Software Testing, Quality Assurance and Maintenance Winter 2015

Lecture 29 — March 18, 2015

Patrick Lam *version 0*

Input Space Partitioning

Recall the most basic form of testing: feed inputs to the program and see how it behaves. Of course, we know that we can't feed all inputs to the program, so we only test a representative set of inputs. *Input space partitioning* makes this idea more formal: test one input from each partition.

Desirable properties for partitions:

Partitions are usually based on *characteristics* of the input (or environment),e.g.

Here is an example of a bad partition on objects which may implement interfaces `List` and `Serializable`:

- objects which implement `List`
- objects which implement `Serializable`
- neither

What about objects that are serializable lists? Instead, use the characteristics separately:

- `List` / not `List`
- `Serializable` / not `Serializable`

Disjoint partitions make it easier to ensure that you indeed cover all partitions.

Input Domain Modelling

We will describe input domains at the unit level, although it applies equally well at the integration level. Three steps:

- find units/functions to test;

- identify parameters of each unit;
- come up with the model.

What Units Should We Test? First, we determine what to test.

- In our sample programs below, there's only one testable function.
- For classes in general, could test all public methods (grouping them together as needed).
- Use cases can also give you hints about how to group both methods and inputs, or about which methods might be most important.

What are the Parameters? Next, we figure out what inputs the units might take. Some possibilities:

Create the Model! Finally, we can group the inputs by finding characteristics, and creating partitions and blocks, from the values. Here's an example of the input domain of two digit numbers:

$$\begin{array}{cccccccccc}
00 & 01 & 02 & 03 & 04 & 05 & 06 & 07 & 08 & 09 \\
10 & 11 & 12 & & 14 & & 16 & & 18 & \\
20 & 21 & 22 & & & \cdots & & & & \\
30 & 31 & 32 & & & \cdots & & & & \\
\vdots & & & & & \cdots & & & & \vdots \\
90 & & & & & \cdots & & & & 99
\end{array}$$

Possible characteristics and partitions:

Definitions

Here are some definitions.

- Characteristics: how we distinguish values;
- Partition: a way of splitting values into a set of blocks ($p : I \to \{0, 1, \cdots, n\}$)
- Block: a set of values that are alike with respect to a characteristic ($p^{-1}(k)$)

Each input value belongs to one block per characteristic. (We'll talk about combinations later.)

Input Domain Models

Coming up with IDMs requires creativity and analysis. Two general approaches:

- interface-based, using the input space directly; or
- functionality-based, using a functional or behavioural view of the program.

Interface-Based Input Domain Modelling. Consider each parameter in isolation. For example:

```
  public boolean containsElement(List list, Object element);
```

Possible interface-based characteristics:

- `list` is null; block 1: `true`, block 2: `false`
- `list` is empty; block 1: `true`, block 2: `false`

Notes:

- (+) "surprisingly good", says the book;
- (+) easy to identify characteristics (but book doesn't provide cookbook);
- (+) easy to translate to test cases;
- (-) doesn't use domain knowledge, e.g. relationships between parameters.