Distinguish (among others), alternative: test item

- **NULL**: for example, not a valid input

If a tester does not adhere to an allowed input sequence of a description, e.g.:

```
S\rightarrow T
```

Expected outcome(s)

```
<table>
<thead>
<tr>
<th>Soll</th>
<th>Test Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>/c01</td>
<td>abc</td>
</tr>
<tr>
<td>/c02</td>
<td>/c01</td>
</tr>
</tbody>
</table>
```

In a description, test case `abc` refers to input `abc`.

### Other Approaches

- Glass-Box Testing
- Generic
- Term-
- Statement-
- Branch-

### Limits of Software Testing

- High quality software - based
- Statistical

### Proof System PD

- Dr. Bernd Westphal
- Albert-Ludwigs-Universität Freiburg, Germany

### Software Examination Paths

- in a context

### Other Approaches

- Glass-Box Testing

### Limits of Software Testing

- High quality software - based
- Statistical

### Proof System PD

- Dr. Bernd Westphal
- Albert-Ludwigs-Universität Freiburg, Germany

### Software Examination Paths

- in a context
A test suite is a finite set of test cases \( \{T_1, \ldots, T_n\} \).

An execution of a test suite is a set of computation paths, such that there is at least one execution for each test case.

An execution of a test suite is called positive if and only if at least one test case execution is positive. Otherwise, it is called negative.

**Tests vs. Systematic Tests**

Systematic test — a test such that

- (environment) conditions are defined or precisely documented,
- inputs have been chosen systematically,
- results are documented and assessed according to criterias that have been fixed before.

\((Ludewig \ and \ Lichter, \ 2013)\)

Test — (one or multiple) execution(s) of a program on a computer with the goal to find errors.

\((Ludewig \ and \ Lichter, \ 2013)\)

Not (even) a test (in the sense of this weak definition)

- any inspection of the program (no execution),
- demo of the program (other goal),
- analysis by software-tools for, e.g., values of metrics (other goal),
- investigation of the program with a debugger (other goal).

\((Our) \ Synonyms \ for \ non-systematic tests: \ Experiment, \ 'Rumprobieren' \ .\)

In the following: test means systematic test; if not systematic, call it experiment.
Specific Testing Notions Cont’d

• Which roles are involved in testing?
  • inhouse test — only developers (meaning: quality assurance roles),
  • alpha and beta test — selected (potential) customers,
  • acceptance test — the customer tests whether the system (or parts of it, at milestones) is acceptable.

Content

Software Testing Introduction
• Test suite; Tests vs. systematic tests.
• More vocabulary
• Limits of Software Testing
• Software examination paths
• Is exhaustive testing feasible?
• Choosing Test Cases
• Generic requirements on good test cases
• Point vs. range errors
• Approaches:
  • Statistical testing
  • Expected outcomes: Test Oracle: -/-
  • Habitat-based
  • Glass-Box Testing
  • Statement/Branch/term coverage
  • Conclusions from coverage measures

When To Stop Testing?
• Model-Based Testing
• Testing in the Development Process

The Limits of Software Testing

In each examination, there are two paths from the specification to results:
• the production path (using model, source code, executable, etc.), and
• the examination path (using requirements specifications).

A check can only discover errors on exactly one of the paths.
If a difference is detected: examination result is positive.
What is not on the paths, is not checked; crucial: specification and comparison.
Recall: checking procedure shows no error reports error
artefact has error
specification implement specification comprehend specification
requirements on result compare examination result ✔/✘/?
information flow development information flow examination (Ludewig and Lichter, 2013)

Recall: Quotes On Testing

"Software testing can be used to show the presence of bugs, but never to show their absence!" (E. W. Dijkstra, 1970)

Why Can’t We Show The Absence of Errors (in General)?

Consider a simple pocket calculator for adding 8-digit decimals:

12345678
+  27
789456
+ 123
= 1464523

• Requirement: If the display shows \(x\), \(+\), and \(y\), then after pressing \(=\),
  • the sum of \(x\) and \(y\) is displayed if \(x + y\) has at most 8 digits,
  • otherwise "-E-" is displayed.

• With 8 digits, both \(x\) and \(y\) range over \([0, 10^8-1]\).
• Thus there are \(10^{16}\) possible input pairs \((x, y)\) to be considered for exhaustive testing, i.e. testing every possible case!
• And if we restart the pocket calculator for each test, we do not know anything about problems with sequences of inputs...
  (Local variables may not be re-initialised properly, for example.)
• Example: Simple Pocket Calculator.

With ten thousand (10,000) different test cases (that’s a lot!), 9,999,999,999,990,000 of the $10^{16}$ possible inputs remain uncovered.

In other words: Only 0.0000000001% of the possible inputs are covered, 99.99999999% not touched.

