The primary prerequisite of functional testing is the presence of an oracle. The basic kinds of oracles recommended by Howden for functional testing are input/output oracles, trace oracles and interface oracles. An input/output oracle is capable of determining that for a given input state, the corresponding output state is produced. A trace oracle confirms that the predicted, i.e. specified path through the program corresponds to the actual path traversed. An interface oracle verifies the internal data exchange between module, i.e. that the data outputs of the sending module correspond to the specified data inputs of the receiving module. All of these oracles can be reduced to a single state oracle which confirms that the actual states of a program match the specified states, inclusive input domain, output domain, internal data domain and location of control flow.

In testing restricted real-time functions it should be the goal to specify all possible states. In this case the finite state machine is the oracle. If, however, the number of possible states prohibits testing them all, it is up to the human oracle who specifies the function to define the most relevant states dependent upon the context of the environment in which the system will operate. The test will then be only as reliable as the specification of the environment. This is where testing reaches its limit as a precise technology and becomes an art.

This brings up two questions. The first question to be answered is:

Have all relevant states been specified?

The second question to be answered is:

Have all specified states been tested?

The question of whether all relevant states have been specified is context dependent and therefore nondeterminable. At best there can only be a statistical approximation based on the past usage profile of the target system. Formally, there can be a consistency check between the value domains specified for the individual data elements and the value domains specified, for the arguments to the functions. Those values outside of these specified input domain states can be considered nonspecified states. There is then a ratio of specified states to total states derived from the value domains of the data which gives the static functional coverage.

The overall functional coverage is the ratio of states actually tested to all states derived from the value domains of the data, whereby the concept of equivalence classes should be used to reduce the number of possible states.

In the end, there are at least three levels of state coverage. These are:
- generation of each input state triggering a specified action
- generation of each input state specified by the human oracle
- generation of each possible input state.

Functional coverage based on the specification is therefore proposed as an alternative to structural and data coverage. However, to make functional coverage measurable, it is necessary to define input states based on a prediction of the usage profile. One way of doing this is to use a special assertion language which pairs preconditions and postconditions together with operational modes and time constraints. Generating all specified preconditions and validating all specified postconditions for all operational modes under all time constraints is equivalent to functional coverage, provided that all relevant input states are covered by the specification. To determine this, a knowledge base of the application is required. This in turn, implies a significant amount of statistical data on the target system performance upon which a predicted usage profile can be based. This is the approach presently being investigated in testing automotive software by the TRUST project. It is hoped that it will lead to a more precise means of measuring test coverage at the functional level.