Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Software Testing

Sibylle Schupp[1]

[1]Institute for Software Systems/Institut für Softwaresysteme
Hamburg University of Technology (TUHH)

Spring 2022

Lecture 2

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Outline

1 Criteria-based test design

2 Input space partitioning I

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Test design

- Main test activities
  - Design tests and turn them into executables
  - Run tests against program
  - Observe and evaluate results
- Test design
  - What are the requirements on the tests?
  - What are appropriate input values for running the tests?
  - Mathematical and technical issues
- Focus on criteria-based test design
  - Alternative: human-based test design

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Two views on testing

- Phase-oriented
  - Different test designs for different development phases
  - Unit, module, integration, system tests
- Structure- and criteria-driven
  - Same test design applicable for different phases
  - At the same time: models ("structures") within one phase differ. Also: different value selection and different test automation
  - Structures: input space, graph, logic expression, syntax
- We will follow the structure-driven view.

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Sources of structures

- Structures are extracted from different software artifacts
  - Graphs: UML use cases, finite state machines, source code, ...
  - Logic expressions: conditional in use cases, guards in FSMs, branches in source code, ...
- Structures are used to model software
  - Abstraction; focuses on particular aspects
- Similar to "model-based testing" (MBT) but not the same:
  - MBT uses models that *specify* the system.
  - Structures use models that *represent* the system. May use the MBT models, but not the other way around.

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Test coverage criteria

Definition

A test requirement (TR) is a specific element of a software artifact that a test set must satisfy. A coverage criterion is a set of rules that impose test requirements on a test set.

- Abstract definition ("artifacts" can be instantiated in different ways)
- Unifies the zoo of criteria

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Examples (structures)

- Input domain characterizations:
  $A : \{0, 1, > 1\}$, $B : \{\text{workday}, \text{weekend}\}$, $C : \{[1 \ldots 2^8 - 1], \geq 2^8\}$
  for parameters A, B, C
- Graphs:
  $G = (\{s_0, s_1, s_2\}, \{(s_0, s_1), (s_0, s_2), (s_1, s_2)\})$, pair of vertices and edges
- Logic expressions:
  `(not X and not Y) and A`
- Syntax:

```
if (x > y)
  z = x - y;
else
  z = 2*x;
```

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Example: jelly bean coverage

Source: Wikipedia

- Six flavors: lemon, pistachio, cantaloupe, pear, tangerine, apricot
- Four colors: yellow (lemon, apricot), green (pistachio), orange (cantaloupe, tangerine), white (pear)
- Two possible coverage criteria:
  - Test one jelly bean per flavor
  - Test one jelly bean per color
- Corresponding test requirements
  $TR_1 = \{$ lemon, pistachio, cantaloupe, pear, tangerine, apricot $\}$
  $TR_2 = \{$ lemon/apricot, pistachio, cantaloupe/tangerine, pear $\}$

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Coverage

Definition

Given a set of test requirements TR for a coverage criterion C, a test set T satisfies C iff for every test requirement tr $\in TR$ there is at least one test t in T such that t satisfies tr.

- Example (jelly beans):
  T_1 ={ lemon, lemon, lemon, pistachio, cantaloupe, cantaloupe, pear, tangerine, apricot, apricot, apricot, apricot}
  T_2 ={ lemon, pistachio, pistachio, pear, tangerine, tangerine, tangerine}
- Does test set T_1 satisfy the flavor criterion? The color criterion? What about test set T_2?

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Infeasible test requirements

Definition

Test requirements that cannot be satified are called <u>infeasible</u> test requirements.

- Common sources: dead code, inconsistent constraints
- Detecting infeasibility is undecidable for most criteria
- In practice, 100% coverage often not achievable

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Subsumption

Definition

A test criterion C_1 subsumes a test criterion C_2 iff every set of test cases that satisfies C_1 also satisfies C_2.

Example

- The flavor criterion subsumes the color criterion.
- You might know from your software engineering class:
  The branch criterion subsumes the node criterion.

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

In-class exercise

Again, the definition

Definition

A test criterion C_1 subsumes a test criterion C_2 iff every set of test cases that satisfies C_1 also satisfies C_2.

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Coverage level

Definition

Given a set of test requirements TR and a test set T, the coverage level of T is the ratio of the number of test requirements satisfied by T to the size of TR.

Example:

- $T_2 = \{$ lemon, pistachio, pistachio, pear, tangerine, tangerine, tangerine$\}$ satisfies 4 of 6 test requirements of the flavor criterion.

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Work flow (test criteria)

Test criteria can be used in two ways.

