"Douglas W. Jones" <jones@cs.uiowa.edu> on 01/29/2002 02:34:30 PM

To: vss@FEC
cc:

Subject: Comments on Draft VSS

Comments on the FEC's Draft Voting System Standards
As revised, Dec 13, 2001

Douglas W. Jones
Department of Computer Science
MacLean Hall
University of Iowa
Iowa City, Iowa 52242

(319)335-0740
jones@cs.uiowa.edu

Submitted to the FEC by E-mail, Jan 29, 2002; This material is also indexed on the web at http://www.cs.uiowa.edu/~jones/voting/

I've read through the entire voting system standards draft, found at

http://fecweb1.fec.gov/pages/vss/vss.html

and I would be happy to appear before the commission, if requested.

By way of introduction, I have served on the Iowa Board of Examiners for Voting Machines and Electronic Voting Systems for almost a decade, and I now chair that board. I have served on the computer science faculty at the University of Iowa for over 20 years, and in addition to extensive comments on the previous draft, I have testified about voting systems and practices before the United States Civil Rights Commission and before the House Science Committee. My full vita can be found on my web page at http://www.cs.uiowa.edu/~jones/

Overall Comments

First, a brief comment on the process itself. The on-line distribution of the standard was very difficult to use and print. Most of the PDF files were difficult or impossible to print using the version of Adobe Acroread installed on our departmental systems, probably because of eccentric PDF usage. I strongly recommend that the distribution of draft Federal standards public comment should itself live up to higher standards! Ideally, the draft should be easily readable and printable using a variety of computers running a variety of operating systems from multiple vendors, and nothing in the draft should require the installation of the latest software version from some particular vendor! The Federal Government must not allow itself to become part of the marketing arm of any system vendor!

Second, there are some things I'd hoped for in Volume II that aren't there. I'd hoped for more human factors material, or at least more to support the interaction between the physical and software standards of Volume I and the human factors standards that will be developed. I've

commented on this extensively, because I think that we can specify many things now to support human factors standards later.

Detailed Comments on Volume I

1 Introduction
1.5 Definitions

The revised terminology in this second draft is excellent. The distinction between paper-based and DRE systems is useful, the new term public-network DRE correctly reflects the fact that so-called E-voting or Internet voting is just a special category of DRE voting, and the application of the precinct count versus central count distinction to all categories of voting systems is correct.

2 Functional Capabilities
2.2 Overall System Capabilities
2.2.4 Integrity
2.2.4.2 Common Standards

The redundancy requirement of 2.2.2.2 is nicely expanded on in 2.2.4.2; this is an improvement over the first draft!

2.2.7 Accessibility
2.2.7.2 DRE Standards
2.2.7.2.b Provide audio information ...

7) Provides the audio signal ... using a 1/8 inch plug ...

You should specify this as a 1/8 inch stereo headphone jack. This is the type of jack that common headphones use (such as the Sony Walkman), and there are other 1/8 inch plugs on the market that have nothing to do with headphones!

2.2.7.2.d For electronic image displays ...

3) Adjust the screen font size to at least 18 points.

Printers usually define a point as approximately 1/72 of an inch, a very strange unit of measure. The traditional British-American definition is that a pica is 0.166 inch, and a point is 1/12 pica. This works out to 1/72.289157 or 0.13833333 inches. This is very awkward, so modern computer-based typesetters have changed the definition to exactly 1/72 of an inch when printing on paper.

Sadly, none of these definitions have been applied uniformly to text presented on a computer screen! 18-point type on one screen will generally not be the same size as 18-point type on another. For several years, Apple made screens with 72 pixels to the inch, while IBM had a default of 96 pixels to the inch. Both tended to define one point as one pixel, and that was a close enough definition to suit the marketplace.

To add to the confusion, the traditional American way of measuring the point size of a type font is to measure the distance from the highest ascender in the font (this might be the top of the letter b) to the bottom of the lowest descender (this might be the bottom of the letter q); Europeans, have traditionally measured letter sizes differently. Because the classic Times Roman font has relatively long ascenders and descenders, the apparent size of 18-point Times Roman text is noticeably smaller than that of 18-point Helvetica, a font with relatively short ascenders and descenders. In sum, point sizes are assigned nominally, with each font designer likely to use different definitions of a point and

likely to use this measure differently. Point size is not meaningful from a human factors point of view!

Given all this, I strongly recommend against specifying text sizes using points. Stick to well defined international standard units such as millimeters. These mean the same thing on paper as they do on a computer screen, and they mean the same thing no matter what brand monitor is attached to the computer! 18 points times 1/72 of an inch comes to exactly 0.25 inches; 18 traditional points is 0.249 inches; thus, the standard, as worded, calls for a dimension of 6.32 or 6.36 millimeters.

Finally, why is no minimum specified. I find 1-point type really hard to read, and I find 4-point type readable but difficult; this is about 1/4 the height of 18 point type. Looking at ballots in my collection, candidate names are printed type with a capital letter height of 3 to 4 millimeters. So, I suggest the following wording:

3) Adjust the size of the text so that the height of capital letters varies over a range of 3 to 6.3 millimeters.

2.2.7.3 Paper-Based Voting Standards

Section 2.2.7.2 should be numbered 2.2.7.3, and a new subsection, 2.2.7.2 Paper-Based systems, should be added! In fact, later parts of the standards make it clear that you are requiring significant handicapped accessibility features for paper-based systems, and you should say that here. For example, 2.4.3.1 requires the ability to magnify a ballot on all voting systems, and this might be the right place to mention this.

There are other possible modifications to paper-based systems to offer handicapped accessibility; I can easily imagine embossing the booklet on the face of a Votomatic machine with Braille names for the candidates, and I can easily imagine extending this idea to optical mark-sense ballots by using an embossed overlay with a hole punched through over each valid voting target. A similar overlay, made of transparent plastic, could help voters with minor motor disabilities avoid making stray marks on the face of the ballot.

If we don't know what assistive technologies to demand here for paper-based systems, we can at least ask the vendor to document what they provide, in the hopes that the marketplace will focus fairly rapidly on an appropriate toolkit of assistive technologies once the vendors start to document the technologies they can offer.

2.2.8 Vote Tabulating
2.2.8.e Write-in voting;

There are several distinct questions that lie hidden under the accommodation of write-in voting. Does it merely allow write-in voting? Does it also sort ballots (or ballot images) containing write-in votes for special treatment? Does it offer help in tabulating write-in votes? These are distinct questions that need distinct answers.

With regard to the final question listed above, I've seen DRE machines that simply print the full ballot image for each ballot containing a write-in vote, leaving it to the precinct officials to go through all of them, and I've seen other machines that count all occurrences of the same spelling, so you get write-in subtotals like "10 for D.Duck, 5 for Donald Duck" and therefore greatly simplify the count. Today's handwriting recognition software is

almost good enough to allow similar support for write-ins on a paper-ballot system.

2.2.8.m Overvotes;
2.2.8.n Undervotes; and
2.2.8.o Totally blank ballots;

I assume you mean the system must be able to count overvotes, undervotes and blank ballots, but some systems can also sort ballots (or ballot images) according to whether they contain one of these, and some systems can prevent the voting of a ballot containing one of these. As with the write-in issue, we must distinguish between these levels of tabulation support.