• In diagrams: (red: uncovered, blue: covered)

Software Testing Introduction

• Test suite; Tests vs. systematic tests.

• More vocabulary

• Limits of Software Testing

• Software examination paths

• Is exhaustive testing feasible?

• Choosing Test Cases

• Generic requirements on good test cases

• Reproducibility

• Test executions should be (as) reproducible and objective (as possible).

• So, strictly speaking, a test case is a triple $(I, S, E)$ comprising a description $E$ of (environmental) conditions.

$E$ describes any aspects which could have an effect on the outcome of a test execution and cannot be specified as part of $I$, such as:

• Which program (version) is tested?
• Built with which compiler, linker, etc.?
• Test host (OS, architecture, memory size, connected devices (configuration?), etc.)?
• Which other software (in which version, configuration) is involved?
• Who is supposed to test when?

Full reproducibility is hardly possible in practice — obviously (err, why. . . ?).

• Steps towards reproducibility and objectivity:
  • have a fixed build environment,
  • use a fixed test host which does not do any other jobs,
  • execute test cases automatically (test scripts).

How to Choose Test Cases?

• A first rule-of-thumb: “Everything, which is required, must be examined/checked. Otherwise it is uncertain whether the requirements have been understood and realised.”

• In other words:
  • Not having at least one (systematic) test case for each (required) feature is (grossly?) negligent. (Dt.: (grob?) fahrlässig).

• In even other words: Without at least one test case for each feature, we can hardly speak of software engineering.

• Good project management: document for each test case which feature(s) it tests.
Testing in the Development Process

• Model-Based Testing
• When To Stop Testing?

From coverage measures

• coverage term
  • Branch

• Significance niveau may be unsatisfactory with small testsuites.

Glass-Box Testing

• correct" with a certain significance niveau.

• hypothesis
  • Software
    • Habitat
      •: if no error is found
      •: Expected outcomes: Testing
        •: If an error is found

Approaches:

• Statistical Testing
  • Is exhaustive testing feasible?

Limits of Software Testing

• More vocabulary

Point vs. Range Errors

• Some values to others
  • conclude from
  • For software, we can (in general, without extra information) not

• δ
  • = 3
  • 2
  • 1
  • etc.

• ε

• (0
  • 9
  • 8
  • 7
  • 6
  • 5
  • 4
  • 3
  • 2
  • 1
  •)

• (12345678
  • , etc.

• (12345705)

• (12345678
  • ,

• wikipedia: multiple "neighbouring" inputs trigger the error.

• Point error
  • an isolated input value triggers the error.

• Range error
  • different classes of inputs considered in the requirements, like "C50", "E1" coins in the vending machine.

• Recall: the developer is not a good tester.

• statistical assumptions on error distribution and truly random test cases.

• input space is huge: not exhaustively testable

Software examination paths

Software Testing Introduction

• One Approach: Statistical Tests

• What Else Makes a Test Case a Good Test Case?
Statistical Testing: Discussion

Ludewig and Lichter (2013) name the following objections against statistical testing:

1. In particular for interactive software, the primary requirement is often no failures are experienced by the “typical user.” Statistical testing (in general) may also cover a lot of “untypical user behaviours” unless (sophisticated) user-models are used.

2. Statistical testing needs a method to compute “soll-values” for the randomly chosen inputs. That is easy for requirement “does not crash,” but can be difficult in general.

3. There is a high risk for not finding point or small-range errors. If they live in their “natural habitat,” carefully crafted test cases would probably uncover them. Findings in the literature can at best be called inconclusive.

Where Do We Get The “Soll”-Values From?

Recall: A test case is a pair \((I, Soll)\) with proper expected (or “soll”) values.

1. In an ideal world, all “soll”-values are defined by the (formal) requirements specification and effectively pre-computable.

2. In this world, the formal requirements specification may only reflectively describe acceptable results without giving a procedure to compute the results.

3. There may not be a formal requirements specification, e.g.
   - “the game objects should be rendered properly”,
   - “the compiler must translate the program correctly”,
   - “the notification message should appear on a proper screen position”,
   - “the data must be available for at least 10 days”.
   - etc.

Then: need another instance to decide whether the observation is acceptable.

The testing community prefers to call any instance which decides whether results are acceptable a (test) oracle.

I’d prefer not to call automatic derivation of “soll”-values from a formal specification an “oracle”. . .

(Person or agency considered to provide wise and insightful [...] prophetic predictions or precognition of the future, inspired by the gods. “ says Wikipedia)
Testing:

- Coverage measures

- Violations of $\phi$ necessarily causes

- Glass-box testing

- One test case per feature

- Software is $\therefore$ there is no unreachable statement

Conclusions from Coverage Measures

- Term coverage

- Unreachable code

- Statement coverage
