Testing Expert Systems using Conventional Techniques

W. T. Tsai  Shekhar Kirani  I. A. Zualkernan*  
{ tsai, shekar, zualkern }@cs.umn.edu  
University of Minnesota, Minneapolis, MN 55455

Abstract

Expert systems are being used in production environments. Testing of an expert system is a difficult problem. In this paper we comment on the difficulties associated with testing of an expert system. We propose pragmatic testing methods of conventional software engineering as a solution to these problems. The application of these techniques is illustrated through an extended example.

1 Introduction

Expert systems are being developed commercially to solve non-traditional problems in such areas as auditing, fault-diagnosis, and computer configuration. As expert systems move out from research laboratories to commercial production environments, establishing reliability and robustness have taken an increasing importance [5, 12, 13, 15, 1, 20, 6, 17].

Expert systems have been traditionally developed under the exploratory prototyping paradigm (see figure 1)[22], where there are no intermediate deliverables and no verification steps between the phases. It is believed in AI that software engineering techniques may not be applicable to expert systems [15] and specialized testing techniques like completeness and consistency checking must be developed for validating expert systems. In this paper we demonstrate that many conventional software testing techniques are indeed applicable to expert system testing.

In this paper we follow life-cycle approach for testing expert systems. Software development life cycle can be broadly classified into 4 stages like specification (problem and solution), design, coding and testing. Testing (generally referred to as verification and validation [11, 14, 3]) establishes a binary relationship between two by-products of the software development process. The objective of testing is to establish the correspondence between a problem specification [25] and an implementation of an expert system based on a solution specification. In this paper we adopt the IEEE standard terminology of error, fault and failure.

*This research was supported in part by National Science Foundation grant number IRI-8715623.

0730-3157/92 $3.00 © 1992 IEEE
Operational Criteria for Goal Satisfaction: Criteria of percent accuracy and precision will be used as two criteria of reliability to judge goal satisfaction. Percent accuracy is defined as the mean size of the answer set. A solution with a minimum percent accuracy of 40% and a maximum precision of 3 is acceptable. The explanation produced should have a 1-1 correspondence with how a problem is actually solved.

Input: The input consists of a set of measurement deviation on a bipolar transistor. The measurements to be included are sheet resistivities of two layers (SRE and SRB), junction capacitances (CEB and CBC), current gain ($h_{fe}$), currents ($I_b$ and $I_c$), punch through voltage ($V_{pt}$), and emitter-base breakdown voltage (BVEB). Only physically possible combinations of these inputs are acceptable.

Output: The output should consist of a determination of physical anomalies. A physical anomaly is a combination of deviations in physical parameters of the device. The physical parameters to be included are widths ($W_e$, $W_b$, $W_c$) and concentrations ($N_e$, $N_b$, $N_c$) of the emitter, base, and collector layers.

Solution specification

Solution specification is represented in the form of three methods shown in figure 2. Each node in a method represents an operation on a hypothesis or data. For example, method 1 indicates that an observation of a measurement deviation in the sheet resistivity of the base (i.e., $SRB_1$ or $SRB_2$) leads to proposing two alternative hypotheses that either the concentration of the base in high or low (i.e., $N_b_1$ or $N_b_2$) or the width of base is low or high (i.e., $W_b_1$ or $W_b_2$). The emitter-base capacitance is used as evidence to decide between the two competing hypotheses. As method 1 shows, if $CEB$ has not changed (i.e., $CEB_0$), then the hypotheses about changes in the concentration of the base are rejected, which leaves base width as the only candidate. The hypotheses about a change in the base width is further confirmed by ensuring that the punch through voltage has moved in the appropriate direction of $V_{pt}$, which leads to the acceptance of a change in the base width as the answer or the product anomaly that led to the observed measurement deviations.

3 Why is testing hard?

Testing in general is a labor-intensive and fault-prone process [11]. Testing generally consists of the three distinct steps: 1. Establish the criteria for testing. 2. Generate an appropriate set of test cases, where each test case consists of test input and expected output. 3. Apply a testing method and evaluate its outcomes. The difficulties associated with each step of testing described above arise due to following reasons:

1. Testing criteria can be unclear.
2. Input and output spaces for selection of test cases are huge.
3. Generation of test case input and expected output is difficult.
4. Generation of expected path for white-box testing is expensive.

4 Pragmatic testing: making testing manageable

The objective of pragmatic testing is to reduce the number of test cases and minimize the cost of testing without sacrificing the quality of testing. In this section, we use expert system MAPS to demonstrate how

---

Figure 2: Method 1, 2 and 3 for solution specification of MAPS

---

2The solution specification for the problem specification described above was derived using protocol analysis techniques[4].
some of the pragmatic testing methods can be used for testing of expert systems.

4.1 Black-box testing techniques

4.1.1 Random Testing

Random testing [11] generates test inputs randomly from input space. For random testing to be effective it must be easy to generate expected outputs [9] and fault-prone regions (input regions that result in failures) should be large enough.