2.4 Voting Functions
2.4.3 Casting a Ballot
2.4.3.1 Common Standards

a) Provide a capability to adjust ... to at least 18 points.

See my note on paragraph 2.2.7.2.d on the subject of using points as a measure. This section raises even more problems, because it mentions a capability to magnify the type. Both points and millimeters are a measure of linear dimension, not apparent visual size, and a magnifying glass (an obvious assistive technology for use on a paper-based voting system) does nothing to change the linear dimension! It only changes the apparent size.

Therefore, I suggest that the wording should specify a minimum size and a degree of magnification, worded as follows:

a) Provide text that is at least 3 millimeters high, and provide the capability to adjust or magnify the text to an apparent size of 6.3 millimeters.

2.4.3.2 Paper-Based System Standards
2.4.3.2.2 Precinct-Count Paper-Based System Standards
2.4.3.2.2.c Allow an authorized election official to turn off ...

The use (or non-use) off the capabilities in question must be uniform state-wide! Non-uniform use of these capabilities within a state during a single election would lead to giving voters in one jurisdiction an advantage over voters in another jurisdiction, in exactly the same way that the differing recount standards in Florida in 2000 lead to such differences, and the supreme court rulings that followed make clear that such non-uniformity within a state is illegal.

Therefore, as near as I can tell, this feature must be enabled or disabled on all precinct-count machines in a state, and not done on a county-by-county or precinct-by-precinct basis. As such, I would hope that these features could be locked on or locked off in all versions of the precinct-count mechanism sold in each state, with no ability of the local election official to override.

In fact, we had exactly this problem in Johnson County Iowa in 2000. Our state administrative rules require precinct count vote tabulators to return overvoted ballots to the voter if possible, and all new systems are required to support this feature. The Johnson County Auditor (supervisors of elections) disabled this feature because he thought it gave an unfair advantage to those voting at polling places (as opposed to absentee voters), despite the state administrative code. He shouldn't have been able to do so!

These comments apply also to 3.2.5.1.3.

2.4.3.3 DRE System Standards
2.4.3.3.o Record an image of the ballot ... in a form readable by humans...

The requirement for a human readable record could be taken as a requirement for a printer embedded in each DRE machine. Some early machines had this. The clarification referencing Section 2.2.2.2.2 references a nonexistent section!

If the "form readable by humans" is explicitly clarified as including humans aided by a computer, I would be happier.

2.4.3.3.p Increment the proper ballot position registers or counters;

This presumes a precinct-count DRE system. Central count DRE systems are quite possible. If we ever get vote-at-home absentee DRE voting (Most use of the term Internet-voting has referred to this), it must be based on central count technology, since maintaining a copy of the ballot cast on the voter's own computer would be very bad!

I suggest adding a Section 2.4.3.3.1 that addresses added requirements for precinct count DRE systems, and possibly 2.4.3.3.2 addressing central count DRE systems - at minimum, this would state that the communication between DRE machine and counting center, by whatever means, must meet the requirements of later sections on information security.

2.5 Post-Voting Functions
2.5.1 Closing the Polling Place (Precinct Count)

The qualifier "precinct count" in the title of this section is a source of problems. If we have a central count DRE system, for example, one that uses telecommunications or some kind of hand carried PROM-pack to send all the ballot images to a central counting center for tabulation, all of the items in this section should still apply.

The only class of voting systems to which this section does not apply is that of central count paper ballot systems.

2.5.2 Consolidating Vote Data
2.5.2.c From other sources ...

One problem with automatic consolidation is that you need a way for the election staff (in some cases, at the precinct, in other cases, at the counting centers) to adjust the consolidated figures. These adjustments must be audited with extreme care, and each such adjustment an event that should go into the audit log.

The requirement that the other sources included in the totals be listed by the vendor is not sufficient. Some of the other sources are easy to identify and should be listed explicitly:

The draft mentions early voting.

The draft does not mention challenged or provisional ballots. Right now, in some jurisdictions, after the challenges are resolved, the challenged ballots that are accepted are put through the precinct-count tabulators even though the polls are already closed. The new standard may be read to forbid reopening the polls (2.5.1.a), so we need a formal mechanism for doing this!

We also need a formal method to reopen the count after the polls

are closed for the purpose of counting ballots that had been set aside during the election. For example, in Iowa, if a precinct-count machine rejects an overvoted ballot and the voter leaves without making a decision about that ballot, it is set aside. After the polls close, the polling place workers examine the condition the machine detected as an overvote to see if it exhibits evidence of voter intent (erasure, obvious cross-out and ballot printing errors may all lead to machine-detected overvotes that should be counted as valid votes). Once the problem has been resolved, the ballot must be included in the tally!

The draft does not mention adjusting the totals to account for the consequences of improper write-in votes.

This latter category poses some special problems! If someone writes in the name of a candidate that is already on the ballot, depending on state law, this write-in vote may be counted as a normal vote for that candidate or ignored. If someone votes for a candidate and also writes in the name of that same candidate, depending on state law, this may count as an overvote, or it may be counted as one vote for that candidate. Thus, the processing of such "improper" write-in votes can add or subtract from the count for a candidate who's name is on the ballot, depending on state law and on the particular "abuse" of the write-in privilege the voter makes.

2.5.3 Producing Results
2.5.3.1 Common Standards
2.5.3.1.c Produce a report for each tabulator of the results ...
2.5.3.1.d Produce a consolidated report for each contest ...

Both items c and d ask for a report of all specific combinations of votes cast to make an overvote. Thus, in a 3-way race for one office, we need to count:

1. undervotes (votes for no candidate)
2. votes for candidate a
3. votes for candidate b
4. votes for candidate c
5. 2-way overvotes for candidates a and b
6. 2-way overvotes for candidates a and c
7. 2-way overvotes for candidates b and c
8. 2-way overvotes for candidates a, b and c

This is workable, requiring only 8 registers. However, when the race grows, the number of registers required grows exponentially. Specifically, for an N-candidate race, we require 2 raised to the N registers, one to count undervotes, N to count legitimate votes for one of the N candidates, and the remainder to count various combinations of overvotes. For a 10-way race, the zero-totals report and final vote-totals reports will be immense, with a total of 1024 entries! We should compact this report by printing only the nonzero overvote combinations, but even then, in a race with thousands or millions of votes, there is the possibility that some few voters will try each of these myriad alternative ways of casting an overvote.

Furthermore, depending on how the state handles write-in votes, the number of candidates for whom votes must be counted may not be known in advance! For the purpose of the overvote tally, it is be sufficient to lump the write-ins as a single pseudo-candidate named write-in, except that, in many states, there are special rules for write-in votes cast for a candidate who's name already occurs on the ballot. Such votes are discounted under Iowa law, but we are proposing a change that would accept them.

I strongly suggest that complex overvote analysis not be included in the first-count. A simple count of the number of ballots containing overvotes in each race is sufficient!

That said, if the ballot images from the tabulating system are available in a standardized format, retrospective analysis of the overvotes should be straightforward, and in the event that a race elicits an unusual number of overvotes, it should be possible to do a retrospective analysis of the overvote patterns.

3 Hardware Standards
3.2 Performance Requirements
3.2.2 Environmental Requirements
3.2.2.1 Shelter Requirements

"... habitable facility ordinarily used as a warehouse ..." is an odd term. I understand the word habitable to refer to the possibility of the space being used for habitation. In contrast, many warehouses have climate controls that do not maintain habitable temperatures. Section 3.2.2.6 explicitly states the possibility of uncontrolled warehouse facilities - with respect to both temperature and humidity, and such facilities are not usually considered habitable.

