Volume I, Section 4
Table of Contents

4 Software Standards

4.1 Scope

This section describes essential design and performance characteristics of the software used in voting systems, addressing both system-level software, such as operating systems, and voting system application software, including firmware. The requirements of this section are intended to ensure that voting system software is reliable, robust, testable, and maintainable. The standards in this section also support system accuracy, logical correctness, privacy, security and integrity.

The general requirements of this section apply to software used to support the entire range of voting system activities described in Section 2. More specific requirements are defined for ballot counting, vote processing, creating an audit trail, and generating output reports and files. Although this section emphasizes software, the standards described also influence hardware design considerations.

This section recognizes that there is no best way to design software. Many programming languages are available for which modern programming practices are applicable, such as the use of rigorous program and data structures, data typing, and naming conventions. Other programming languages exist for which such practices are not easily applied.

The Standards are intended to guide the design of software written in any of the programming languages commonly used for mainframe, mini-computer, and microprocessor systems. They are not intended to preclude the use of other languages or environments, such as those that exhibit “declarative” structure, “object-oriented” languages, “functional” programming languages, or any other combination of language and implementation that provides appropriate levels of performance, testability, reliability, and security. The vendor makes specific software selections. However, the use of widely recognized and proven software design methods will facilitate the analysis and testing of voting system software in the qualification process.

4.1.1 Software Sources

The requirements of this section apply generally to all software used in voting systems, including:

- ♦ Software provided by the voting system vendor and its component suppliers;
- ♦ Software furnished by an external provider (for example, providers of COTS operating systems and web browsers) where the software may be used in any way during voting system operation; and
- ♦ Software developed by the voting jurisdiction.

Compliance with the requirements of the software standards is assessed by several formal tests, including code examination. Unmodified software is not subject to code examination; however, source code generated by a package and embedded in software modules for compilation or interpretation shall be provided in human readable form to the ITA. The ITA may inspect source code units to determine testing requirements or to verify that the code is unmodified and that the default configuration options have not been changed.

Configuration of software, both operating systems and applications, is critical to proper system functioning. Correct test design and sufficient test execution must account for the intended and proper configuration of all system components. Therefore, the vendors shall submit to the ITA, in the TDP, a record of all user selections made during software installation. The vendor shall also submit a record of all configuration changes made to the software following its installation. The ITA shall confirm the propriety and correctness of these user selections and configuration changes.

4.1.2 Location and Control of Software and Hardware on Which it Operates

The requirements of this section apply to all software used in any manner to support any voting-related activities, regardless of the ownership of the software or the ownership and location of the hardware on which the software is installed or operates. These requirements apply to:

- ♦ Software that operates on voting devices and vote counting devices installed at polling places under the control of the voting jurisdiction;
- ♦ Software that operates on ballot printers, vote counting devices, and other hardware typically installed at central or precinct locations (including contractor facilities); and
- ♦ Election management software.

However, some requirements apply only in specific situations indicated in this section. In addition to the requirements of this section, all software used in any manner to support any voting-related activities shall meet the requirements for security described in Section 6 of the Standards.

4.1.3 Exclusions

Some voting systems use equipment, such as personal computers, that may be used for other purposes and have resident on the equipment general purpose software such as operating systems, programming language compilers, database management systems, and Web browsers. Such software is governed by the Standards unless:

- The software provides no support of voting system capabilities;
- The software is removable, disconnectable, or switchable such that it cannot function while voting system functions are enabled; and
- Procedures are provided that confirm that the software has been removed, disconnected, or switched.

4.2 Software Design and Coding Standards

The software used by voting systems is selected by the vendor and not prescribed by the Standards. This section provides standards for voting system software with regard to:

- Selection of programming languages;
- Software integrity;
- Software modularity and programming;
- Control constructs;
- Naming conventions;
- Coding conventions; and
- Comment conventions.