In MAPS, the oracle (simulator) is used to generate expected output for randomly selected test input. Test inputs for random testing are generated from both possible and legal input space. Test inputs are randomly sampled from legal input space consisting of 123 legal measurement deviations. Number of test inputs sampled are varied from 10% to 100% of the size of the input space. The expected output of each legal test input is generated by running the oracle (a VLSI process simulator).

Figure 3: Legal input and output space for the example expert system (MAPS).

Figure 4: Accuracy and precision of answer for random testing on possible input space

In MAPS, the class of measurement deviations related to Ib (base current) and SRE (emitter resistance) are significant from a domain perspective. Hence Ib and SRE can be used for partitioning the input space into four partitions:

1. Ib(t1) and SRE(f1) with all other parameters at any value.
2. Ib(t1) and SRE(normal) with all other parameters at any value.
3. Ib(normal) and SRE(t1) with all other parameters at any value.
4. Ib(normal) and SRE(normal) with all other parameters at any value.

In this example, the test cases are generated by randomly sampling from each partition. Number of test inputs in each partition generated are varied from 10% to 100% of the size of the partition. The expected precision for MAPS. However, as figure 5 shows random sampling from legal input space generates precise estimate for both accuracy and precision for a small portion (20%) of the space. For a large-scale expert system, random sampling of even a small percentage of the input space can be prohibitive and hence it is necessary to decompose a large expert system.

4.1.2 Partition testing (input based)

As opposed to random testing, partition testing tries to reduce the number of test cases required by partitioning the input space into classes and sampling test cases from each class. The success of partition testing depends on presence of the problem specification and other by-products generated within the development paradigm. In general, however, the partitioning criteria are not obvious. Partitions can be based either on input or output space. In this paper we demonstrate domain based partitioning of the input space for the example expert system.

Figure 5: Accuracy and precision of answer for random testing on legal input space

Figure 6: Accuracy and precision of partition testing (domain) on possible input space

Figure 7: Accuracy and precision for partition testing (domain) on legal input space

For MAPS, the class of measurement deviations related to Ib (base current) and SRE (emitter resistance) are significant from a domain perspective. Hence Ib and SRE can be used for partitioning the input space into four partitions: 1. Ib(t1) and SRE(f1) with all other parameters at any value. 2. Ib(t1) and SRE(normal) with all other parameters at any value. 3. Ib(normal) and SRE(t1) with all other parameters at any value. 4. Ib(normal) and SRE(normal) with all other parameters at any value.

In this example, the test cases are generated by randomly sampling from each partition. Number of test inputs in each partition generated are varied from 10% to 100% of the size of the partition. The expected
output is generated by the oracle as in the case of random testing.

Figure 6 shows that input partition testing is worse than random testing for estimating both the accuracy and precision. Figure 7 shows similar results for sampling based on legal input space. Table 1 shows the summary of results within each partition. It is interesting to note that most failures are concentrated in one partition (corresponding to $I_7(11)$ and $SRE(11)$). The results of this domain based partitioning gives a worst estimate than random testing. However, the partition testing helps in pointing out potential failure-prone partitions of the input space. Concentration on these partitions may reduce cost of testing.

### 4.2 White-box testing techniques

White-box testing exploits the internal structure of a program in addition to input and expected output. White-box testing methods include path based partition testing, cause-effect graph testing, white-box dynamic flow testing, data-flow dynamic testing and ablation testing. We demonstrate the application of some of these test methods below.

#### 4.2.1 Partition testing (path based)

Path based partition testing bases the partition on the path space of a program. This partitioning can be carried out with the help of an explicit solution specification.

In MAPS, paths in the solution specification are used to partition the input and output space. The path through methods 1 and 2 (figure 2) defines 9 partitions of the input space as shown in table 2. Partition 9 is special partition that contains all measurement deviations not covered by any paths in the methods 1 and 2. The possible inputs and expected outputs for each partition are generated from solution specification (see figure 2). The legal test inputs for each partition is determined by using the oracle.

This partition strategy shows that most failures are concentrated in 3, 4, 5 and 6 partitions for possible input space. However, none of these partitions are legal. The performance of the program is overestimated (100% accuracy) in the remaining legal partitions. This means that the system gives an excellent performance on test cases generated using solution specification. Results from applying path based partition testing to MAPS shows that path based partition testing is severely limited by the solution specification. Out of total of 123 legal test inputs only 9 can be tested. This testing also tends to give a false positive impression of the quality of the expert system.

#### 4.2.2 Cause-effect graph testing

Cause-effect graph testing uses solution specification and the implemented solution to generate test cases in an exhaustive fashion. The exhaustive nature of this testing makes it particularly effective for detecting unanticipated side effects [11].

For our example, solution specification is used to identify causes and corresponding effects for the method 1 and method 2 (see figure 2). The causes are:

1. SRE, 2. SRE, 3. CEB normal, 4. $V_{in}$, 5. $V_{out}$, 6. CEB, 7. CEB, 8. BVE, 9. BVE, 10. SRE, 11. SRE, 12. SRE is 0, 13. $I_{in}$ and 14. $I_{out}$.

