LECTURE 10A.

EVALUATION-BASED TECHNIQUES

Test Design Techniques

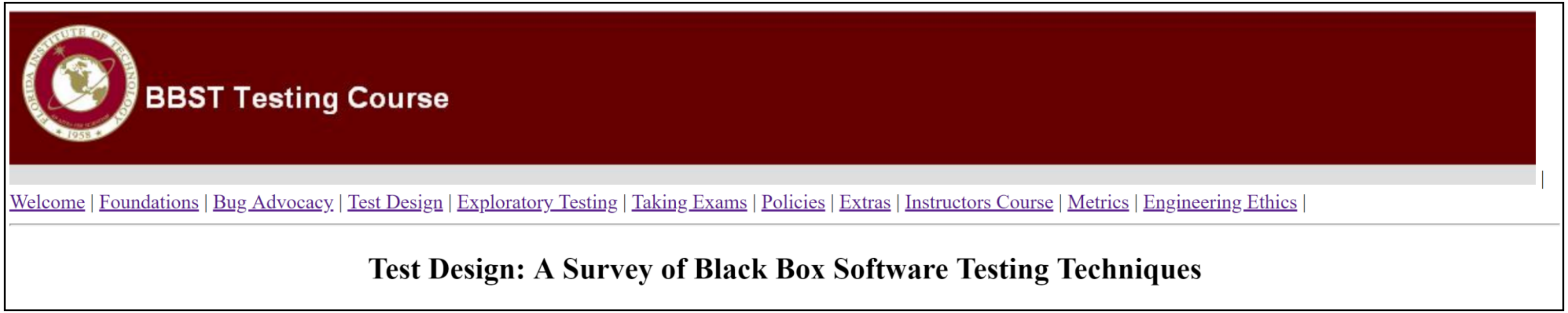
[04 May 2022]

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

- Last lecture…
  - **Bug reporting**
- **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.
- **Desired-result techniques**
  1. Build verification;
  2. Confirmation testing;
  3. User acceptance testing;
  4. Certification testing.

Last Lecture...

- Topics approached in Lecture 09:
- **Bug Reporting**
  - RIMGEA;
  - Type of Bugs
    - Coding Bugs;
    - Design Bugs;
  - Examples;
  - Quality-based Bug Taxonomy.

TDTs Taxonomy

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

Test Case. Attributes

- **A test case** is
  - a question you ask 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.
- 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.**

Evaluation-based Techniques

- **An evaluation-based technique describe**
  - **a method for determining whether the program passed or failed the test.**

- **it does not specify *how* the testing should be done or *how* the data should be collected;**
  - **it tells that once data is collected the tester can evaluate it;**
- it relies on an **oracle** that can be used in automated testing;

- **an oracle is**
  - **a mechanism or heuristic principle for determining whether a program has a problem;**
- **the oracle**
  - **does not have to be complete or strong or always correct;**
  - **does have to be useful and specific enough to decide if the result obtained during testing is right or wrong.**

Evaluation-based Techniques. Focus

Evaluation-based techniques focus on *how to tell whether the test passed or failed*.

- E.g.: **mathematical oracle** is focused on comparison to a known good result.

Evaluation-based Techniques

- **Evaluation-based Techniques:**
  - Function equivalence testing;
  - Mathematical oracle;
  - Constraint checks;
  - Self-verifying data;
  - Comparison with saved results;
  - Comparison with specifications or other authoritative documents;
  - Diagnostics-based testing;
  - Verifiable state models.

Function Equivalence Testing. Definition

- **Function equivalence testing** allows
  - to test a function considering **another function** that is considered to have the same behaviour and **to be already tested**;
- the function in SUT = the test function;
- the other function = **the reference function** or **the oracle function**;
- the tester may compare the program's evaluations of hundreds (or billions) of sets of data;
  - he either finds a difference between the functions or conclude he has tested so much that he will not find a difference with further testing.
- E.g.: the square-root bug in [Hoffman2003];

Evaluation: Another function is used a reference function to decide whether the function under test works properly.

Mathematical Oracle. Definition

- **A mathematical oracle** allows
  - to derive a **predicted value** from the mathematical attributes of the software under test;
- E.g.:
  - invert calculations (square a square root, or invert a matrix inversion);
  - a sine function’s shape is predictable;
- **Mathematical oracle** vs **Function equivalence testing**
  - *similarities:*
    - running tests with arbitrarily many values;
    - making and checking exact predictions;
  - *differences:*
    - **function equivalence testing** – everything is done using **an external reference function;**
    - **mathematical oracle** – everything is done **within the software under test.**

Evaluation: It relies on mathematical characteristics of the function in SUT.