4.2.1 Selection of Programming Languages

Software associated with the logical and numerical operations of vote data shall use a high-level programming language, such as: Pascal, Visual Basic, Java, C and C++. The requirement for the use of high-level language for logical operations does not preclude the use of assembly language for hardware-related segments, such as device controllers and handler programs. Also, operating system software may be designed in assembly language.

4.2.2 Software Integrity

, Self-modifying, dynamically loaded, or interpreted code is prohibited, except under the security provisions outlined in section 6.4.e. This prohibition is to ensure that the software tested and approved during the qualification process remains unchanged and retains its integrity. External modification of code during execution shall be prohibited. Where the development environment (programming language and development tools) includes the following features, the software shall provide controls to prevent accidental or deliberate attempts to replace executable code:

- Unbounded arrays or strings (includes buffers used to move data);
- Pointer variables; and
- Dynamic memory allocation and management.

4.2.3 Software Modularity and Programming

Voting system application software, including COTS software, shall be designed in a modular fashion. However, COTS software is not required to be inspected for compliance with this requirement.. For the purpose of this requirement[1], "modules" may be compiled or interpreted independently. Modules may also be nested. The modularity rules described here apply to the component sub modules of a library. The principle concept is that the module contains all the elements to compile or interpret successfully and has limited access to data in other modules. The design concept is simple replacement with another module whose interfaces match the original module. A module is designed in accordance with the following rules:

[1] Some software languages and development environments use a different definition of module but this principle still applies.

a. Each module shall have a specific function that can be tested and verified independently of the remainder of the code. In practice, some additional modules (such as library modules) may be needed to compile the module under test, but the modular construction allows the supporting modules to be replaced by special test versions that support test objectives;

b. Each module shall be uniquely and mnemonically named, using names that differ by more than a single character. In addition to the unique name, the modules shall include a set of header comments identifying the module's purpose, design, conditions, and version history, followed by the operational code. Headers are optional for modules of fewer than ten executable lines where the subject module is embedded in a larger module that has a header containing the header information. Library modules shall also have a header comment describing the purpose of the library and version information;

c. All required resources, such as data accessed by the module, should either be contained within the module or explicitly identified as input or output to the module. Within the constraints of the programming language, such resources shall be placed at the lowest level where shared access is needed. If that shared access level is across multiple modules, the definitions should be defined in a single file (called header files in some languages, such as C) where any changes can be applied once and the change automatically applies to all modules upon compilation or activation;

d. A module is small enough to be easy to follow and understand. Program logic visible on a single page is easy to follow and correct. Volume II, Section 5 provides testing guidelines for the ITA to identify large modules subject to review under this requirement;

e. Each module shall have a single entry point, and a single exit point, for normal process flow. For library modules or languages such as the object-oriented languages, the entry point is to the individual contained module or method invoked. The single exit point is the point where control is returned. At that point, the data that is expected as output must be appropriately set. The exception for the exit point is where a problem is so severe that execution cannot be resumed. In this case, the design must explicitly protect all recorded votes and audit log information and must implement formal exception handlers provided by the language; and

f. Process flow within the modules shall be restricted to combinations of the control structures defined in Volume II, Section 5. These structures support the modular concept, especially the single entry/exit rule above. They apply to any language feature where program control passes from one activity to the next, such as control scripts, object methods, or sets of executable statements, even though the language itself is not procedural.

4.2.4 Control Constructs

Voting system software shall use the control constructs identified in Volume II, Section 5:

a. Acceptable constructs are Sequence, If-Then-Else, Do-While, Do-Until, Case, and the General loop (including the special case for loop);

b. If the programming language used does not provide these control constructs, the vendor shall provide them (that is, comparable control structure logic). The constructs shall be used consistently throughout the code. No other constructs shall be used to control program logic and execution;

c. While some programming languages do not create programs as linear processes, stepping from an initial condition, through changes, to a conclusion, the program components nonetheless contain procedures (such as "methods" in object-oriented languages). Even in these programming languages, the procedures must execute through these control constructs (or their equivalents, as defined and provided by the vendor); and

d. Operator intervention or logic that evaluates received or stored data shall not re-direct program control within a program routine. Program control may be re-directed within a routine by calling subroutines, procedures, and functions, and by interrupt service routines and exception handlers (due to abnormal error conditions). Do-While (False) constructs and intentional exceptions (used as GoTos) are prohibited.