The effects are:

- E-1: $W_1$ with confidence level equal to ACCEPT.
- E-2: $W_2$ with confidence level equal to ACCEPT.
- E-3: $N_1$ with confidence level equal to ACCEPT.
- E-4: $N_2$ with confidence level equal to ACCEPT.
- E-5: $N_3$ with confidence level equal to ACCEPT.
- E-6: $N_4$ with confidence level equal to ACCEPT.
- E-7: $W_2$ with confidence level equal to CONFIRM.
- E-8: $W_3$ with confidence level equal to CONFIRM.

Figure 8 contains a cause effect graph for MAPS expert system derived from solution specification in conjunction with the system using [11]. The causes combine together to generate intermediate effects. The numbers shown for each cause and intermediate effects in figure 8 correspond to rule numbers in knowledge base.

MAPS was tested on 10 test cases derived from the figure 8. It was successful for all test cases except

<table>
<thead>
<tr>
<th>Partition</th>
<th>Possible Partition</th>
<th>Legal Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_7(11)$, $SRE(11)$</td>
<td>324 32 183 69 6 21</td>
<td>1</td>
</tr>
<tr>
<td>$I_7(11)$, $SRE(0)$</td>
<td>162 0 9 9 0 0</td>
<td>1</td>
</tr>
<tr>
<td>$I_7(0)$, $SRE(11)$</td>
<td>162 8 63 30 0 0</td>
<td>1</td>
</tr>
<tr>
<td>$I_7(0)$, $SRE(0)$</td>
<td>81 0 3 15 0 0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Partition size and failures for input partition testing. I - Partition size, II - Failures at 20% and III - Total failures

<table>
<thead>
<tr>
<th>Possible Partition</th>
<th>Legal Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Path based partitioning of input space using solution specification. (A: Partition number, J: High, L: Low, T: Change, O: Normal, +: Any value, -: Not known)
two test cases. A study of solution specification and
the knowledge base revealed that these failures were
caused by two faults in the solution specification. Lack
of side effects in cause-effect graph shows that the
solution specification is well defined. MAPS exam-
ple demonstrates that the cause-effect testing helps in
pointing out discrepancies between the expert system
and the solution specification in a systematic manner.
This type of testing can also be used to demonstrate
the quality of the solution specification. However, for
a large expert system the cost of cause-effect testing
can be enormous.

4.2.3 White-box dynamic flow testing

White-box dynamic flow testing not only checks for
input and expected output but also ensures that an ap-
propriate path is taken by the program. The expected
path is generally generated through a solution specifi-
cation. The path is compared to the path followed by
an expert system in arriving at an outcome.

In our example, methods 1 and 2 (solution specifi-
cation) were used to generate four test cases as shown
in figure 9. Each test case consists of an input, the
expected output and the associated paths.

There are a total of 40 legal paths in the solution
specification. Using solution specification one can de-
rive only 40 out of 123 legal test paths (corresponding
to 123 legal inputs). The program was successful on a
sample of 4 paths from these 40 paths. This testing is
effective for checking whether the program faithfully
follows its solution specification. A disadvantage of this
technique is the effort is proportional to the number of
paths in the solution specification and solution speci-
ification generates only a small portion of the path
space.

4.2.4 Data-flow dynamic testing

Data-flow dynamic testing tests for path incom-
pleteness, redundancies and ambiguities in an imple-
mentation. In a rule-based expert system, for exam-
ple, there can be several rules that are executed be-
tween define and use (DU) of a data value. Data-flow
testing tests all paths between such a DU pair.

In MAPS the DU pair exists between two design
modules like Propose-Confirm, and Confirm-Accept.
Consider the three rules in figure 10. The data item
\(< W_1 \uparrow, \text{PROPOSE} >\) is defined in rule 29 and used in
rule 28. Rule 28 further defines \(< W_1 \uparrow, \text{CONFIRM} >\)
and is used in rule 27. To test a DU pair \(< W_1 \uparrow, \text{PROPOSE} >,\text{ rule-29, rule-28}\) \[3, 14\] a test input is
generated such that rules 29 and 28 are enabled.
Because these rules use other data values such as
\('SRB \uparrow\)', and \('CEB 0\)', the test input is generated with
SRB and CEB set to 1 and 0 (no change) values re-
spectively. The expected output for this DU pair is
generated using the oracle.

Figure 10: Sample rules used in the example expert
system (MAPS)

Using reverse engineering techniques [26], it was de-
timated that the knowledge base of program has a total of 40 DU pairs. Out of these, 4 were randomly selected. The program was successful on all the 4 test cases. For effective data-flow testing however, due to the complexity of the implementation, additional tools such as reverse engineering may be required. The cost of this technique is proportional to the number of DU-pair paths.

5 Conclusion

In this paper we have shown why testing an expert system is difficult. We have shown the feasibility of random testing, partition testing (input and path based), cause-effect graph testing, white-box dynamic flow testing and data-flow testing. In addition this paper opens up many research issues such as: 1. Demonstration of other testing methods such as mutation testing[17], 2. Comparative evaluation of testing methods[17], 3. Type and degree of formality of an appropriate specification techniques, 4. Generation of expected outputs different from experts or archived test cases such as dual or N-fold development[10] and 5. Software reliability models for expert systems.

References