7 (a) Basic Notions

7 (b) Dynamic testing

7 (c) Static Analysis

7 (d) Modelling
7 (a) Basic Notions

7 (b) Dynamic testing

7 (c) Static Analysis

7 (d) Modelling
No Additional Material

For this subsection no additional material has been added yet.
7 (a) Basic Notions

7 (b) Dynamic testing

7 (c) Static Analysis

7 (d) Modelling
Dynamic testing means that one operates the system under test.
Done by the execution of **test cases**, which investigates certain aspects of the system.
Each test set consists of
- a set of input **test data**
  - often called **test vector**.
- a specification of the expected output,
  - output is often called **output vector**.
- a statement of the function being tested.
In case of interactive programs, the test data will usually a sequence of inputs.
Basic Notions

- With each test cases one associates
  - **pre-conditions**
    - specify the state of the system before the test is executed,
  - **post-condition**
    - define the state the system must be in after the test.
- So tests will check whether if the test input vector fulfills the pre-condition, the test output vector fulfills the post-condition.
- The goal is to show that for any input fulfilling the pre-condition the output will fulfil the post-condition.
Some tests investigate the operation of the system under the condition that the **pre-conditions are not met**.

- Used in order to check what happens if the system deviates from its operation.
Basic Notions

- The **input space** of a system is the set of possible inputs.
  - If a system has $n$ inputs of a simple type like integer, floating point numbers, it has an **n-dimensional input space**.
Categories of Dynamic Testing

There are 3 main categories of dynamic testing:

- Functional testing,
- structural testing,
- random testing.
Functional Testing

- **Functional testing** is the testing of functions of the system as defined by its specification.
  - For each aspect of the operation tests are carried out.
  - However, tests might cover more than one function.
  - One has to make sure that all functions are covered by the tests.
- It is **black-box testing**, no details about the implementation are needed.
- Often a **test-matrix** is written, which associates each function with tests. See next slide.
  - Used in order to make sure that one has complete coverage of all functions.
### Example Test Matrix

<table>
<thead>
<tr>
<th>Test</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Structural Testing

- **Structural testing** looks at the internal structure of a system, and uses it into order to check the operation of individual components and their interactions.
  - In case of **hardware testing** uses test signals to investigate particular modules in the system.
  - In case of **software testing**, this involves tests in order to check certain routines or certain execution paths. Allows to investigate critical conditions.

- **Coverage-based testing** is structural testing with the goal of testing a large proportion of the system, by having tests for every branch or loop in the system.

- Structural testing is necessarily **white-box testing**.
Random Testing

- **Random testing** uses a test data which are randomly chosen from the input space.
  - Could be randomly sampled from the entire input space.
  - Could be sampled following some probability distribution.
    - The distribution might match the one expected for the operation.

- Aims at detecting fault conditions which are missed by more systematic techniques.
Dynamic Testing Techniques

- We list some of the techniques used.
- **Test cases based on equivalence partitioning.**
  - The input and outputs of the system/component to be tested is **partitioned** into sets of ranges which are **equivalent**, i.e. expected to be treated the same way.
  - Tests are performed to investigate **each partition**.
  - Both valid and invalid values are partitioned and tested.
  - E.g. for a function dealing with student marks, one might expect that
    - the ranges 40 – 49%, 50 – 59% etc. form valid partitions.
    - the ranges < 0%, > 100% form invalid partitions.
Dynamic Testing Techniques

- **Test cases based on boundary value analysis.**
  - Tests the performance of the system at the **boundaries** of equivalent partitions of inputs and outputs.
  - Again both valid and invalid values are partitioned and tested.
  - For instance, in the above example one might check for
    - valid boundary values like 50%, 49% etc.,
    - for invalid boundary values like −1%, 101%,
    - for valid values at the boundary to invalid values like 0%, 100%.
Dynamic Testing Techniques

- **State transition testing** identifies the different states of the component and system.
  - Then tests are performed in order to investigate
    - transitions between states,
    - events causing such transitions,
    - actions resulting from such transitions.

- **Probabilistic testing** determines the reliability of a system.
  - Attempts to measure failure rates over a given period of time, or failures on demand.
  - This testing is difficult to perform for critical systems, since there a very low failure rate is demanded, so probabilistic testing should return a failure rate of 0.
Dynamic Testing Techniques

- **Process simulation** is the simulation of the process or equipment to be controlled by the system.
  - Allows to reproduce lots of situations quickly and safely.

- **Error guessing** means that the test engineer predicts input conditions which are likely to cause problems.

- **Error seeding** means the insertion of errors into a system to see if they are detected by the testing procedures.
  - Is a test for the testing process.
  - May allow to predict the number of unfound errors.
Dynamic Testing Techniques

- **Timing and memory tests** investigate response time and memory consumption of a system.