3.2.2.4 Electrical Supply

The voltages and line frequencies listed in paragraphs a and b are all nominal, with no statement of noise or brownout tolerance. These constraints really ought to be lifted verbatim (or by citation) to relevant commercial or industrial standards.

In fact, the missing details are listed in Section 3.4.7, with additional detail in 3.4.11 and 3.4.12. These should be referenced here.

3.2.4 Vote Recording Requirements
3.2.4.2 Paper-Based Recording Standards
3.2.4.2.1 Paper Ballot Standards

Paragraphs a and b both say that various attributes of the paper or cardstock used must conform with the vendor's specs. Most of the items listed are the same. Given this, why are these paragraphs distinct? Can't they be combined?

Paragraph c permits the use of timing marks and forbids their appearing in any data fields. This prohibition is odd. In fact, it is perfectly feasible to encode useful data (for example, ballot style information) in the timing, alignment or fiducial marks (for practical purposes, these terms are synonyms). For example, the top and bottom edges of a set of marks parallel to the column of votes may be used for timing purposes, while the width of those same marks may be used to encode ballot style information. On the face of it, item c forbids this.

The following wording appears closer to what I guess is the intent of this paragraph:

c) The ballot tabulating system may use printed or punched alignment marks to locate the voting targets on the ballot, so long as these marks are clearly outside the area of the ballot in which votes are cast, so as to minimize the likelihood that these marks will be mistaken for voting targets or the likelihood that casting votes will obliterate these marks.

3.2.4.2.3 Marking Devices

It is difficult to send useful marking devices through the mail with absentee ballots. Postal rules impose a surcharge for mailings over 1/4 inch thick, so the usual pencil included with an absentee ballot is too slim to hold comfortably; as a result, many absentee voters will use whatever pen or pencil is handy!

Even in polling places, markers provided to voters frequently run dry by the end of a long day of voting, and the tips of provided pencils break. If the lines are long and the poll-workers are looking overworked, voters will frequently use their own pens or pencils instead of requesting official replacements.

This is an area where human factors and technical specifications of the voting system both play a part!

Therefore, it is crucial that mark-sense tabulating machines be tested with a wide variety of commonly used marking devices! The vendor should provide broad descriptions of the types of marking devices that, if used to make the prescribed form of mark, have proven to be reliably readable, and to illustrate the limits of the scanning technology, the vendor should also list marking devices that do not make reliably readable marks.

By way of illustration, I recently completed a series of tests on the ES&S PBC 100 where I demonstrated that it could not reliably detect marks made with a pink highlighter, while it could reliably read marks made in the approved manner with the other eight devices I tried.

I also strongly recommend that in addition to simple lists of marking implements, the vendor should disclose the spectral sensitivity of the system. Is it sensing the reflectivity of the marks in one specific narrow band of colors (infrared, red and green are commonly used by many of today's ballot scanners), or do the sensors being used respond over a broad range of wavelengths.

Under current rules, I have seen changes in sensor technology made without any notice to the customers and without any change in the ballot marking instructions. In the case of the ES&S 150, for example, the original machines sensed the ballot using infrared light, while newer and upgraded machines use visible light (red or green). This change should have been made uniformly, on a statewide basis, so that the same counting criteria are used in all counties in the state that use this machine, and the instructions should have been changed to conform with the change -- where the old ballots had said, "Use blue or black pencil or ink, DO NOT USE RED!" (because the old infrared sensors were frequently unable to read red ink) the new instructions may need to say "DO NOT USE GREEN" if the new system uses green illumination.

See http://www.cs.uiowa.edu/~jones/voting/optical/ for more detail on these issues, in my tutorial on optical mark-sense voting technology.

3.2.4.2.5 Frames or Fixtures for Printed Ballots

In general, this section is good. I believe we will see more frame use for mark-sense ballots than is common today, as such frames can be used to solve several problems with handicapped accessibility and multilingual ballot presentation.

However, all punched card ballots I know of are printed, if nothing else, with ballot position numbers and a dot on each

potential voting target. Given this, I think the term mark-sense ballot is better than printed ballot for this section.

3.2.4.3. DRE Systems Recording Requirements
3.2.4.3.2. DRE System Vote Recording
3.2.4.3.2.b Incorporate multiple memories ... to detect discrepancy.

Memory is passive, it cannot detect anything. Multiple memories allow detection and, if used properly, correction of errors. Therefore, the phrase "to detect discrepancy in the content of individual memories" should be changed to "to allow detection and correction of errors caused by failures of one or the other of the individual memories."

3.2.4.3.2.c Provide at least two processes ...

This requirement is good, but item 1) goes too far -- it may be impossible to entirely avoid using common software and common storage devices for these processes. What we must demand is that, to the extent possible, these processes must be isolated from each other, and in addition, the common components that they are forced to share (for example, the touch screen interface software) must be subject to a far higher standard during the code and design audit than the isolated parts.

3.2.5 Paper-based Conversion Requirements
3.2.5.1 Ballot Handling
3.2.5.1.2 Exception Handling (central count)

This section, particularly the final paragraph, is a welcome improvement, however, it does not support the legislation currently under consideration in Iowa. This requires, in addition to the features required in the draft, the ability to identify (on first count) blank ballots, and (on a recount of a particular race) undervotes in that race -- because we want all undervotes to be subject to visual inspection on a recount, and all blank ballots to be subject to such inspection on the first count. Therefore, I suggest the following wording for the last paragraph of this section:

Additionally, the system shall provide capabilities can be activated by an authorized election official to identify ballots containing overvotes, blank ballots, and ballots containing undervotes in a designated race. If enabled, these capabilities shall perform one of the above actions in response to the indicated condition.

3.2.5.1.3 Exception Handling (precinct count)

See my comments on 2.4.3.2.2.c - the decision to enable or disable these features is a state decision, not one that can be safely left to the local jurisdiction. The machine (or upgrade to the machine) should be sold with these settings fixed as required by state law, and the ability to change these settings should be tightly guarded.

Also, in addition to the ability to return ballots that are undervoted in a particular race, we need the ability to return a totally blank ballot -- use of the wrong marking implement may make an otherwise perfectly good ballot read as blank, while undervotes in a particular race are common and are more likely to be the result of a voter's intentional abstention.

Finally, I doubt that a state will want to make it inconvenient to abstain from bottom-of-the-ticket races such as county soil and water conservation district board, while it may well be valuable

to detect undervotes in presidential and gubernatorial races. Therefore, item d.5 should probably ask that this facility be disabled on a race-by-race basis.

3.2.5.2 Ballot Reading Accuracy

This is an area where human factors and technical specifications of the voting system both play a part!

Therefore, we need far more than a mere statement that the machine accurately recognizes votes made using prescribed marking or punching implements to mark or punch the ballot in the prescribed way, and that the machine can correctly distinguish these from unmarked or unpunched ballots.

