COMMENTS ON THE

AUGUST 31, 2007 DRAFT OF THE

VOLUNTARY VOTING SYSTEM GUIDELINES

SUBMITTED MAY 5, 2008 TO THE

UNITED STATES ELECTION ASSISTANCE COMMISSION

Cem Kaner, J.D., Ph.D.

Professor of Software Engineering

Florida Institute of Technology

kaner@kaner.com

Member:

Election Assistance Commission
Technical Guidelines Development Committee

I submit these comments in my capacity as a TGDC member, nominated to TGDC by the Institute of Electrical and Electronics Engineers (IEEE). My analysis of this draft was greatly assisted by the IEEE Standards Coordinating Committee 38 (Voting Standards). Except where specifically noted, comments that follow reflect my own views and recommendations and should not be interpreted as positions of IEEE or IEEE SCC38 or of my home institution, Florida Institute of Technology.

Most of my comments address testing-related or human-factors-related issues. This narrowness is intentional. I am widely regarded as an expert in software testing, with strong grounding in real-world practice. I also hold a doctorate in experimental psychology, with significant experience in user-oriented design.

OVERVIEW

Strong doubts about election outcomes are expressed routinely in the American media. Serious concerns have been raised about the accuracy of vote counts from all types of electronic voting systems, including touch screen and optical scan systems. Definitive recounts and audits have been impossible, largely because of limitations of the systems. The result has been a growing cynicism about one of the foundations of our democratic society, the free election.

Even though the current draft of VVSG requires many improvements in voting systems, the most important theme underlying my comments is that there is a relatively straightforward way to do much better: *Rather than relying primarily on independent test labs for certification of the quality of electronic voting system software, VVSG should require vendors to make their non-COTS source code publicly reviewable and require test labs to make their test documentation and test results publicly reviewable. The intent is to strongly supplement the work of the independent test lab, not to replace it.*

This proposal would mitigate three critical problems:

1. The primary quality-control (QC) work for software must be done by the creator of the software. The highly formalized end-of-development verification by an independent test lab is for confirmation that product development (including thorough vendor quality control) has been complete and successful. The lab's work is too expensive, too rigid, and too late to achieve much else. *Unfortunately, there has been too much evidence over the past few years that vendor QC has too often been seriously deficient.* Repeated revelations of major flaws on national television undermine the public's confidence in a system that must be trustworthy and trusted.
2. The VVSG is demanding information from test labs that they are not well suited to provide. For example, it demands a highly creative security audit, in the form of "open-ended vulnerability testing" (OEVT) from organizations whose best skill is routinization and documentation of tests tightly linked to unambiguous specifications. It also demands reliability estimates from organizations who do not have, and cannot cost-rationally obtain, the type of data that could provide such estimates.
3. No system that contains software is perfect, and no system is perfectly secure against all imaginable threat scenarios. Even if the underlying quality of voting systems were to improve so much that the only demonstrable threat scenarios are genuinely implausible, the foundation of cynicism and mistrust that has been laid over the past decade leaves American society vulnerable to misimpressions caused by demonstration of genuinely-irrelevant flaws. Because voting system software and test results are shrouded in secrecy, it will be impossible to reassure the public that demonstrable-but-implausible weaknesses are not the tip of an iceberg hidden behind nondisclosure agreements. *Accuracy, and the level of security needed to protect that accuracy, are not enough. If the public does not believe in the accuracy of election results, the elected government loses its perceived legitimacy.*

As I have evolved and circulated this proposal over the past year, several concerns have been raised. Let me acknowledge and answer them here:

1. *Vendors and test labs consider their work products proprietary and will not want their trade secrets revealed to the public.* In a marketplace, customers and sellers negotiate to achieve a result that is mutually acceptable. The voting equipment market is very large. If current vendors cannot bring themselves to disclose their technology at any price, there is no shortage of other American companies who have many years of experience embedding software in life-critical or business-critical products, who would be glad to enter this market. It may be that (at least over the short term), the price of voting equipment would have to rise to compensate vendors for the value of their disclosures. *How much is it worth to make this problem go away?*
2. *Reliance on testing by the public, as with open source software, is insufficient to guarantee achievement of a trustworthy result.* I agree. Public review should enhance, but not replace, the work of the test labs.
3. *There is no mechanism for dealing with problems exposed by public review. There is no guarantee that these will impact the use or upgrading of the systems tested.* Test results reported by members of the general public may or may not be valid and may or may not reflect serious underlying problems. The results provide data that can lead to improved future performance by vendors and test labs and better decision-making by the purchasers of voting equipment.