- **Performance testing** tests that necessary levels of performance are reached.
  - E.g. that a certain number of operations per time unit are achieved.

- **Stress testing** tests the performance of a system under a very high workload.
  - Important for instance for the test of (web-, data base- and other) **servers**.
7 (a) Basic Notions

7 (b) Dynamic testing

7 (c) Static Analysis

7 (d) Modelling
(c) Static Analysis

- Static testing investigates a system **without operating it**.
- Techniques can be
  - performed **manually**, e.g. walkthroughs, inspections, use of checklists,
  - or using automated **static code analysis tools**
    - e.g. conformance tests for hardware, formal methods, data/information flow analysis, semantic analysis, complexity measurement, range checking.
Static Analysis

Static analysis aims at establishing properties of the software or software which are **true under all circumstances**.

- In contrast with **dynamic testing**, which can only test a **small subset** of the input set.
Static Analysis Techniques

- **A code walkthrough** means that an engineer leads colleagues through the design or implementation of software and convinces them of its correctness.

- **Design review** means peer review and systematic investigation of documents by a number of engineers.

- **Checklists** consists of a set of (usually very general) questions used in order to critically and systematically check certain aspects of a system.

- **Formal proofs** are used to show the correctness of some aspects of the design or implementation of a system.
Fagan inspections form a systematic audit of quality assurance documents in order to find errors and omissions.

Consists of **5 stages**:

- planning,
- preparation,
- inspection,
- rework,
- follow-up.
Static Analysis Techniques

- **Control flow analysis**
  - Analysis of software to detect poor and potentially incorrect program structure.
  - Looks for inaccessible code, infinite loops, poor or error-prone structural program elements.
  - Performed in SPARK Ada.
Static Analysis Techniques

- **Data flow analysis**
  - Analysis of the flow of data through a program.
  - Checks appropriateness of operations and comparison between actual and required data flow.
  - Checks
    - whether variables are initialised,
    - the input/output behaviour of variables,
    - the dependencies between variables.
  - Performed in SPARK Ada.
Static Analysis Techniques

- **Symbolic execution** uses algebraic variables instead of numeric inputs and computes the result of the program in the form of algebraic expressions.
  - Results of a program can be compared with those predicted by the specification.
  - Usually results too complicated to be analysed, need some form of user guidance.
  - Some tools (**semantic analysers**) perform automatic simplification of data.
  - Check of verification conditions in SPARK Ada together with the simplifier form an example of symbolic execution.
Metrics are measures for certain properties of the software.
- Measure for instance reliability and complexity.
- Tools perform the analysis of such metrics.
- Such tools measure for instance:
  - The graph theoretic complexity based on the complexity of the program graph.
  - Module accessibility, the number of ways a module can be accessed.
  - Complexity measures.
  - Number of entry and exit points per module
Static Analysis Techniques

- **Sneak circuit analysis.**
  - **Sneak currents** are latent conditions in a system, which cause it to malfunction under certain conditions.
  - Might be
    - physical paths,
    - timing irregularities,
    - ambiguous display messages,
    - and others.
  - **Sneak circuit analysis** aims at locating such weaknesses by looking at basic topological patterns within hardware and software.
7 (a) Basic Notions

7 (b) Dynamic testing

7 (c) Static Analysis

7 (d) Modelling
(d) Modelling

- Modelling used especially in the early phases of project development.
- Particularly important when producing the *specification* and the *top-level design*.
- Plays as well an important role later, especially during *system validation*.
Formal methods can be used to model a system.

Software prototyping/animation means that a software prototype is created which represents certain features of the specification.

- Used for the validation of the specification.
Performance modelling consists of the following steps:

- A model of the system processes and their interactions is constructed.
- Then the requirements of processor time and memory requirements for each function of the system are determined.
- Finally the total system demand is determined under average and worst-case conditions.
- This is used in order to guarantee that the system always satisfies the demand, including margins for safety.
Modelling Techniques

- **State transition diagrams** means that
  - the system is represented by finitely many discrete states;
  - with the transitions formed by the system, one obtains a finite state machine.
- the system can now be analysed and checked for **completeness**, **consistency**, **reachability**.
- **Model checking** is a technique based on state transition diagrams.
  - Used especially in hardware verification.
Process algebras and Petri-nets model a system in terms of various processes.

- Conditions like correctness, termination, deadlock-freedom can be examined using these techniques.
- Commonly used especially for concurrent systems, e.g.
  - railway interlocking systems,
  - networks,
  - verification of the Netscape web-browser.
Modelling Techniques

- **Data flow analysis** (see above) can be considered as well as a modelling technique.
- **Structure diagrams** represent the program structure by a structure chart, which is a tree representing the relationship between the program units.
- **Environmental modelling** means that one simulates the operating environment of a system in order to test it in an almost real environment.