Note that the set of all ballot markings or punchings can be divided into several sets:

Prescribed marks or punchings, those made in exact conformance to the instructions given to the voter.
Reliably detectable marks or punchings, those that the tabulating machine will reliably detect and count. The prescribed mark or punching must, of course, be reliably detectable!
Marginal marks or punchings, those that will sometimes be detected but will sometimes be ignored. No physical tabulating machine can prevent a mark falling in this category.
Reliably ignored marks or punchings, those that will never be detected and counted.
Acceptable mark or punch, a mark or punch that meets the legal requirement for acceptance as a vote. A state should not accept the machine if the machine reliably ignores a significant number of acceptable marks that are found to occur in practice.
Unacceptable mark or punch, a mark or punch that does not meet the legal requirement for acceptance as a vote. A state should not accept the machine if the machine reliably detects a significant number of unacceptable marks.

In order to allow the conformity of the vote tabulaing system with state laws governing acceptable and unacceptable marks to be determined, the vendor must be required to clearly document not only the prescribed mark or punch, but also the nature of the class of reliably acceptable and reliably ignored marks or punches.

These should be described in relation to marks or punches found in practice - for example, for a machine that counts votes cast using Votomatic-style pre-scored ballots, does it reliably detect holes through the chad made by forcing the voting stylus through the chad without detaching the chad, or are these marginal? For an optical mark-sense ballot where the prescribed mark is a black filled in oval, does the machine reliably count X and checkmarks made in the oval?

One way of documenting the machine's sensitivity is to document the largest hole or darkest mark that is sometimes be ignored and also the smallest hole or lightest mark that is sometimes detected. These are the extremes of the marginal class.

These comments also relate to 9.2.1.7 where the tests of counting accuracy are described.

See http://www.cs.uiowa.edu/~jones/voting/optical/ for more

detail on these issues, in my tutorial on optical mark-sense voting technology.

3.4 Design, Construction and Maintenance Characteristics
3.4.7 Electrical Power Disturbance

This clarifies requirements in Section 3.2.2.4, and I am surprised that it isn't cited from there or gathered with that material.

Additionally, 3.4.11 and 3.4.12 add further clarification, and it is odd that they're not subsections of one single section on electrical disturbance.

3.4.12 Electrical Fast Transient

Lightning Surges are just one kind of fast transient, and it is really odd that they're in a separate section. A system meeting the requirements of Section 3.4.12 would appear to meet all of the requirements of 3.4.11, and I wonder if these can therefore be combined.

3.4.15 Product Marking

Given the prohibition in Section 2.3.1.3.1.b against the display of any corporate logo on the display screen of a DRE machine, I wonder if we might want to place some limits on the nature of product markings that aren't on the display screen. I note that the Data-Punch and Votomatic punched-card voting fixtures, the ES&S Optech Eagle, and most other voting systems I've seen have the vendor's trademark and name clearly displayed on the face of the machine.

I suspect that the vendor's name and the machine make and model should be visible to the voter, but that this information should be presented without trademark or other commercial content. I don't care whether this is on-screen or off, except to the extent that it may steal 'real-estate' on the screen that may have better uses. Aside from this, I believe that no information should be displayed on the working face of the machine that is not required by the voter or poll-workers in the routine use of the machine from the front. All other information belongs elsewhere on the machine. Power requirements and caution notices, for example, go on the back by the line cord, and much of the maintenance schedule belongs on the back or even on the inside of the cover (where the outside maintenance label says "open cover and read the additional instructions).

3.4.18 Human Engineering-Controls and Displays

Item e prohibiting flickering displays at rates between 2 and 55 Hz touches on an important issue, but 55 is an awfully high level. Standard motion picture film is projected at 24 frames per second, and broadcast television uses 30 full-frames per second (interlaced, made from 60 half-frames per second). Motion picture projection at 24 frames per second is marginal (you can frequently see the individual frames in certain action scenes shot with short shutter times so the individual frames aren't blurred by the motion), but if the motion picture shows only static scenes or objects moving at a modest pace, you don't notice the flicker at all.

4 Software Standards
4.1 Scope
4.1.2 Software Sources

The final line "unmodified COTS software is not subject to source code examination" opens dangerous loopholes. I remember one system examination where the vendor revealed (at the examination) that the DRE voting system was based on Windows 95. This was at the time that Microsoft was vigorously defending itself in court, and had just made the assertion that Internet Explorer was an integral part of Windows and could not be removed. The vendor of the voting system, in answer to my question, said that all internet features of Windows 95 had been removed from their system.

So, who do I believe, Microsoft or the vendor? Furthermore, the vendor asserted that the version of Windows they used was unmodified COTS software, and therefore not subject to source code audit. Yet they had done something to it, and nowhere in the Wyle Labs report did it mention any audit of the way Windows was installed on the voting machine. I don't believe the exclusion of unmodified COTS software in the old FEC standards is that different from the current draft, so I suspect that the same discussion would emerge under the new standard, as written.

Therefore, I ask that the audit of systems including unmodified COTS software include an audit of the way in which that software is configured and installed. This audit should include an explicit audit of the option settings used in the software installation, and an explicit audit of the modules that were included in the installation or of the way in which problematic components were excluded.

By the way, COTS belongs in the glossary in Appendix A.

4.2 Software Design and Coding Standards
4.2.2 Software Integrity

The prohibition of self-modifying code was present in the 1990 standard, and it has proven to be ineffective because the software auditors interpreted the prohibition very narrowly. Here is an example; I quote from an exchange between myself and officials at ES&S from late 2001, an exchange that was part of the Iowa examination of the current firmware release for the model 100 precinct count system:

The firmware change log for upgrade of ES&S PBC 100 firmware to version 4.04 indicated: "Added support for reading PCMCIA modem communication utilities from the election definition PCMCIA SRAM card."

I responded (in my E-mail commentary on the change log) : "This worries me. How much software is on the election definition card. My impression has been that the election definition card is just data, but if it's capable of being used to load software, it raises some red flags because this means that the election definition cards end up being far more critical elements in the security of the system than they had been."

Steven M. Bolton, VP of product management at ES&S responded: "This was a temporary fix due to memory allocation and has subsequently been eliminated. This placed the drivers of the modem on the PCMCIA card to save room for the compiled source code of the firmware on the system. It did not raise any concern at Wyle as it was only placing 'off the shelf' modem drivers on the card that were read only during the modem transmission."

I would assert that the loading of an I/O driver from the election

definition card at the time the polls are opened is an excellent example of self-modifying code, and should have been forbidden under both the 1990 standards and the proposed draft standard. Neither ES&S nor Wyley Labs read the standard this way, and this worries me! Where do they draw the line? How do they define self-modifying code.

Here is my working definition:

Self modifying code is code that changes any code resident in the system by any means after the time the application begins execution.

This wording would forbid all dynamic loading of executable code into the memory of the system. If we wish to allow such self-modification, we ought to be doing so using point-by-point exceptions to the broad prohibition in 4.2.2, not by permissive readings!

The prohibition of self-modifying code is related quite strongly to the prohibition against the inclusion of compilers in 6.4.e, so a cross-reference may be appropriate. The only way a resident compiler or assembler can threaten the integrity of the voting system is if the compiled or assembled code can be run; running such code requires a loader, and all loaders can be viewed as a form of self-modifying code.

Therefore, I recommend that the prohibition against self-modifying code be changed to a prohibition against self-modifying, dynamically loaded or interpreted code except under the security guidelines outlined in Section 6.4.e.