4. *Open source projects often find it very difficult to entice the public to thoroughly test products under development.* The difference between a voting system and products like Firefox and OpenOffice is the enormous public interest in voting systems. For example, the National Science Foundation considers public interest ("broader impact") in its assessment of every proposal that it reviews. Dissertations and other controlled research often assess software engineering methods against some types of real-world products. Given access to voting software and voting machines, assessment against such a system would be a popular choice—no one could criticize it as unrealistic or as having limited interest. Along with traditional research funding, it seems highly likely that several nongovernmental organizations would fund ongoing assessments of voting equipment. Imagine a fundraising campaign by *MoveOn.Org* to support testing of voting equipment: "We must expose holes in voting equipment before they are exploited by evil right-wing politicians." How many millions of dollars would such a campaign raise in its first week? And after MoveOn's first advertisement, how long before comparable fundraising campaigns from many other parts of the political spectrum?
5. *This would be unfair to vendors because it would force them to surrender their intellectual property rights to their technology.* This is a common misperception of this proposal. Disclosure of intellectual property, without loss of protection, is common. Patents can issue only for publicly disclosed inventions. Copyright protects published work from unauthorized republication. Given publicly-reviewable code, if some of manufacturer X's code appeared in manufacturer Y's products, how long would it take X to discover Y's theft? How hard would it be for X to establish prior publication? This proposal strips secrecy from the software but preserves the other intellectual property rights.

I am not advancing this proposal out of any personal agenda that favors open source software. It is true that I am currently engaged in National Science Foundation-funded projects to develop "open source" course materials, but this is a personal choice as my most effective way to achieve a broad instructional impact for a new approach to online professional instruction. My professional history has been much more tightly linked to proprietary software: Before joining Florida Tech's faculty in 2000, I worked for 17 years in Silicon Valley's software industry, as an individual technical contributor, manager, independent consultant, and attorney focused on software-related law. I was elected to the American Law Institute in recognition of my work on computer law, in 1999, a mere 5 years after my graduation from law school. During all of that time, I worked primarily for and with developers and vendors of proprietary software. As an attorney, I often represented developers of software, providers of software testing services, and authors of articles and books, helping them protect their intellectual property rights.

I propose public reviewability of voting system software (non-COTS components only) and test materials only because I see no other way to (a) so significantly increase the trustworthiness of such systems and (b) provide a foundation for public confidence in them. I see both of these as critical success criteria for the VVSG.

SPECIFIC COMMENTS, SECTION BY SECTION

Glossary: COTS

I think this is an excellent definition of COTS and that it should not be changed.

Introduction, Section 1.4 (Structure)

REPLACE the following text:

> "A separate volume of tests will accompany the VVSG in the future. The VVSG contains descriptions for test methods and general protocols for how requirements are to be tested, but does not contain the actual tests themselves."

WITH this text:

> *"The VVSG contains descriptions of some test methods and general protocols for how requirements are to be tested, but does not contain the actual tests themselves. A separate volume of examples of recommended tests will accompany the VVSG in the future."*

Rationale:

It might be useful to create a separate volume of examples of recommended tests, but VVSG should not encourage the notion that any particular finite list of tests of the software would be sufficient to test the voting system. Software defects are more like design defects than manufacturing defects—a software error shows up in every manufactured copy of the same product and discovery of new defects comes from testing the system in a new way, not from testing for the same flaw again in a new copy of the device. Sound software testing practice encourages diversity of testing, rather than focus on a pre-specified set of tests that the software can be optimized to pass.

Introduction, Section 2.2 (Usability performance requirements)

The creation of a testing protocol so standardized that even the test ballot is fully specified runs the risk of design optimization by the vendor in ways that pass the test easily without improving the usability of the system as a whole. A test ballot is only a tiny sample of the population of possible ballots, and it stops being a representative sample when application designers consider it specifically in their designs and test designers know that they will be required to use this ballot in their tests of each version of each voting device they test. Metrics based on such a test are likely to show apparent improvement over time, as systems are optimized toward better performance on this particular test. This improvement may or may not correlate with the underlying level of usability.