4.2.5 Naming Conventions

Voting system software shall use the following naming conventions:

a. Object, function, procedure, and variable names shall be chosen so as to enhance the readability and intelligibility of the program. Insofar as possible, names shall be selected so that their parts of speech represent their use, such as nouns to represent objects, verbs to represent functions, etc.;

b. Names used in code and in documentation shall be consistent;

c. Names shall be unique within an application. Names shall differ by more than a single character. All single-character names are forbidden except those for variables used as loop indexes. In large systems where subsystems tend to be developed independently, duplicate names may be used where the scope of the name is unique within the application. Names should always be unique where modules are shared; and

d. Language keywords shall not be used as names of objects, functions, procedures, variables, or in any manner not consistent with the design of the language.

4.2.6 Coding Conventions

Voting system software shall adhere to basic coding conventions. The coding conventions used shall meet one of the following conditions:

a. The vendors shall identify the published, reviewed, and industry-accepted coding conventions used and the ITAs shall test for compliance; or

b. The ITAs shall evaluate the code using the coding convention requirements specified in Volume II, Section 5.

These standards reference conventions that protect the integrity and security of the code, which may be language-specific, and language-independent conventions that significantly contribute to readability and maintainability. Specific style conventions that support economical testing are not binding unless adopted by the vendor.

4.2.7 Comment Conventions

Voting system software shall use the following comment conventions:

a. All modules shall contain headers. For small modules of 10 lines or less, the header may be limited to identification of unit and revision information. Other header information should be included in the small unit headers if not clear from the actual lines of code. Header comments shall provide the following information:

   1) The purpose of the unit and how it works;
   2) Other units called and the calling sequence;
   3) A description of input parameters and outputs;
   4) File references by name and method of access (read, write, modify , append, etc.);
   5) Global variables used; and
   6) dDate of creation and a revision record;

b. Descriptive comments shall be provided to identify objects and data types. All variables shall have comments at the point of declaration clearly explaining their use. Where multiple variables that share the same meaning are required, the variables may share the same comment;

c. In-line comments shall be provided to facilitate interpretation of functional operations, tests, and branching;

d. Assembly code shall contain descriptive and informative commentssuch that its executable lines can be clearly understood; and

e. All comments shall be formatted in a uniform manner that makes it easy to distinguish them from executable code.

4.3 Data and Document Retention

All systems shall:

a. Maintain the integrity of voting and audit data during an election, and for at least 22 months thereafter, a time sufficient in which to resolve most contested elections and support other activities related to the reconstruction and investigation of a contested election; and

b. Protect against the failure of any data input or storage device at a location controlled by the jurisdiction or its contractors, and against any attempt at improper data entry or retrieval.

4.4 Audit Record Data

Audit trails are essential to ensure the integrity of a voting system. Operational requirements for audit trails are described in Section 2.2.5.2 of the Standards. Audit record data are generated by these procedures. The audit record data in the following subsections are essential to the complete recording of election operations and reporting of the vote tally. This list of audit records may not reflect the design constructs of some systems. Therefore, vendors shall supplement it with information relevant to the operation of their specific systems.

4.4.1 Pre-election Audit Records

During election definition and ballot preparation,, the system shall audit the preparation of the baseline ballot formats and modifications to them, a description of these modifications, and corresponding dates. The log shall include:

a. The allowable number of selections for an office or issue;

b. The combinations of voting patterns permitted or required by the jurisdiction;

c. The inclusion or exclusion of offices or issues as the result of multiple districting within the polling place;

d. Any other characteristics that may be peculiar to the jurisdiction, the election, or the polling place's location;

e. Manual data maintained by election personnel;

f. Samples of all final ballot formats; and

g. Ballot preparation edit listings.