4.2.3 Software Modularity and Programming

This set of restrictions bothers me because it is exactly the kind of restriction I'd have placed on a classroom assignment in an upper level undergraduate computer science class in the second half of the 1970's. Nothing in this standard reflects progress in the 25 years that followed.

4.2.3.a Each module shall have a specific function ...

Item a asks for a specific function, and that is good, but there is no demand for clear documentation of intermodule communication. Certain aspects of the code audit can be greatly simplified if there is clear documentation of the relationship between modules, and some of these can be simplified even more if there are enforcement mechanisms such as firewalls to prevent all but the documented information from being shared between modules.

There are several firewall technologies, including scope rules at one extreme, the use of separate processes under a secure operating system, or at the other extreme, use of a separate computer system withing the voting machine. The standard should not demand the use of any particular firewall technology, but it should at least invite clarity about the use of any such technologies.

4.2.3.b Each module shall be uniquely and mnemonically named ...

This section appears to assume that modules are stored in distinct source files. The standard should not mandate this!

First, I understand why the use of separate source files for distinct modules is essential in languages like C and C++. In part, this is

historical -- compile times for large programs in the early 1970's, when C was developed, were very slow, and text editors were inconvenient for all but the smallest files. In part, however, the use of separate source files in C and C++ serves an important function. It allows strict control of data communication between modules. Items declared in a file can be made local to that file by declaring them static, and this mechanism allows strict control over the flow of information between modules, a requirement that supports paragraph a.

Second, languages that force each logical unit into a separate file frequently mask the top-level structure of the program. I've had this problem with large C, C++ and Java programs, where the construction of the program as a myriad of little files leaves me, as a reader of the program, almost completely in the dark about where to look for a component referenced from within one part of the program.

Therefore, I'd like it to be made extremely clear that the nature of the unit of program structure being referred to as a module is flexible. Any file or named syntactic unit within a file may be considered a module.

4.2.3.c ... relatively small and easily identifiable modules ...

The general requirement that modules be small and easily identifiable is quite reasonable, but the specifics of this section seem gratuitous. I've read enough software audit notes from Wyle Labs and enough change notes from developers responding to the audits to be sure that the rules like this from the 1990 version of the standard are being enforced mechanically, without sensible thought.

It does not always clarify a large module to break it into small pieces! Sometimes, this only makes the big picture that much harder to digest. I have a considerable amount of experience with large programs where there are regular code structures where the regularity is the key to understanding the code. Where this occurs, breaking the code into separate modules hides the regularity and therefore prevents the reader from quickly grasping what is going on.

Has anyone seen a software coding standard that focuses on real readability and not on gratuitous figures of merit such as lines of code? We need one here!

4.2.3.d ... a single entry point, and a single exit point ...

This requirement has side effects that directly contravene Section 4.1.1 and 4.2.1! This difficulty is admitted in Section 4.1, but the extent of the problem appears, to me, to be greater than anticipated in this section.

Specifically, the emphasis on control flow makes this section inapplicable to pure logic programming and to pure declarative languages.

The requirement that each module have a single entry and exit interferes with the effective use of object oriented programming languages (C++ and Java being the best known). In these languages, the notion of module is best mapped to the concept of class, and classes can be most effectively used if each class instance hides a key item of information and provides several methods to the outside world used to manipulate that information.

Even classic C can be used in a semi-object-oriented fashion by gathering a group of functions into one module that represents a class instance, where the non-static functions of the module serve as methods of that instance, and the static data in the module represent hidden or private components of the class instance. This methodology is effectively forbidden by a literal reading of this section.

There is value in having each method, function or procedure found in a module have one entry and one exit point, and it may be that this simple fix is sufficient to repair the damage done by the wording of this section, as it appears in this draft.

4.2.4 Control Constructs

Again, this set of restrictions bothers me because it is exactly the kind of restriction I'd have placed on a classroom assignment in an upper level undergraduate computer science class in the second half of the 1970's. Nothing here reflects progress in the 25 years that followed.

I've read enough software audit notes from Wyle Labs and enough change notes from developers responding to the audits to be sure that the rules like this will be enforced mechanically, without sensible thought.

Has anyone seen a software coding standard that focuses on real readability and not on gratuitous restrictions on the set of constructs that may be used? We need one!

With specific regard to Figure 4.2 regarding if-then-else, I believe we ought to permit the unlimited use of else-if clauses, so long as the conditions tested on each if or else-if are parallels. As currently written, the standard easily allows the following C construct under the rules for Figure 4.5 (case):

```
switch (i) {
        case 1: ----------- ; break;
        case 2: ----------- ; break;
        case 3: ----------- ; break;
}
```

But, read the way I'm used to hearing about Wyle Labs reading code, it does not seem to permit the similar form without incurring a penalty for deep nesting:

```
if        (strcmp( s, "one" ) == 0) {
        ------------ ;
} else if (strcmp( s, "two" ) == 0) {
        ------------ ;
} else if (strcmp( s, "three" ) == 0) {
        ------------ ;
}
```

In fact, these express identical logic except for the change of domain from traditional integers to textual representations of the names of the integers.

A second restriction that worries me is the restriction on loop exits. Multi-exit loops are forbidden, and there are many cases where multi-exit loops clarify the structure of code.

Also, the set of allowed control structures does not include any

sane exception handling mechanism. There are tools in standard Pascal and in the standard C library that allow for reasonable exception handling methodologies if they are not abused. Ada, and Java have built-in exception handling mechanisms that are quite civilized, and these can be simulated very closely in Pascal and C; the Pascal/C simulation of Ada/Java exception handlers requires that exceptions exits by a non-local goto (Pascal) or longjmp (C) to the end of the block that armed that exception. In fact, all exception handlers are equivalent to such exits.

See http://www.cs.uiowa.edu/~jones/syssoft/style.html#except for a C programmer's manual of style that includes some documentation of tools for exception handling in C that closely mimic the tools that an Ada or Java programmer would expect.

4.2.6 Coding Conventions

See http://www.cs.uiowa.edu/~jones/syssoft/style.html for a C programmer's manual of style that I wrote for my students several years ago. I don't hold it up as an example of perfection, but just as an example of a reasonably good coding style that I've developed over many years of practice. My comments on the specific conventions that follow are grounded in this experience.

4.2.6.c Have no line of code shall exceeding 160 columns ...

Item c is garbled, and the 160 column limit seems generous. While punched cards are long gone, the 80 character line width is still common, and for good reason. Readable text on the page of a book is rarely much longer than this, and if our goal is readability, we ought to ask for something similar. Furthermore, most people's screens aren't large enough to comfortably display lines over 100 characters long in a readable font. Therefore, I suspect that neither programmers nor code auditors are well served by permission to use such long lines!

In my own programming, I rarely exceed an 80 character line, and I can easily live with an absolute 80 character upper bound on the line length. If the limit must be longer, I'd rather see it raised to 100 or 120 characters.

4.2.6.c For each line of source code ... A function call inside an if ...

The final sentence of this paragraph, starting "a function call inside an if condition ..." is too limited. The same considerations that justify careful attention to function calls within the conditions on if statements also justify such attention to function calls in the expressions governing for, while, and select statements. Therefore, I recommend rewording the final sentence to begin: "A function call within the expression governing a conditional or iterative control structure ..."