Usability testing should include both, tests against a standard protocol and testing against a larger set of randomly varied ballots. In addition, usability testing should consider ballot designs that the voting equipment enables that are not optimal. In practice, weak design by voting officials (the Florida butterfly ballot is an example. As a non-Florida example, consider this one from California: http://voxexmachina.wordpress.com/2008/02/06/call-shenanigans-on-los-angeles-county/). Some designs are semantically confusing—outside of the scope of evaluation of the system that merely presents what is written. But to the extent that a ballot is confusing partially because of its content but partially because of the equipment design, that is a usability issue worth understanding. Usability evaluators should be asking, from system to system, what are the particular details of this system that might make user confusion more likely, and how serious are those risks?

The results of usability testing should be public, and compared from tested version to tested version of the voting device. Trends in these results over time should be carefully considered.

Introduction, Section 2.4.1 (Independent voter-verifiable records)

An IVVR is not a valid record for this purpose unless it is (a) immutable, and (b) sufficiently durable to be available for audits, reviews of audits, and as raw data for reasonably timely election-related research.

For example, an IVVR could be stored electronically if it was stored on a read-only disk, but not on a read-write medium because there is no assurance that the record approved by the voter matches the record stored. Storage via printing on thermal tape probably meets immutability, but not durability.

Introduction, Section 2.4.1 (Independent voter-verifiable records)

When a voter is presented with an IVVR, the record presented must be read off the storage device, so that the voter confirms that what is in storage matches the voter's expectations.

Optical scanners do not do this. Instead, the voter is left to trust that the scanner has read the ballot correctly and stored it accurately.

IVVR might be necessary for software independence but it is not sufficient. Some (probably small) percentage of voters will actually check the accuracy of their individual single records. The election consists of a large number of records that must be stored safely, retrieved completely and accurately, and then read and tabulated accurately, with accurate transmission of the tabulated results and accurate integration of those results into larger pools of results.

Introduction, Section 2.5 (Open-ended vulnerability testing)

ADD the following text to the end of Section 2.5:

> *" Because of the importance of this reliance, OEVT team members must attest under oath that in their professional opinion (a) they have had sufficient time to develop sufficient knowledge of the system and of tools and techniques to investigate it; and (b) their testing was sufficiently competent and thorough to meet the objectives of this phases of testing."*

Rationale:

First, we are creating a task that depends on the expertise of the task-performers, in an environment in which it is very difficult to qualify that expertise. There is no widely accepted credential for testing expertise: there are no degree programs in testing, nor broadly recognized certifications, nor professional society fellowships. At a minimum, we must require the people who claim to be experts to certify that their work has been competently done.

Introduction, Section 2.5 (Open-ended vulnerability testing)

"Open-Ended Vulnerability Testing" is an odd name for the activity described here because this testing is not open-ended. It is tightly time-constrained (closed-ended). The current expectation in VVSG is a period of 12 person-weeks.

The style of testing described for OEVT is normally called "exploratory testing". One of the critical risks of exploratory testing is that it will be done poorly (at best, superficially or focused only on previously well-understood risks) if the tester has insufficient time to learn the particular system and assess its particular risks.

Introduction, Section 2.8 (End-to-end testing)

The prohibition against component testing may go too far. Lab test results are poor sources of estimates of a system's reliability (see comments on reliability estimators later) and so I do not accept as valid the concern expressed that partial-system tests limit the thoroughness of testing for reliability. Lab tests of the kind envisioned in VVSG do not test for reliability whether they are end-to-end or not.

Requiring that every test always be run in the context of an end-to-end task imposes an enormous testing cost on the voting equipment vendor. It is important to consider cost-benefit issues for individual tests: the more expensive we make each test, the smaller the set of tests we can rationally impose on the vendor. Software defects are more like design defects than manufacturing defects—a software error shows up in every manufactured copy of the same product and discovery of new defects comes from testing the system in a new

way, not from testing for the same flaw again in a new copy of the device. Frequent, perfect execution of a narrow range of tests will yield less information than strategies involving broader sets of tests.

This issue becomes moot if we allow the public to obtain these systems and run their own tests. Given public and researcher interest in these systems, such testing will probably be a far more varied (and thus richer) sample of component and end-to-end tests than can be achieved by any test lab or afforded by any vendor.