4.4.2 System Readiness Audit Records

The following minimum requirements apply to system readiness audit records:

a. Prior to the start of ballot counting, a system process shall verify hardware and software status and generate a readiness audit record. This record shall include the identification of the software release, the identification of the election to be processed, and the results of software and hardware diagnostic tests;

b. In the case of systems used at the polling place, the record shall include the polling place's identification;

c. The ballot interpretation logic shall test and record the correct installation of ballot formats on voting devices;

d. The software shall check and record the status of all data paths and memory locations to be used in vote recording to protect against contamination of voting data;

e. Upon the conclusion of the tests, the software shall provide evidence in the audit record that the test data have been expunged;

f. If required and provided, the ballot reader and arithmetic-logic unit shall be evaluated for accuracy, and the system shall record the results. It shall allow the processing, or simulated processing, of sufficient test ballots to provide a statistical estimate of processing accuracy; and

g. For systems that use a public network, provide a report of test ballots that includes:

   1) Number of ballots sent;

   2) When each ballot was sent;

   3) Machine from which each ballot was sent; and

4) Specific votes or selections contained in the ballot.

4.4.3 In-Process Audit Records

In-process audit records document system operations during diagnostic routines and the casting and tallying of ballots. At a minimum, the in-process audit records shall contain:

a. Machine generated error and exception messages to demonstrate successful recovery. Examples include, but are not necessarily limited to:

   1) The source and disposition of system interrupts resulting in entry into exception handling routines;

   2) All messages generated by exception handlers;

   3) The identification code and number of occurrences for each hardware and software error or failure;

   4) Notification of system login or access errors, file access errors, and physical violations of security as they occur, and a summary record of these events after processing;

   5) Other exception events such as power failures, failure of critical hardware components, data transmission errors, or other type of operating anomaly;

b. Critical system status messages other than informational messages displayed by the system during the course of normal operations. These items include, but are not limited to:

   1) Diagnostic and status messages upon startup;

   2) The “zero totals” check conducted before opening the polling place or counting a precinct centrally;

   3) For paper-based systems, the initiation or termination of card reader and communications equipment operation; and

   4) For DRE machines at controlled voting locations, the event (and time, if available) of activating and casting each ballot (i.e., each voter's transaction as an event). This data can be compared with the public counter for reconciliation purposes;

c. Non-critical status messages that are generated by the machine's data quality monitor or by software and hardware condition monitors; and

d. System generated log of all normal process activity and system events that require operator intervention, so that each operator access can be monitored and access sequence can be constructed.

4.4.4 Vote Tally Data

In addition to the audit requirements described above, other election-related data is essential for reporting results to interested parties, the press, and the voting public, and is vital to verifying an accurate count.

Voting systems shall meet these reporting requirements by providing software capable of obtaining data concerning various aspects of vote counting and producing reports of them on a printer. At a minimum, vote tally data shall include:

a. Number of ballots cast, using each ballot configuration, by tabulator, by precinct, and by political subdivision;

b. Candidate and measure vote totals for each contest, by tabulator;

c. The number of ballots read within each precinct and for additional jurisdictional levels, by configuration, including separate totals for each party in primary elections;

d. Separate accumulation of overvotes and undervotes for each contest, by tabulator, precinct and for additional jurisdictional levels (no overvotes would be indicated for DRE voting devices); and

e. For paper-based systems only, the total number of ballots both processed and unprocessable; and if there are multiple card ballots, the total number of cards read.

For systems that produce an electronic file containing vote tally data, the contents of the file shall include the same minimum data cited above for printed vote tally reports.

4.5 Vote Secrecy (DRE Systems)

All DRE systems shall ensure vote secrecy by:

a. Immediately after the voter chooses to cast his or her ballot, record the voter's selections in the memory to be used for vote counting and audit data (including ballot images), and erase the selections from the display, memory, and all other storage, including all forms of temporary storage; and

b. Immediately after the voter chooses to cancel his or her ballot, erase the selections from the display and all other storage, including buffers and other temporary storage.