Reading Technique for Equivalence Partition Testing

For each requirement, generate a test or set of test cases that allow you to ensure that an implementation of the system satisfies the requirement. Follow the procedure below to generate the test cases, using the questions provided to identify faults in the requirements:

1) For each requirement, read it through once and record the number and page on the form provided, along with the inputs to the requirement.

   Q1.1 Does the requirement make sense from what you know about the application or from what is specified in the general description?

   Q1.2 Do you have all the information necessary to identify the inputs to the requirement? Based on the general requirements and your domain knowledge, are these inputs correct for this requirement?

   Q1.3 Have any of the necessary inputs been omitted?

   Q1.4 Are any inputs specified which are not needed for this requirement?

   Q1.5 Is this requirement in the appropriate section of the document?

   a) For each input, divide the input domain into sets of data (called *equivalence sets*), where all of the values in each set will cause the system to behave similarly. Determine the equivalence sets for a particular input by understanding the sets of conditions that effect the behavior of the requirement. You may find it helpful to keep the following guidelines in mind when creating equivalence classes:

      - If an input condition specifies a range, at least one valid (the set of values in the range) and two invalid equivalence sets (the set of values less than the lowest extreme of the range, and the set of values greater than the largest extreme) are defined.
      - If an input condition specifies a member of a set, then at least one valid (the set itself) and one invalid equivalence sets (the complement of the valid set) are defined.
      - If an input condition requires a specific value, then one valid (the set containing the value itself) and two invalid equivalence sets (the set of values less than, and the set greater than, the value) are defined.

      Each equivalence set should be recorded in the spaces provided on the form under the appropriate input.

      Q2.1 Do you have enough information to construct the equivalence sets for each input? Can you specify the boundaries of the sets at an appropriate level of detail?

      Q2.2 According to the information in the requirements, are the sets constructed so that no value appears in more than one set?

      Q2.3 Do the requirements state that a particular value should appear in more than one equivalence set (that is, do they specify more than one type of response for the same value)? Do the requirements specify that a value should appear in the wrong equivalence set?

   b) For each equivalence set write test cases, and record them beneath the associated equivalence set on the form. Select typical test cases as well as values at and near the edges of the sets. For example, if the requirement expects input values in the range 0 to 100, then the test cases selected might be: 0, 1, 56, 99, 100. Finally, for each equivalence set record the behavior which is expected to result (that is, how do you expect the system to respond to the test cases you just made up?).

      Q3.1 Do you have enough information to create test cases for each equivalence set?

Q3.2 **Are there other interpretations of this requirement that the implementor might make based upon the description given? Will this affect the tests you generate?**

Q3.3 **Is there another requirement for which you would generate a similar test case but would get a contradictory result?**

Q3.4 **Can you be sure that the tests generated will yield the correct values in the correct units? Is the resulting behavior specified appropriately?**

Acknowledgements

These guidelines are based on information found in:

Jacobson, Ivar, *et al*. (1992) *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Publishing Company.

Stacey, Deborah A. *Software Testing Techniques*. 27-320 Software Engineering Lecture Notes, University of Guelph.
http://hebb.cis.uoguelph.ca/~deb/27320/testing/testing.html#1.4.2