Also, I would recommend a special case to permit the C, C++ construct:

```
if ( count == final ) break;
```

Combining if and break in this way should be counted as a single primitive control element structure, in keeping with the spirit of figure 4-6, despite the fact that C counts the break element as a kind of statement. This exception should also apply to raising an exception in those settings where civilized exception handling primitives are available.

4.2.6.e If() statements shall ...
4.2.6.f If() statements shall ...

Items e and f are duplicates in the HTML version of the code.

Also, items e and f are really applicable only to Pascal, C, C++ and Java. Other languages, such as Algol 68 and Ada don't have this problem. Furthermore, the same problem applies to for and while statements in C. Therefore, I suggest the following wording:

In programming languages where control statements have no explicit terminating delimiter (for example, the if, for and while statement forms in Pascal, C and C++), an explicitly delimited block should be used.

The example in part e can remain as is.

However, 4.2.6.d seems to allow the following one-line form:

```
if (flag == true)  counter = counter + 1;
```

This does not offend me, and I would prefer to further amend the standard to give explicit permission to use this form! This leads me to the following final proposal for the wording to use here:

In programming languages where control statements have no explicit terminating delimiter (for example, the if, for and while statement forms in Pascal, C and C++), an explicitly delimited block should be used whenever the entire control statement cannot be expressed in on a single line of code under the guidelines of paragraph d.

4.2.6.h Avoid mixed-mode operations ...

This used to mean something; it makes me think PL/I or perhaps Fortran, but I haven't written a PL/I or Fortran program for 20 years. What is this intended to restrict? Does this refer to mixed types within an expression, for example, integer mixed with floating without explicit conversions between one and the other? If so, say so!

4.2.6.i The program may exit() at any point ...

First, this makes specific reference to a construct of C and C++; program termination is the general term, and it is accomplished by other means in other languages.

Second, programs with multiple termination points cause exactly the same class of problems as functions with multiple exits. The intent of 4.2.3.d is obviously to exclude this, and I'd hold that if you permit multiple termination points in a program, you must also permit multiple exits from a function.

Third, this is written under the assumption that the voting application is one program. In fact, if a civilized operating system such as Windows NT or a UNIX derivative is used, the voting application may be structured as several concurrent programs, each able to exit independently. In fact, a function that must execute behind a firewall should be started as a parallel process on these operating systems; in this case, it will return to the "calling process" by termination.

Usually, all but one of the multiple termination points of a

program represent exceptions, termination because of error, while there is usually only one point for successful termination. Therefore, multiple termination points should be considered as a special case of raising exceptions, and as I've pointed out in my comments on Section 4.2.4, the coding standards don't cover the use of exception handling mechanisms of any type!

4.2.6.l Have functions with fewer than six levels of indented scope ...

This is a gratuitous constraint that can force readable routines to be broken up into less readable families of routines. I agree that deep nesting can lead to unreadable code, but my experience with reading code audits from Wyle Labs suggests that, to date, the auditors have been very rigorous in their application of standards like this, without obvious attention to real issues of readability.

4.2.6.m Initialized every variable

There's a small grammar problem here. The other requirements are in present tense; this one slipped into past tense.

Not all initialization is meaningful! True, uninitialized variables are a potential source of error, but gratuitous initialization can distract from readability by leading the reader into a false sense of confidence.

I'm concerned that the requirement, as worded, would lead the reviewers of C code, for example, to require that every variable be initialized at its point of declaration. Given the Wyle Labs source code reviews I've read, this is a very likely interpretation, and the likely response, in order to get the code to pass the review, is to simply add explicit initialization to meaningless values like 0 instead of trying to argue with the reviewer that the correct initial value is computed later in the code.

I would reword this requirement as follow: Initialize each variable unless the correctness of the code does not depend on the initial value of that variable. Each such exception should be clearly documented, and in those cases where the initial value is not provided at the point of declaration, clear documentation should lead the reader to the point where it is initialized.

4.2.6.n For all if() statements, be implemented with comparisons ...

First, this requirement ought to apply to the boolean conditions governing not only if statements, but also loop exits that are controlled by while or until clauses.

Second, this requirement is pointless in languages like Pascal and Ada where the condition on the if, while and until statements are constrained to be Boolean. It serves a useful and I believe necessary purpose in C, C++ and related languages because the type checking rules of these languages allow integers wherever a boolean value is expected.

Therefore, I suggest the following wording:

In programming languages where there is no distinct Boolean data type, C and C++ for example, all Boolean tests governing control flow, for example, in if and while statements, shall be implemented using explicit comparison.

4.2.6.o All constants other than 0 and 1 shall be defined or enumerated

This is another example of a rule that is easy to state and well

intended but that fails in its intended use. There are many cases where 0 and 1 must be defined. The classic one in C and C++ is:

```
#define 0 FALSE
#define 1 TRUE
```

I would strongly suggest that a program that uses 1 and 0 to mean true and false should be considered to be evil and forbidden despite the fact that it is perfectly legal to do so in C, C++ and related languages.

At the same time, there are other constants that should be preserved in naked form. For example, consider:

```
average = (a + b)/2
```

Here, the reason for the constant 2 is self evident, and any added commentary will only detract from readability by wasting the reader's time.

4.2.6.p In C and C++, only the minimum implementation ...

I would go farther and forbid the "b?c:d" form in voting software, except to the extent that it is used in many parts of the C and C++ standard header files.

A list of evil features of C and C++ should also exclude the comma operator, again excepting its use in the standard C and C++ header files. Most C programmers don't even know they can write:

```
a = (b = c, d);
```

Anyone who really knows C will know that this can be rewritten as:

```
b = c;
a = d;
```

Anyone who doesn't know C extremely well, however, will be really baffled!

The restriction in item 4.2.6.n is really in this same class, since it deals with yet another flaw in the C family of languages.

In general, the C family of languages, including C++ and Java, is full of features that can be abused to write truly horrible code that defies understanding. Prior to 1980, the C family of languages could have been stopped and the world could have been persuaded to use Pascal, Ada, Modula and languages based on them, unfortunately it is too late now, and we must live with C and its descendants. Even Bjarne Stroustrup, the developer of C++, has commented that most of the least desirable features of C++ are there because of the need for C compatibility.

4.2.7 Comments Conventions

The name of this section seems to violate some rule of grammar. "Commenting conventions" or "Comment conventions" both sound better.

4.2.7.a All modules longer than 10 lines shall contain headers ...

Item 2 presumes that modules are callable. In fact, in the case of object oriented programming, the class is the natural item that I would be tempted to identify with the concept of module, and you don't call a class (except that in some object oriented languages, the call to a class name causes an instance to be instantiated).

A more appropriate item 2 would require header comments to state the dependencies of this module has on other modules and the nature of that dependency.

Item 2 includes the phrase "and the calling sequence". If this refers to the calling sequence for calling some function or procedure in this module, that is part of item 3 and not logically part of "other units called".

Item 3, a description of input parameters and outputs, varies in importance from language to language. This is crucial in assembly language, where there is no standard calling sequence and no requirement inherent in the language that parameters be documented in any way. In languages such as Pascal, Ada and C++, the language requires significant documentation of the calling sequence and parameters, to the extent that careful use of function names, type names and parameter names can eliminate the need for most header comments.

Item 5 should be extended to require not only documentation of the global variables used but the nature of the use, for example, is the global variable inspected, inspected and then written, or simply written, disregarding the previous value. This corresponds exactly to the documentation requested in item 4 for file references, and in fact, files really should be considered to be logically just another kind of global variable.