Part 1, Section 1.1.4 (Epollbooks and ballot activation)

A majority of respondents, in a poll of the IEEE SCC38 Voting Standards committee, supported the recommendation that:

> *"The use of electronic poll books be banned in polling locations and that printed poll books be required."*

The underlying concern is that in past cases in which fraud has been detected, printed poll books with voters' actual signatures have been important records.

VVSG does not take a comprehensive look at the auditability of election records and therefore it is difficult to assess the impact of substitution of printed poll books with electronic ones. Given that there have been so many problems with electronic voting equipment, the addition of more electronic components is bound to be met with appropriate mistrust.

Part 1, Section 1.1.6 (Core requirements)

In the new "Core Requirements" for voting equipment, members of the IEEE SCC38 Voting Standards committee favored adding the notion of complete observability. Operationalized, this means that every bit of storage in the voting system should be readable and examinable by a tester or other inspector (e.g., auditor or forensic analyst).

Part 1, Section 3.1 (Usability, accessibility & privacy requirements: Overview)

A majority of respondents, in a poll of the IEEE SCC38 Voting Standards committee, supported the recommendation that:

> *"The use of electronic voting machines be limited insofar as practical to use by disabled voters who cannot otherwise vote unassisted. And that limitation remain in place until reasonable standards and security for electronic voting machines that provide for public trust and transparency can be developed and implemented."*

Implicit in this recommendation is a lack of confidence in the currently proposed VVSG.

Until these systems' code and tests become publicly reviewable, doubts about the trustworthiness of these systems will persist, fueled by widely-publicized examples of problems, and equally-widely publicized comments about the secrecy of the ensuing investigations.

Part 1, Section 3.1 (Usability, accessibility & privacy requirements: Overview)

Chapter 3 repeatedly identifies three types of users of the electronic voting system: voters, poll workers and election officials.

Chapter 3 does NOT identify a fourth type of user, the election auditor. This is a critical role for these systems, especially because of the low level of public confidence in electronic voting systems and the high level of

mistrust expressed by the media. If these systems are not easy to audit, and if audits are not obviously thorough and trustworthy, they will never be accepted by the public or the press. Nor should they be.

It is commonplace, in requirements analysis and user interface design, to consider the several roles (types of users, personas) involved in the system and to evaluate the system through the eyes of each role.

VVSG consider usability or utility of the voting system for the auditor only briefly (Part 1, Section 4.2), explaining that "audits are considered part of the election procedures and cannot be mandated by the VVSG." Thoroughgoing assessment of the usability and utility of the voting system for the auditor does not mandate that audits happen; it only mandates that if audits are attempted, the equipment will doing an acceptable job of supporting them. This omission is serious and has broad implications for the VVSG document. I believe that this Chapter, Part 1: Chapter 7, and much of Part 3 would change if such an analysis was undertaken.

Part 1, Section 3.2.1.1 (Overall performance metrics)

The emphasis on repeatable, controlled experiments that are not "realistic" measures of voting behaviors is not a good way to assess the usability of the system and not a good foundation for discovering or reporting usability problems.

The creation of a Voting Performance Protocol that is so standardized that even the test ballot is fully specified runs the risk of design optimization by the vendor in ways that pass the test easily without improving the usability of the system as a whole. A test ballot is only a tiny sample of the population of possible ballots, and it stops being a representative sample when application designers consider it specifically in their designs and test designers know that they will be required to use this ballot in their tests of each version of each voting device they test. Metrics based on such a test are likely to show apparent improvement over time, as systems are optimized toward better performance on this particular test. This improvement may or may not correlate with the underlying level of usability.

Usability tests (content and results) should be public records. Trends in test results should be compared from version to version of a voting system.

Part 1, Section 3.2.3.-A.1 (Visual privacy)

In practice, insertion of ballots into optical scanners has been confusing enough for voters that poll workers often end up handling the ballot. I know this from voting in several elections in Brevard County, Florida and from observing the conduct of one election. In one election, poll workers insisted on inserting ballots themselves (and occasionally commented on the choices of the voter). In others, voters were allowed to insert their ballots themselves but a poll worker was stationed beside the scanner and could easily see what was on the ballot. The rationale for the stationing (I asked) was that people often had trouble with the scanner and so the poll worker was there to help quickly--speed being important when there were (as there were in fact) long lines of voters waiting to cast their vote.

