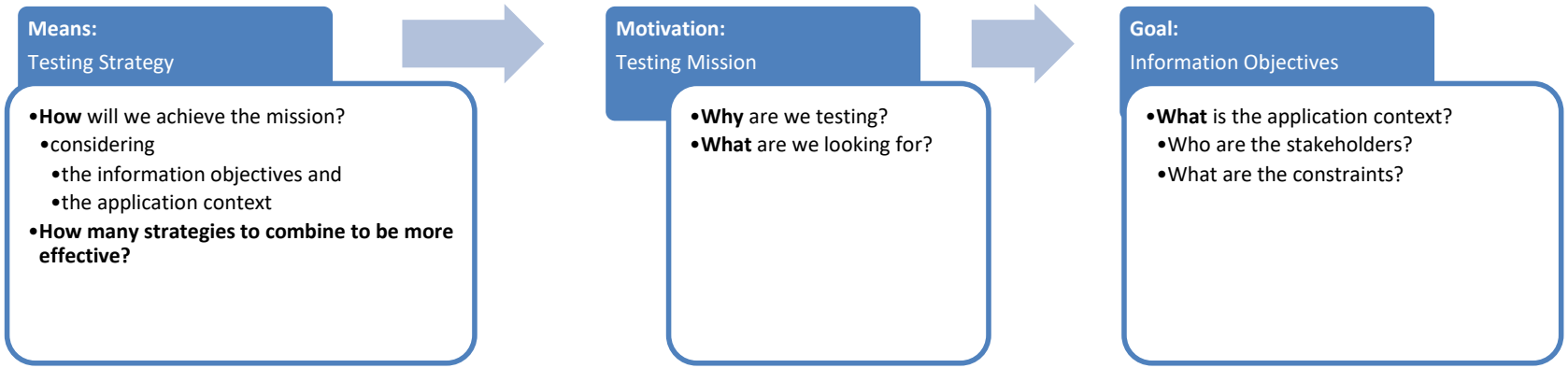
Testing Strategy. Types

- **types of testing strategies** [ISTQB2018]**:**
  - **analytical** – based on an analysis of some factor, e.g., requirement or risk; in risk-based testing tests are *designed* and *prioritized* based on the level of risk.
  - **model-based** – tests are based on some *model* of some required aspect of the product, such as a function, a business process, an internal structure, or a non-functional characteristic (e.g., reliability); e.g., business process models, state models, and reliability growth models.
  - **methodical** – relies on making systematic use of some predefined set of tests or test conditions, such as a *taxonomy* of common or likely types of *failures*, a list of important quality characteristics, or company-wide look-and-feel standards for mobile apps or web pages.
  - **process-compliant**, i.e., standard-compliant – involves analyzing, designing, and implementing tests based on external rules and standards, such as those specified by industry-specific standards, by process documentation, by the rigorous identification and use of the test basis, or by any process or standard imposed on or by the organization.
  - **directed**, i.e., consultative – driven primarily by the advice, guidance, or instructions of stakeholders, business domain experts, or technology experts, who may be outside the test team or outside the organization itself.
  - **regression-averse** – motivated by a desire to avoid regression of existing capabilities; it includes reuse of existing testware (especially test cases and test data), extensive automation of regression tests, and standard test suites.
  - **reactive** – testing is reactive to the component or system being tested, and the events occurring during test execution, rather than being pre-planned (as the previous strategies do); tests are designed and implemented, and may immediately be executed in response to knowledge gained from prior test results; exploratory testing is a common technique employed in reactive strategies.