4.2.8 COTS Software

This requirement is insufficient. Not only should changes to COTS components be documented, but the nature of any changes to the default configuration options should be documented. See my comments about Section 4.1.2 for an example.

5 Telecommunications
5.3 Prohibitions

Item d, system programming and software installation, can be interpreted as a prohibition against the use of telecommunication in program development. I'm not sure this restriction is reasonable.

Item f, official election results, is also problematic. I believe we can do this with today's technology, although I've only seen a few voting systems I'd trust to do this in a secure way. Most that advertise secure services are not, in fact, up to snuff.

In sum, I believe that the prohibitions here are excessively conservative, so much so that they render most of Section 6.6 and the second paragraph of 9.2.1.4 irrelevant. I'd like to see a clause allowing exceptions to this prohibition! The mechanism to allow these exceptions should place the burden of proof on the vendor. Consider the following final paragraph:

An exception allowing the use of telecommunications technology for one or more of the above functions may be allowed, providing that the vendor can demonstrate that the necessary technology

exists and that this technology is used effectively. The documentation required to obtain such an exception should be described elsewhere, probably somewhere in Section 6.

6 Security Standards
6.2 Access Control
6.2.1 Penetration Analysis

This is an excellent requirement, except -- as the chair of a state board of examiners, I want more than the mere assurance that the ITA has seen a penetration analysis.

I would not urge the penetration analysis to be disclosed in full to the boards of examiners for voting machines in all 50 states, because each such board ought to include at least 3 people, and disclosure of such a document to 150 people, even if each was bound by a nondisclosure agreement, is something I'd be troubled by.

But, if the ITA report includes commentary on the adequacy of the penetration analysis and on the sufficiency of the measures taken by the system in response to each identified threat, I'll be satisfied if this ITA commentary is disclosed to each state.

We've had a bit of a struggle here in Iowa to allow the ITA report itself to be disclosed to the board of examiners for voting machines without turning it into a public document -- essentially all other paper that comes before the board becomes public record. I'd hope that the vendors would disclose their ITA reports, including the report on the penetration analysis, to any state willing to agree to similar conditions.

6.4 Software and Firmware Installation

The borders of item e are fuzzy! It is appropriate to implement some voting functions by interpretive mechanisms, and whatever can be implemented by an interpreter can be done by a compiler, and visa versa.

Furthermore, some interpreted languages will be very difficult to exclude from the system. Most notable among these is Postscript. The Postscript printer control language is, it turns out, based on a general purpose programming language, Forth. As a result, every postscript printer can be used to execute Forth programs! Similarly, Microsoft's Office suite allows Visual Basic programs to be embedded in any Microsoft Office document.

It is also worth noting that any controls in the standard governing the inclusion of compilers or assemblers (and by extension, as I have suggested, interpreters) relates to the material in Section 4.2.2 governing the presence of self-modifying code.

I personally recommend that interpretive mechanisms be allowed, and having recommended this, I see no reason to forbid compiling and then loading of code that otherwise might be executed by an interpreter. To prevent this from causing problems, I recommend that each such mechanism be protected. There are 2 basic ways to provide such protection:

1) The language supported by any interpreter or compiler that is resident during system operation should be verified to be incapable of expressing programs that could compromise the correct operation of the voting system. Thus, an interpretive drawing language used to render the logos of political parties on the screen can be permitted if the language is unable to perform

any operations other than drawing on the designated area of the screen.

2) If the above cannot be done, the language implementation can be run in its own protection domain; such domains are also called sandboxes and the fence around such a domain is sometimes called a firewall. Within this domain, programs in the language are not limited in what they can do, but the domain boundary makes it impossible for the program to manipulate any resources other than those it is permitted to use. The technologies for enforcing the boundaries around such a protection domain are old -- See Anita K. Jones's PhD thesis done at Carnegie Mellon University around 1973.

6.5 Telecommunications and Data Transmission
6.5.4 Protection Against Malicious Software

This entire section is far better than its predecessor in the previous revision!

6.5.4.2 Forms of threats

This entire section is still packed with poorly defined terms that, in many cases, do not properly belong in Section 6.5. Furthermore, many of these specific threats are dealt with, apparently unknowingly, by other parts of the standard. Cross references are needed to make this clear.

6.5.4.2.a File viruses ...

The constraints on the use of self-modifying code in 4.2.2 and the inclusion of compilers and assemblers in 6.4, if interpreted broadly, prevent inclusion of mechanisms in the voting system that would allow the execution of any files. This eliminates the threat posed by virus included in executable code.

6.5.4.2.b Macro viruses ...

The constraints on the use of interpreters that I advocate in my notes on Section 6.4 would effectively eliminate the threat of macro viruses because these rely on an interpreter for a macro language.

6.5.4.2.c Worms

Worms also rely on the ability to inject executable code into a system, but unlike viruses, they do not attach themselves to files, but rather, exploit mechanisms in the communication software that allow injection of code into the machine via a communication channel. See Shoch and Hupp, "the Worm Programs", Comm ACM 25(3) 172-180, 1982 for the defining work on worms. Formally, the prohibition against self-modifying code from 4.2.2 should be read as including a prohibition against including any software that can load code from a communications line, and this reading prohibits worms! It is noteworthy, however, that malicious worms on the Internet have exploited loopholes in the design of network software that could be used to inject code into a system but were never intended for that purpose. In sum, the primary defense against worms lies in a rigorous audit of all network interface software to assure that all such software is free of the potential to allow violation of the constraints documented in 4.2.2.

6.5.4.2.d Trojan Horses
6.5.4.2.e Logic Bombs

Neither of these threats is a network threat, except that it is correct to classify all viruses as a category of Trojan horse, in the sense that the infected file enters through the gateway to the system as something desired, but it contains, within it, a hidden threat. The aspect of this threat that makes it a virus is the fact that the threat includes the ability to convert other files into carriers for the same threat.

But, Trojan horses in the world of software (or hardware!) are system components (files, programs, routines) that come from a third party (Athens) and are included as parts of the system (Troy) without careful inspection. They serve useful purposes (for example, by being beautiful), while in fact, they contain hidden threats (Athenian soldiers). Any insufficiently audited third-party component may serve as a Trojan horse. The only defense against such attacks is a thorough audit of all third-party components.

It is worth noting that so-called Easter eggs in software are a class of Trojan horses. An Easter egg is a benign Trojan horse, typically an unauthorized feature that is included in a piece of software without the approval of the vendor purely as an expression of pride, whimsy or playful spirit on the part of one or more programmers involved in developing that software. The most famous Easter egg was probably the flight simulator game that was included in Microsoft Excel 97. The fact that Easter eggs are as prevalent as they are in commercial off-the-shelf software is strong evidence that the accepted industry standards for protection against Trojan Horses in the development of such software are extremely weak.

Logic bombs are very similar to Trojan horses, except that they are typically included in a system by insiders. A disgruntled employee may add a logic bomb to any software product. Again, the defense against logic bombs lies in the audit process, not in the communications interface!

In fact, malicious viruses and worms can generally be described as viruses or worms that include logic bombs. Without such a bomb, the virus or worm merely copies itself from file to file or from system to system. Such simple viruses and worms are dangerous only because they consume resources. If the virus or worm also carries a logic bomb, it becomes a targeted weapon that can attack specific information or functionality in any system that executes its code.