1. Generate test values so that they satisfy the criterion directly.
   - Generator-based approach
   - Hard to achieve with automated tools
2. Generate test values, measure coverage afterwards
   - Recognizer-based approach
   - What to do if coverage not 100%?

Coverage-analysis tools

- Both the generator- and recognizer-based approach are undecidable for most criteria.
- But recognition is easier and in practice "good" coverage is often possible.

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Outline

1. Criteria-based test design
2. Input space partitioning I

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

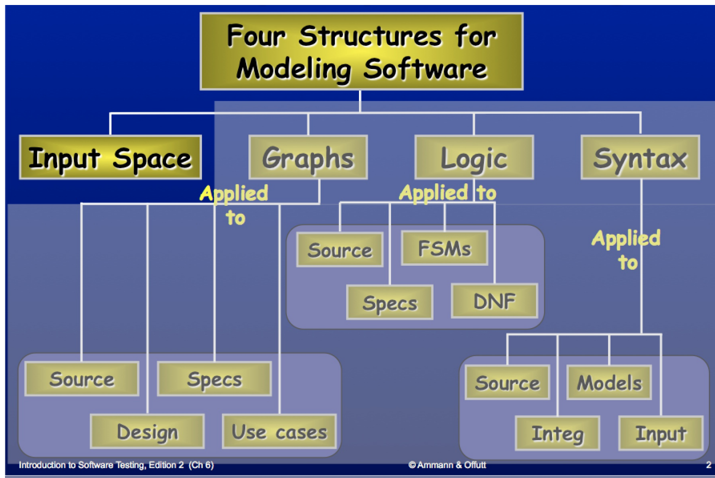
Outline

1 Criteria-based test design

2 Input space partitioning I
- Introduction
- Input domain modeling

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Input space coverage



Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Input domains

Definition

The input domain ("space") of a program is the multi-set of all possible values of its input parameters.

- Input parameters
  - Run-time parameters (user input)
  - Method parameters
  - Global variables, data from files
  - Internal object state
- Input domains are large.
- Testing by partitioning the input domain in regions
  - Assumption: all regions are equally good
  - For test sets: pick one value from each region

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Benefits of input-space partioning (ISP)

- Applicable at several levels of testing
  - Unit, integration, system
- Applicable by hand
- No implementation knowledge needed

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Partitioning domains

Definition

Let D be a domain. A partition scheme q of D defines a set of blocks $B_q = b_1, \ldots, b_Q$ such that

$$\bigcup_{b \in B_q} b = D \qquad b_i \cap b_j = \emptyset \; \forall i \neq j, b_i, b_j \in B_q$$

- Note: completeness and disjointness
- “Blocks” are called “equivalence classes‘‘ in mathematics.

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

In-class exercise

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Characteristics

Partitioning is based on characteristics of a program.

- Examples of characteristics
  - Values: input A is null (vs. not-null)
  - Predicates: input array is ascendingly sorted (vs. unsorted, descendingly sorted)
  - Relations: minimal separation of two objects (vs. unknown, arbitrary)
  - Events (external): requestA (vs. request B, failure)
- Sources of characteristics
  - Program parameters, specification, environment
- Steps
  - Find characteristics
  - Partition each characteristic
  - Choose tests by combining values from characteristics

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Example (partitioning)

- Consider the characteristic “order of array A” and the following definition

$$
\begin{aligned}
b_1 &= \text{sorted ascendingly} \\
b_2 &= \text{sorted descendingly} \\
b_3 &= \text{arbitrary, unsorted}
\end{aligned}
$$

- Careful: not a partition!
- Disjointness violated for arrays of length 1
- Source of the problem: characteristic addresses more than one property

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Example (partitioning), cont'd

- Consider the two characteristics:
  C_1 = "array A is sorted ascendingly"
  C_2 = "array A is sorted descendingly"
- C_1, C_2 can be partitioned in two classes each:

Characteristic	Partitions	
C_1	C_1 is true	C_1 is false
C_2	C_2 is true	C_2 is false

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

In-class exercise

```
public class GenericStack<E> {
  private java.util.ArrayList<E> list
      = new java.util.ArrayList<E>();

  public void push(E o) {
    list.add(o);
  }
  public E pop() {
    E o = list.get(getSize() - 1);
    list.remove(getSize() - 1);
    return o;
  }
  public boolean isEmpty() {
    return list.isEmpty();
  }
  // ...
}
```

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Outline

1 Criteria-based test design

2 Input space partitioning I
- Introduction
- Input domain modeling

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Input domain modeling: overview

Finding partitions in a systematic way requires input modeling.