Testing Strategy. Details

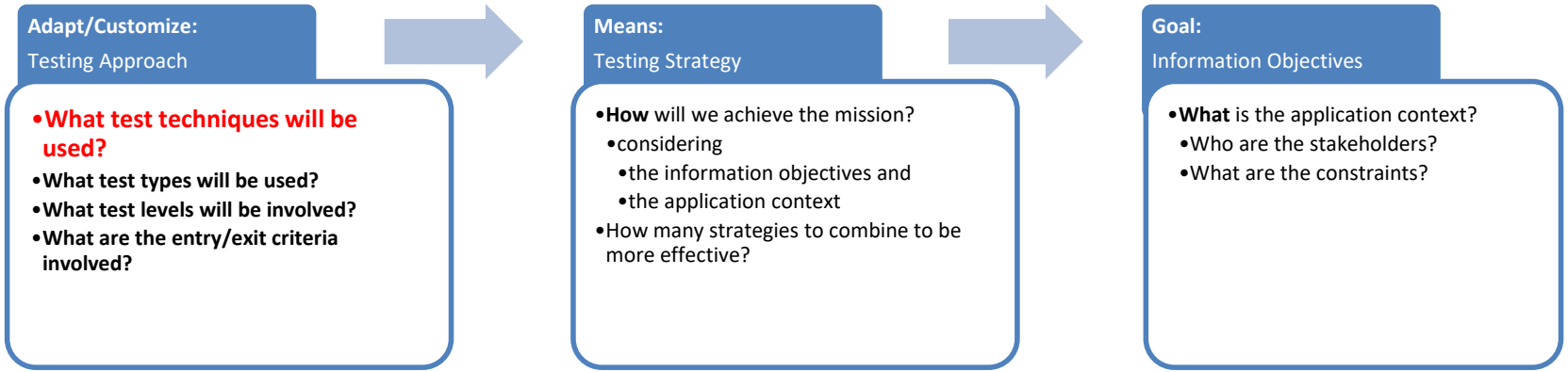
- Types of testing strategies:
  - analytical, model-base, methodical, reactive, etc.

- **An appropriate test strategy is often created by combining several types of test strategies;**
  - E.g.: **risk-based testing** (an **analytical strategy**) can be combined with **exploratory testing** (a **reactive strategy**); they complement each other and may achieve more effective testing when used together.



Testing Approach. Definition

- **A test approach** is
  - **the adjustment of the testing strategy for a particular project**;
  - it allows selecting *the test techniques, test levels, and test types, and for defining the entry criteria and exit criteria*.
  - it depends on the application context and may consider factors such as risks, safety, available resources and skills, technology, the nature of the system (e.g., custom-built versus COTS), test objectives, and regulations.



Testing Approach. Types (1)

- **Black-box testing** (**behavioral testing**) is
  - testing without using knowledge about the source code;

  The black box tester becomes an *expert in the relationships between the program and the world in which it runs*.
  - *Question:* Does this do what the users (human and software) expect?

- **White-box testing** (**structural testing**) is
  - testing using knowledge about the internals of the program, i.e., data, source code;

  The glass box tester becomes an *expert in the implementation of the product under test*.
  - *Question:* Does this code do what the programmer expects or intends?”

- **Grey-box testing** has no standard definition;
  - a **blend** of *black-box* and *white-box* approaches.

Testing Approach. Types (2)

- **Exploratory testing** is
  - style of software testing that emphasizes the personal freedom and responsibility of the individual tester to **continually** optimize the value of her work
  - by treating
    - test-related learning,
    - test design,
    - test execution, and
    - test result interpretation as
  - mutually supportive activities that **run in parallel throughout the project**.

- **Scripted testing** is
  - style of software testing where **test cases are designed in advance**, both the individual steps and the expected results; these tests are later performed by a tester who compares the actual result with the expected one;
  - **opposed to exploratory testing.**

Test Design Technique. Definition

- **A technique** is
  - the body of specialized procedures and methods used in any specific field, especially in an area of applied science;
  - *method of performance* or *way of accomplishing* something.

- **Test design technique (TDT)** is
  - **a method of designing, running and interpreting the results of tests.**

- E.g.:
  - **function testing, specification-based testing, tours,**
  - **quick-tests, scenario-based testing,**
  - **confirmation testing, beta testing,** etc.

Test Design Technique (TDT). Details

- **Test design technique** is
  - a method of designing, running and interpreting the results of tests.

- techniques involve **skill**
  - someone gets better at applying it as he gains experience with it.
- techniques are **more action than theory**
  - someone needs some theoretical background to understand a technique, and a technique might apply theoretical knowledge, but the technique itself is about how to do a type of testing.
- techniques **are different from each other, they are complementary**
  - **any technique will be more effective obtaining some types of knowledge, e.g. some types of bugs, but less effective for others.**

Approach vs. Technique. Example (1)

- **Example 1:**
  - Some testers call **exploratory testing** and **scripted testing** test ***techniques***.