Assessment of the privacy actually afforded by a voting system must include observation of the use of the equipment in actual elections. There should also be a public reporting system for members of the public to identify system-level privacy issues (like this one) and any other usability issues.

Without a well-publicized, easy to access, feedback loop from the actual users of the system to the system developers, we will never understand the actual usability characteristics of a system or how it needs to be improved.

Part 1, Section 5 (General security requirements)

VVSG does not explicitly consider equipment to be used with mail-in ballots. A system that works adequately for in-person voting might be subject to very different use patterns, and might in many ways be less secure, when used to record mail-in ballots.

The particular issue that I have been asked to raise on this (via a poll of the IEEE SCC38 Voting Standards committee) considers automated signature verification, and the particular recommendation I have been asked to make to you is the addition of this restriction:

> *"The use of totally-automated signature verification is banned for validation of mail ballots."*

Part 1, Section 6.3.2 (Accuracy / error rate)

Rather than defining a benchmark in terms of a typical case, VVSG should require the manufacturer to specify a maximum ballot complexity (number of contests is one important aspect of ballot complexity, number of alternatives per contest is another) and demonstrate acceptable results in simple, typical, and maximally complex cases.

If we relax the end-to-end testing requirement, we can assess accuracy empirically by generating large numbers of randomly-completed ballots by computer, feeding these ballots to the voting machine, and using the records of what was generated as the oracle for what was recorded. In an optical scan test, this is easily achieved by printing completed ballots. Similarly, by connecting a test computer to the input port for the keyboard, we can simulate human user input to DREs, allowing massive sets of automated tests, all with oracles.

Part 1, Section 7.5.1 (Issuance of voting credentials and ballot activation)

A majority of respondents, in a poll of the IEEE SCC38 Voting Standards committee, supported the recommendation that:

> *"The use of electronic poll books be banned in polling locations and that printed poll books be required."*

The underlying concern is that in past cases in which fraud has been detected, printed poll books with voters' actual signatures have been important records.

VVSG does not take a comprehensive look at the auditability of election records and therefore it is difficult to assess the impact of substitution of printed poll books with electronic ones. Given that there have been so many problems with electronic voting equipment, the addition of more electronic components is bound to be met with appropriate mistrust.

Part 2, Section 2.1-A (Quality and Configuration Management Manual: Develop and present)

The Quality and Configuration Management Manual shall be a public record.

Part 2, Section 2.1-A.3 (Project plan)

The project plan shall be a public record.

Part 2, Section 2.1-A.4 (Quality check)

The results of each quality check shall be a public record.

Part 2, Section 2.1-A.5 (Problem log)

The problem log shall be a public record.

Part 2, Section 2.1-A.7 (Testing statements for every part, component, and assembly)

The testing statements shall be a public record.

Part 2, Section 2.1-A.8 (Inspection processes)

The descriptions or statements of inspection processes shall be a public record.

Part 2, Section 2.1-A.9 (Testing statements for the entire voting system)

The testing statements for the entire voting system shall be a public record.

Part 2, Section 2.1-A.12 (Records of all critical parts, components, and assemblies)

The records of all critical parts, components, and assemblies shall be a public record.

Part 2, Chapter 3 (Technical Data Package (manufacturer))

The Technical Data Package shall be a public record.

Part 2, Chapter 4 (Voting equipment user documentation (manufacturer))

The auditor should be considered a “user” of the system and user documentation should be supplied to support the performance of the auditor, including detailed recommendations for auditing the accuracy of election results on this particular type of system.

Part 2, Chapter 4 (Voting equipment user documentation (manufacturer))

The user documentation shall be a public record.

Part 2, Chapter 5 (Test plan (test lab))

The test plan shall be a public record.

Part 2, Section 5.1-A (Test plan references)

All test plan references shall be public records unless they are proprietary materials supplied with COTS software embedded in the voting system.

Part 2, Section 5.1-E (Test plan, reproducible testing)

The test plan should document a set of reproducible test cases that provide the traditional level of coverage of product requirements. However, testing should also include exploratory (“open-ended”) testing of the functionality, performance, usability and other quality attributes of the system. The test documentation should provide a final report that describes the strategy of the exploratory testing, outlines the testing actually done and describes the results. However, because this testing will (and should) vary from build to build of the software, it should not be documented to the level of detail of a traditional reproducible test.

