Voting System Standards

This document is part of Agenda Document Number 01-62 on the agenda for consideration at the December 13, 2001, meeting of the Federal Election Commission.

Volume II, Section 5
Table of Contents

5 Software Testing

5.1 Introduction

This section contains a description of the testing to be performed by the ITA to confirm the proper functioning of the software components of a voting system submitted for qualification testing. It describes the scope and basis for software testing, the initial review of documentation to support software testing, and the review of the voting system source code.

Further testing of the voting system software is addressed in the following sections:

a. Volume II, Section 3, for specific tests of voting system functionality, and

b. Volume II, Section 6, for testing voting system security and for testing the operation of the voting system software together with other voting system components.

5.2 Scope and Basis of Software Testing

ITAs shall design and perform procedures that test the voting system software requirements identified in Volume I. All software components designed or modified for election use shall be tested in accordance with the applicable procedures contained in this section.

Unmodified, general purpose COTS non-voting software (e.g., operating systems, programming language compilers, data base management systems, and Web browsers) is not subject to the detailed examinations specified in this section. However, the ITA shall examine such software to confirm the specific version of software being used against the design specification to confirm that the software has not been modified. Portions of COTS software that have been modified by the vendor in any manner are subject to review.

Compatibility of the voting system software components or subsystems with one another, and with other components of the voting system environment, shall be determined through functional tests integrating the voting system software with the remainder of the system.

The specific procedures to be used shall be identified in the Qualification Test Plan prepared by the ITA. These procedures may replicate testing performed by the vendor and documented in the vendor's TDP, but shall not rely on vendor testing as a substitute for software testing performed by the ITA.

Recognizing variations in system design and the technologies employed by different vendors, the ITAs shall design test procedures that account for these variations.

5.3 Initial Review of Documentation

Prior to initiating the software review, the ITA shall verify that the documentation submitted by the vendor in the TDP is sufficient to enable:

a. Review of the source code; and

b. Design and conducting of tests at every level of the software structure to verify that the software meets the vendor's design specifications and the requirements of the performance standards.

5.4 Source Code Review

The ITA shall compare the source code to the vendor's software design documentation to ascertain how completely the software conforms to the vendor's specifications. Source code inspection will involve an assessment of the extent to which the code adheres to the requirements in Volume I, Section 4. The following checklist contains the types of questions the ITA will ask during the source code review:

a. Completeness:

   1) Is the code a complete and precise implementation of the design as documented in the software design documentation?

   2) Was the code integrated and debugged to satisfy the design specified in the software design documentation?

   3) Does the code create the required databases, including the appropriate initial data?

4) Are there any unreferenced or undefined variables, constants, or data types?

b. Consistency:

1) Is the code logically consistent with the software design documentation?

2) Are the same format, invocation convention, and structure used throughout?

c. Correctness:

1) Does the code conform to specified standards?

2) Are all variables properly specified and used?

3) Are all comments accurate?

4) Are all programs invoked with the correct number of parameters?

d. Modifiability:

1) Does the code refer to constants symbolically to facilitate change?

2) Are cross-references or data dictionaries included to show variable and constant access by the program?

3) Does code consist of programs with only one entry point and one exit point? (exception is with fatal error handling)

4) Does code reference labels or other symbolic constants rather than addresses?

e. Predictability:

1) Is the code written in a language with well-defined syntax and semantics:

2) Was the use of self-modifying code avoided?

3) Does the code avoid relying on defaults provided by the programming language?

4) Is the code free of unintended infinite loops?

5) Does the code avoid recursion?

f. Robustness - Does the code protect against detectable runtime errors (e.g., range array index values, division by zero, out of range variable values, and stack overflow)?

g. Structuredness:

1) Is each function of the program recognizable as a block of code?

2) Do loops only have one entrance?

h. Traceability:

1) Does the code identify each program uniquely?

2) Is there a cross-reference framework through which the code can be easily and directly traced to the software design documentation?

3) Does the code contain or reference a revision history of all code modifications and the reason for them?

4) Have all safety and computer security functions been flagged?

i. Understandability

1) Do the comment statements adequately describe each routine, using clear English language?

2) Were ambiguous or unnecessarily complex coding used? If so, are they clearly commented?

3) Were consistent formatting techniques (e.g., indentation, use of white space) used to enhance clarity?

4) Was a mnemonic naming convention used? Does the naming reflect the type of variable?

5) Is the valid range of each variable defined?

6) Does the code use mathematical equations which correspond to the mathematical models described/derived in the SDD?

j. Verifiability - Are implementation practices and techniques that are difficult to test avoided?