- Any technique may be used in
  - an exploratory way or
  - a scripted way or
  - a way that includes both exploratory and scripted *elements*.

- ***Exploration* and *script-following* reflect broad visions about the best way to organize and do testing, not specific tactics for designing individual tests**. Therefore, they are called ***approaches*** rather than techniques.

Approach vs. Technique. Example (2)

- **Example 2:**
  - When someone says he will do **black-box testing** he doesn't know:
    - what he will actually **do**,
    - what tools he will **use**,
    - **what bugs** he will look for,
    - **how** he will look for them,
    - or how he will decide whether **he has found a bug**.
- **Some techniques are more likely to be used in a black-box way**, so they are called "black-box techniques." **It is the technique,** e.g., "usability testing", **that has black-box characteristics, not "black-box" that is the technique.**
- **The approach of the technique is black-box.**

TDT Dimension. Definition

- **A test design technique dimension** is
  - **an aspect of the testing that the test design technique focuses on;**
- synonyms for **dimension**
  - **driving ideas;**
  - **essential characteristics;**
  - **perspective**;

- test design techniques are classified according to such dimensions, making them useful to be applied in certain cases when compared to others;

- **there are 7 dimensions:**
  - **scope, coverage, person who achieves testing, associated risks, activities, evaluation and desired results.**

TDT Dimension. Types (1)

- There are seven dimensions (1-3):
  - **Scope**: **what** gets tested;
    - E.g.: **function testing** technique tests individual functions;
  - **Coverage**: intended extent of testing, **how much** of it will is;
    - E.g.: **function testing** technique tests every function; typically, the tester analyzes the *scope* (what to test) and *coverage* (how much of it) together.
  - **Testers**: **who** does the testing;
    - E.g.: user testing is focused on testing by people who would normally use the product.

TDT Dimension. Types (2)

- There are seven dimensions (4-7):
  - **Risks:** potential problem you're testing for, **what problems** may have the product.
    - E.g.: testing for **extreme value (boundary) errors**.
  - **Activities: how** are actually done the tests.
    - E.g.: **All-pairs testing** specifies how you combine conditions to obtain test cases.
  - **Evaluation / Oracle: how to tell whether the test passed or failed**;
    - E.g.: **function equivalence testing** relies on comparison to a reference function.
  - **Desired result:** testing with a tightly-defined objective; **what is the testing objective**;
    - E.g.: **build verification testing** checks whether the build is stable enough for more thorough testing.

TDT Dimension. Details

- **All testing involves all dimensions.**
  - **A testing technique *focuses* the attention on one or a few dimensions, *leaving* the others open to the tester's judgment.**
- the tester can combine a technique that is focused on one dimension with techniques focused on the other dimensions to achieve the result he wants.
- he might call the result of such a combination a *new technique* (some people do), but the process of thinking is more useful than adding another name to the ever-expanding list of inconsistently defined techniques in use in the testing field.

TDT Dimension. Example (1)

- Example: **Function testing technique**
  - Focus on:
    - **Scope**: Focus on individual functions, testing them one by one;
    - **Coverage**: Test every function (or a subset) of the product;
  - What *function testing* doesn't specify:
    - **Testers**: **Who** does the testing;
    - **Risks**: **What bugs** we're looking for;
    - **Activities**: **How to run the tests;**
    - **Evaluation / Oracles**: **How to evaluate the test results;**
    - **Desired result: What is the testing objective**.

TDT Dimension. Example (2)

- many techniques implement more than one underlying idea.
  - a technique is classified depending on what the tester has in mind when he uses it.
- Example: **Feature integration testing**
  - is **coverage-oriented** if the tester is checking whether every function behaves well when used with other functions;
  - is **risk-oriented** if the tester has a theory of error for interactions between functions.