6.5.4.3 Use of Protective Software

This section is reasonable, no matter how Section 6.5.4.2 is rewritten.

6.5.4.4 Monitoring and Responding to External Threats

Again, this section is reasonable, and the voting-specific procedures at the end, in paragraph f, are excellent.

I do have one quibble. If I was paranoid, I'd demand that there be no change to the voting software after the date that the ballot for the election is determined. Trojan horses designed to respond in a partisan manner to an unknown ballot are difficult to design, but once the ballot is fixed, it becomes far easier to design a Trojan horse that responds to the specifics of the ballot to achieve a particular outcome in the election.

Given the slow speed of the software audit process, and given the

fact that the community of programmers has become quite expert at including Easter eggs (benign Trojan horses) in commercial software, I find it very hard to imagine a timely response other than simple disconnection form the communications network to a new threat that is discovered after the names of the candidates on the ballot are determined.

6.6 Security for Transmission of Official Data ...

Section 5.3 seems to prohibit quite a bit in this domain. Any reason for prohibiting the transmission of official election results would appear to apply equally well to the transmission of individual official ballot images!

Where a central count DRE machine would transmit ballot images to a central counting system as each vote is cast or when the polls close, a precinct count DRE machine would transmit just the official totals. Any corruption of the election results that could be done by corrupting one of these transmissions applies equally to the other, and the appropriate countermeasures for defending against such corruption are the same!

The safeguards documented in the remainder of this section represent a reasonable start toward meeting these goals, although in detail, they presume what I'll call the classical E-voting model (central count with ballot transmission normally done at the moment each voter commits a ballot).

9 Overview of Qualification Tests
9.2 Testing Scope
9.2.1 Test Categories
9.2.1.4 Focus of Security Tests

The second paragraph covers functions that are effectively prohibited by Section 5.3. My proposed change to 5.3 would allow this function and therefore make this paragraph relevant. Also, this paragraph should cite Section 6.6 where there is a list of many threats to the security of the system.

9.2.1.5 Focus on Integration Tests

The acronyms PCA and FCA are not defined here or in the glossary (Appendix A). I guess from context that these are audits of some kind, but I can't figure out anything more. Curiously, Section 9 included a nice refresher on the meaning of ITA.

9.2.1.7 Tests of Ballot Counting Accuracy

There is no specification of the nature of the test deck or test entry data. Under the 1990 standard, this omission has allowed the use of test punched-card ballots punched on keypunches instead of using the Votomatic fixture, and it has allowed the use of test optical-scan ballots mechanically printed with test patterns using only the prescribed mark. In sum, the testing has served only to test the logical function of the software and the physical durability of the input mechanisms.

This testing has not explored the marginal behavior of the voting systems! As I pointed out in Section 3.2.5.2, we need the vendors to document the marginal behavior of their systems, and having said this, I believe that part of the qualification testing should be to verify that the margins for the system are where the vendor said they were.

Thus, tests should include not only testing with the prescribed

mark or the prescribed punching or the prescribed pressures on the centers of sensitive areas on the screen, but the tests should also explore sensitivity to nonstandard marks, nonstandard punches (the classic Votomatic case where the stylus pierces the chad without removing it) and screen touches on the margins of the sensitive area or using marginal force.

9.3 Applicability
9.3.1 General Applicability
9.3.1.2 Software

There is a typo in the first line of paragraph a of the HTML version. "com-mencing"

As pointed out in my comments on 4.1.2, we need to be careful about COTS software. I have documented a perfectly feasible attack on the voting system that could be written as part of the window manager. As such, this is a Trojan horse attack on the voting system that could be carried out by the vendor of a COTS window manager.

It is noteworthy that the most widely used COTS window manager today comes from a vendor that is involved with antitrust litigation and has expressed strong opinions about who they wanted to see win the most recent presidential election. Under the rules currently in place, our standards leave the door wide open to the possibility of unethical actions such a vendor may be tempted to undertake.

See http://www.cs.uiowa.edu/~jones/voting/taubate.html for details.

As a result, I strongly urge that the system software that sits in the path from voter to voting application and from voting application to official canvass be subject to 9.3.1.2 unless it can be shown that defensive measures have been taken to prevent that software from interfering with the correct behavior of the system.

As an example of such a defensive measure, consider a file system. I would hold that a COTS file system should be subject to audit if it is used to store the plaintext of ballot images or other critical election data, but I would also hold that simple encryption of the ballot images is probably sufficient to exempt the file system from audit.

I have not figured out an equivalent simple countermeasure against the window manager attack I cited above!

9.3.2 Modifications to Qualified Systems
9.3.2.3 Utility Software and/Device Handlers

The slash in the title might have been "and/or", but I can't tell for sure.

In any case, as I pointed out in my comments on Section 9.3.1.2, this exception ought to apply only to those software components that are incapable of carrying Trojan horses that might attack the voting application.

9.5 Qualification Test Process
9.5.2 Qualification Testing
9.5.2.6 Witness of System Build and Installation

This is an important section! Good!

But should this be connected to the software escrow? I've long suspected that much of the software being escrowed today verges on being useless. If we make an escrow copy, this should be made at exactly this point, in front of witnesses who can assert that the escrow copy is indeed the version that was built.

A Glossary

There are more than a few acronyms used in the body of the report that belong here. I've pointed out PCA and FCA, for example.

Detailed Comments on Volume II

2 Technical Data Package

As I've commented several times with respect to Volume 1, notably Section 3.2.5.2, all paper-based systems must have, as part of their technical documentation, a clear statement not only of the prescribed mark (which may be a hole in a punched card) but also fairly complete documentation of the ranges of marks that are reliably detectable, reliably ignored, and marginal (occasionally detected and occasionally ignored). Only if these are clearly documented can we reasonably hope to determine how well the machine conforms to state laws concerning what marks should and should not be counted! Furthermore, the ITA really ought to verify that the mark documentation is accurate.

If the mark sensing thresholds are adjustable, there needs to be additional information about how they are set and about calibration procedures.

I can't figure out where in this section this information should be disclosed. It isn't hardware documentation (mark sensing thresholds may be set by hardware or software or some combination). Perhaps it ought to be part of the system description?

This may also need to be referenced in Section 3. I can't figure out where in this section this information should be

2.9 System Maintenance Procedures

In my investigation of Votomatic voting systems after the 2000 general election, and in conjunction with reporters for the Palm Beach Post and Fort Lauderdale Sun Sentinel, I helped uncover significant problems with the maintenance procedures for Votomatic and more particularly DataPunch punched-card voting fixtures.

See http://www.cs.uiowa.edu/~jones/cards/chad.html for details.

This experience leads me to conclude that the vendor's recommended maintenance procedures, specifically the procedures for removing chad from a punching fixture after each election, should have been subject to significant independent testing. What we found was that the instruction "shake well to remove chads" (the vendor's instruction) was insufficient to prevent the accumulation of chad jams behind some ballot positions, and these jams could accumulate to the point that they made it impossible to vote those ballot positions!

3 Functionality Testing

There should be a reference here for paper-based systems

requiring the testing of the documented range of marks that are reliably sensed, reliably ignored and marginal (neither reliably sensed nor ignored). See my comments on Section 2.