Constraint Checks. Definition

- **Constraint check testing** allows
  - to check if the program has not output a result that **is impossible**, i.e., **it cannot happen**;
- something does not have to be truly impossible;
  - **it just has to be unlikely enough that it would be worth the tester's time to investigate why the program gave that result;**
- E.g.:
  - an American postal code can't be 6 digits;
  - a Canadian province's name can be checked against a short list of provinces;
  - in an order entry system, the order number of an order should not be smaller than the number of an order that was placed later.

Evaluation: It does not tell if result is correct or not, it checks if it is impossible only.

Self-Verifying Data. Definition

- **Self-verifying data (oracle)** allows
  - to embed the **correct test result in a set of test data**;

- E.g.:
  - add a comment field to a database of test case records;
  - provide a checksum, hash code, or digital signature to authenticate the result;
  - functions built into the SUT that serve as the equivalent of embedded test data by providing the should-be-correct result on demand.

Evaluation: It allows to check the test result by embedding the right answer into the test.

Comparison with Saved Results. Definition

- **A comparison with saved results testing** allows
  - to consider **already existing testing results** when decide if the current results are right or wrong;
- E.g.: **regression testing** – **the most common example of a technique built around saved results**;
- steps:
  1. run a test;
  2. if the program passes, keep its output data;
  3. after a new build, run the test again;
     - check whether the new test results match the saved test results.

Evaluation: It allows to investigate the current test run results based on previously saved results of the same test run.

Comparison with Saved Results. Details

- **Comparison with saved results testing** vs **Self-Verifying Data**
    - *similarities:*
        - **comparison with saved results** is **a special case** of **self-verifying data**;
            - it uses test related data to check the test results;
    - *differences:*
        - the data required to perform comparison is not **embedded** but obtained by **previous test runs**;
        - *the test results may become obsolete* whenever the program changes;
            - this is the main challenge of regression testing.

Comparison with Specifications or Other Authoritative Documents. Definition

- **A comparison with specifications or other authoritative documents** allows
  - to check **existing documents**, e.g., specifications, when decide whether the current results are right or wrong;
- E.g.: **specification-based testing** checks the product against every factual claim made about the product in the specification or any other document that the program must verify against;
  - **a factual claim** is any statement that can be shown to be **true** or **false**;
  - approaches:
    - **coverage-based techniques:** when the tester thinks about **every claim** and he designs tests to check each claim;
    - **evaluation-based technique:** when the tester thinks about **what claims** and he designs tests considering an oracle, i.e., (part of) specifications.

Evaluation: Specifications are oracles as they tell what the program should do.

Diagnostics-based Testing. Definition

- **A diagnostic** is
  - **the mechanism** used to determine whether a program has a problem;
- **Embedded diagnostics** are
  - diagnostics **the programmers write into the program** for testers to use;
- they may indicate unexpected values of internal variables ---> **they show problems before running into visible failures;**
  - this helps bug replication and troubleshooting;
- **Diagnostics-based testing** allows
  - to consider **(embedded) diagnostics** to decide whether the program works correctly;
- **it might not tell what the "right" behavior is; it alerts that something looks wrong.**

Evaluation: It allows to use investigate the current test run results based on (embedded) diagnostics in the program.

Diagnostics-based Testing. Details

- approaches:
  - **during running the test;**
    - as part of the normal (or test-customized) operation of the program, the program runs diagnostics;
      - if the test triggers an unusual state, the program reports a diagnostic issue.
  - **after running the test;**
    - the tester runs a diagnostic immediately after running the test;
    - the diagnostics can expose effects of the test that would otherwise be invisible, such as:
      - memory corruption,
      - assignment of incorrect values to internal variables,
      - tasks that were only half-completed,
      - etc.

Verifiable State Models. Definition

- **A verifiable state testing** allows
  - to consider an *oracle* to compare the program's behavior **to a model of how it should behave;**
- when a program is in any given state, it will ignore some inputs (or other events) and respond to others;
  - the program's response takes it to its next state, i.e., *a state transition*;
- **state model-based testing** can be performed if:
  - *a state model* that ties inputs to transitions, and
  - ability to tell whether *the program is actually in the state predicted by the model, i.e.,* ***a verifiable state model*****.**

Evaluation: It allows to use the state model as an oracle to check if the program has reached with the predicted state.

Next…

- **Desired result techniques**

References

- **[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.
- **[Kaner2000]** Kaner, C., Falk, J., & Nguyen, H.Q. (2nd Edition, 2000b), Bug Taxonomy (Appendix) in Testing Computer Software, Wiley, http://www.logigear.com/logi_media_dir/Documents/whitepapers/Common_Software_Errors.pdf
- **[Bach1999]** Bach, J. (1999), Heuristic risk-based testing, Software Testing & Quality Engineering, http://www.satisfice.com/articles/hrbt.pdf
- **[Hoffman2003]** Hoffman, D. (2003). Exhausting your test options. Software Testing & Quality Engineering, 5(4), 10-11.