TDTs Classification

- The main test design techniques are:
  - **Black-box approach:**
    - **Coverage-based techniques;**
    - **Tester-based techniques;**
    - **Risk-based techniques;**
    - **Activity-based techniques;**
    - **Evaluation-based techniques;**
    - **Desired result techniques;**
  - **White-box approach:**
    - **Glass-box techniques.**

TDTs Classification. Coverage-based Techniques

- **Coverage-based techniques:**
  1. Function testing;
  2. Feature / function integration testing;
  3. Tours;
  4. Equivalence class analysis;
  5. Boundary testing;
  6. Best representative testing;
  7. Domain testing;
  8. Test idea catalogues;
  9. Logical expressions;
  10. Multivariable testing;
  11. State transitions;
  12. User interface testing;
  13. Specification-based testing;
  14. Requirements-based testing;
  15. Compliance-driven testing;
  16. Configuration testing;
  17. Localization testing.

A coverage-based technique means the tester should run every test of a given type. In practice, the tester probably will not every test of any type, but the tester might measure the coverage of that type of testing.

TDTs Classification. Tester-based Techniques

- **Tester-based techniques:**
  1. User testing;
  2. Alpha testing;
  3. Beta testing;
  4. Bug bashes;
  5. Subject-matter expert testing;
  6. Paired testing;
  7. Eat your own dog food;
  8. Localization testing.

There is a mystique in designing a technique around the type of person who tests. However, what they will actually do may have little to do with what someone may imagine will happen.

TDT Classification. Risk-based Techniques

- **Risk-based techniques:**
  1. Boundary testing;
  2. Quick-tests;
  3. Constraints;
  4. Logical expressions;
  5. Stress testing;
  6. Load testing;
  7. Performance testing;
  8. History-based testing;
  9. Risk-based multivariable testing;
  10. Usability testing;
  11. Configuration / compatibility testing;
  12. Interoperability testing;
  13. Long sequence regression.

Risk-based testing starts from an idea of how the program could fail. Then test cases are designed in order to try to expose problems of that type of failure.

TDTs Classification. Activity-based Techniques

- **Activity-based techniques:**
  1. Guerrilla testing;
  2. All-pairs testing;
  3. Random testing;
  4. Use case testing;
  5. Scenario testing;
  6. Installation testing;
  7. Regression testing;
  8. Long sequence testing;
  9. Dumb monkey testing;
  10. Load testing;
  11. Performance testing.

Activity-based techniques focus on "how-to", these might be the techniques that most closely match the classical notion of a “technique”.

TDTs Classification. Evaluation-based Techniques

- **Evaluation-based techniques:**
  1. Function equivalence testing;
  2. Mathematical oracle;
  3. Constraint checks;
  4. Self-verifying data;
  5. Comparison with saved results;
  6. Comparison with specifications or other authoritative documents;
  7. Diagnostics-based testing;
  8. Verifiable state models.

If the tester has a well-specified oracle, he can build a set of tests around that oracle.

TDTs Classification. Desired-result Techniques

- **Desired-result techniques:**
  1. Build verification;
  2. Confirmation testing;
  3. User acceptance testing;
  4. Certification testing.

Document-focused testing **is performed if the tester runs a set of tests primarily to collect data needed to fill out a form or create a clearly-structured report.**

Lecture Summary

- We have discussed:
  - stakeholders;
  - computer program;
  - software testing;
  - test case and its attributes;
  - application context, information objectives;
  - testing mission, testing strategy, testing approach;
  - test design technique definition;
  - 7 relevant aspects of a test design technique;
  - test design technique classification based on the discussed dimensions.

Lab Activities on week 01-02

- Tasks to achieve in week 01-02 during Lab 01:
  - **Exercise 1** and **3** from **Lecture 01**;
  - **Play the game *"Testing is…"*** and **vote your favorite testing definition**;
  - **Install a *mind mapping tool*** and **elaborate a mind map for a testing concept** presented in **Lecture 01** or **Lecture 02**.