- Input domain modeling employs a 5-step process.
- Step 3 yields the actual input domain model (IDM), other steps prepare model resp. use model for test-value generation.
  - Step 1: Identify testable functions
  - Step 2: Identify all parameters
  - Step 4: Choose combinations of values according to test criterion
  - Step 5: Refine combinations of blocks into test inputs

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Modeling the input domain: steps 1 and 2

- Step 1: Identify testable functions
  - A method has one testable function (clear).
  - An API has a testable function per public method, but methods might share characteristics.
  - A UML use case has one testable function.
  - An integrated system could have one testable function per component or functionality.
- Step 2: Identify all parameters
  - Parameters, internal state variables, global variables (including files, databases)
  - Depends on step 1, mostly straightforward

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Modeling the input domain: steps 3-5

- Step 3: Model the input domain
  - The scope of the domain is defined by parameters.
  - The structure of the domain is defined by characteristics.
  - Each characteristic is partitioned into blocks.
  - Each block represents a set of values.
- Step 4: Choose combinations of values according to test criterion
  - A test input is a tuple of values, one per parameter.
  - Values are selected per block (and characteristics).
  - Number of possible combinations (of blocks of the characteristics of the various parameters) is typically infeasible.
  - Coverage criteria guide the choice.
- Step 5: Refine combinations of blocks into test inputs
  - Choose appropriate values from each block.

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Two approaches to input-domain modeling

1. Interface-based approach
   - Characteristics emerge directly from input parameters
   - Simple, partially automated
   - Ignores relations between parameters as well as other information (domain, semantics)
2. Functionality-based approach
   - Characteristics emerge from a behavioral view on the program.
   - Requires more design effort, cannot be automated
   - Might result in better tests or fewer tests that are as effective

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Example (interface-based approach)

http://www.cs.gmu.edu/~offutt/softwaretest/edition2/java/Triangle.java

```
public enum Triangle { Scalene, Isosceles, Equilateral, Invalid }
public static Triangle triang (int Side1, int Side2, int Side3)
  // Side1, Side2, and Side3 represent the lengths of the sides
  // of a triangle.
  // Returns the appropriate enum value.
```

- IDM for each parameter is the same.
- Sensible characteristics?
- Relation of a side to 0

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Example (functionality-based approach)

http://www.cs.gmu.edu/~offutt/softwaretest/edition2/java/Triangle.java

```
public enum Triangle { Scalene, Isosceles, Equilateral, Invalid }
public static Triangle triang (int Side1, int Side2, int Side3)
  // Side1, Side2, and Side3 represent the lengths of the sides
  // of a triangle.
  // Returns the appropriate enum value.
```

- Requirement/domains/semantics: the parameters form a triangle
- IDM can combine all parameters
- Sensible characteristics?
- Type of the triangle

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Finding characteristics

- In the functionality-based approach: essentially a creative task
- Where to look for characteristics?
  - Preconditions, postconditions
  - Special values (null, 0, . . .)
  - Invariants (incl. relationships between parameters)
- Best practice
  - More characteristics with fewer blocks is better.

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Another example

http://www.cs.gmu.edu/ offutt/softwaretest/edition2/java/

```
public boolean findElement (List list, Object element)
// Effects:
//    if list or element is null throw NullPointerException
//    else return true if element is in the list, false otherwise
```

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Another example (interface-based approach)

```
public boolean findElement (List list, Object element)
// Effects:
//   if list or element is null throw NullPointerException
//   else return true if element is in the list, false otherwise
```

- Two parameters: `list`, `element`
- Characteristics and partitions:

Characteristics	b_1	b_2
list is null	true	false
list is empty	true	false
element is null	true	false

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

Another example (functionality-based approach)

http://www.cs.gmu.edu/ offutt/softwaretest/edition2/java/Triangle.java

```
public boolean findElement (List list, Object element)
// Effects:
//   if list or element is null throw NullPointerException
//   else return true if element is in the list, false otherwise
```

- Two parameters: `list`, `element`
- Characteristics and partitions:

Characteristics	b_1	b_2	b_3
number of occurrences of element in list	0	1	> 1
element occurs first in list	true	false	
element occurs last in list	true	false	

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

In-class exercise

```
   public static int search (List list, Object element)
// Effects: if list or element is null throw NullPointerException
//   else if element is in the list, return an index
//        of element in the list;
//   else return -1
//   for example, search ([3,3,1], 3) = either 0 or 1
//                search ([1,7,5], 2) = -1
```

Slides and audiovisual material for private use only. Download or further distribution prohibited.]

References

- AO, Ch. 5.1 and 5.2
- AO, Ch. 6.1