Next Lecture

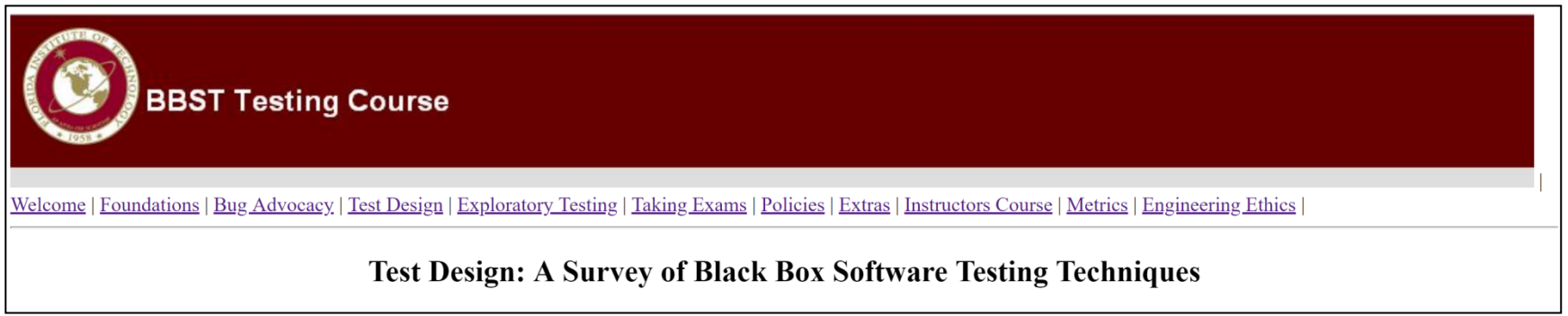
- Coverage-based techniques
- Focus. Objectives;
- Techniques and Examples:
  - Function testing;
  - Tours;
  - Equivalence Class Partitioning;
  - Boundary Value Analysis;
  - Domain Testing;
  - Specification-based Testing;
  - etc.

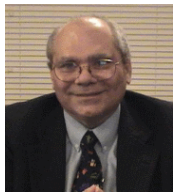
References

- **[CFI2021]** Stakeholders, https://corporatefinanceinstitute.com/resources/knowledge/finance/stakeholder/
- **[StakeholderMap2021]** Stakeholders, https://www.stakeholdermap.com/primary-stakeholders.html
- **[GauseWeinberg2011]** Donald C. Gause, Gerald M. Weinberg, Exploring Requirements: Quality Before Design, Dorset House, 2011.
- **[KanerBach2005]** Kaner, C., Bach, J., Requirements Analysis for Test Documentation, http://www.testingeducation.org/BBST/extras/BBSTTestDocs2005.pdf.
- **[Rochkind2004]** Rochkind, Marc J. (2004). Advanced Unix Programming, Second Edition. Addison-Wesley. p. 1.1.2.
- **[Frentiu2010]** M. Frentiu, Verificarea si validarea sistemelor soft, Presa Universitara Clujeana, 2010.
- **[BBST2010]** BBST – Fundamentals of Testing, Cem Kaner, http://www.testingeducation.org/BBST/foundations/BBSTFoundationsNov2010.pdf.
- **[Pressman2000]** Roger S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill, Inc., 2000.
- **[Crosby1980]** Philip B. Crosby, Quality Is Free, Signet Shakespeare, 1980.
- **[Juran1998]** A. Blanton Godfrey, Joseph Juran, Juran's Quality Handbook, McGraw-Hill, 1998.
- **[Weinberg1992]** Gerald Weinberg, Quality Software Management , Vol. 1: Systems Thinking, Dorset House Publishing, 1992.
- **[Patton2005]** R. Patton, Software Testing, Sams Publishing, 2005.
- **[Dijkstra1969]** E.W. Dijkstra, Software engineering techniques, Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27-31 October 1969.
- **[Myers2004]** Glenford J. Myers, The Art of Software Testing, John Wiley & Sons, Inc., 2004
- **[IEEE1990]** IEEE STD 610, In IEEE Standard Glossary of Software Engineering Terminology, 1990.
- **[BBST2011]** BBST – Test Design, Cem Kaner, http://www.testingeducation.org/BBST/testdesign/BBSTTestDesign2011pfinal.pdf.
- **[Firesmith2015]** Donald Firesmith, *Four Types of Shift Left Testing,* https://insights.sei.cmu.edu/sei_blog/2015/03/four-types-of-shift-left-testing.html

Acknowledgements

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





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