Part 2, Chapter 6 (Test Report (test lab))

The test report shall be a public record

Part 2 6.1-J.2 (Test report, error rate)

The report total error rate is not a statistically meaningful measurement. It is easy to manipulate this statistic by including a large pool of easy-to-pass tests. This is an example of the fundamental difference between hardware and software reliability. We expect the hardware device to fail on a random basis when we present identical tests, but we should not expect that of the software. Software failures are more likely to be triggered by combinations of data and sequence that have not been previously tested.

Part 2, Section 7.1-A (Public Information Package contents)

The public information package should be a public record.

It should be redacted to exclude only proprietary information provided by third parties to document COTS software. Because public testability of the voting system is a critical aspect of the quality control of the system, and of the gradual development of public confidence in systems of this kind, the determination of what can be redacted is well within the scope of the VVSG.

Part 3 (Testing Requirements)

VVSG does not consider the case in which a voting system passes certification testing but later use reveals flaws that, if found during testing, would have caused a failure of testing.

VVSG should explicitly address this case, providing a process for decertification of the voting equipment and also a reconsideration of the qualification of the independent test lab that missed the problem.

A majority of respondents, in a poll of the IEEE SCC38 Voting Standards committee, supported the recommendation that:

> *" a section be added to the VVSG that pertains to the decertification of voting systems. This section will provide details regarding the process that would occur if evidence of a flaw has been revealed in a certified voting product or component such that it should not have passed the ITA examination and/or would be in violation of the VVSG requirements. Upon receipt of this evidence by the EAC, the ITA that performed the original certification testing of this product or component shall be immediately contacted and required to perform additional testing with a results report issued to the EAC within 10 days. If the product or component fails the test, the EAC must immediately rescind the certification, and all entities (states, counties, municipalities) that have this product deployed for use must immediately be notified (at the vendor's expense) of the decertification. Certification can later be made to a new version of the product that has corrected the failure if it subsequently passes both component and integration testing by the ITA."*

Part 3, Section 2.1 (Conformity assessment process: Overview)

VVSG specifies the testing that an independent test lab will perform.

The first problem with this process is that the equipment vendor picks the test lab. This creates a strong incentive for the test lab to please the vendor, in order to obtain the vendor's repeat business and the business of this vendor's competitors. The labs therefore have a disincentive against creating tests that are more harsh or more extensive (more time consuming and more expensive) than the bare minimum specified in VVSG. This is not a genuinely independent set of tests and it is a poor way to engender public trust in the test results.

VVSG must address and eliminate the problem of test lab conflict of interest.

Part 3, Section 2.1 (Conformity assessment process: Overview)

Independent test labs play an important role in the qualification of critical products. VVSG should continue to require conformity assessment by independent test labs. However, it should also enable testing by the general public.

Test lab testing is stunningly expensive. Wise manufacturers do the vast majority of their testing in-house, long before submitting software to a regulatory-function independent lab. This is one of the key reasons that the labs normally focus on confirmatory tests, rather than open-ended, harsh investigations. (This is not a universally-accepted characterization of the relationship, but I speak from knowledge of several test lab executives who have been friends, coworkers, legal clients or technical clients and this is what I have heard repeatedly from the sharpest of them, and from several technical staff who are experienced, as vendors of non-voting software products, with taking products through independent lab assessments.)

In the case of voting equipment,

(1) There is reason to believe that voting equipment vendors' internal quality control processes to date have not been sufficiently thorough, and

(2) Potential conflicts of interest could motivate vendor staff to embed code that could be used to change the results of an election. This is a difference in kind from other regulated industries, in which the vendor and its staff have much to lose and nothing to gain by embedding defects into their code.

VVSG attempts to compensate for the (deserved or undeserved) lack of trust in vendor quality control and integrity by expanding the role of the independent test lab to include thorough source code review and open-ended vulnerability testing (i.e. exploratory testing).

However, the time cost to thoroughly review this much code is enormous and is almost certainly beyond the expectations of the authors of VVSG, the vendors or the labs. Similarly, the time cost to build skills needed to do a thorough job of exploratory testing is well beyond the 12 person-week scope of VVSG. Independent test labs offer strong skills in creating and executing thoroughly documented tests that trace well to unambiguous documents, but these are very different skill sets from those needed for exploratory testing. It is not clear that labs whose core competencies support traditional regulated-industry testing would even know how to assess the competence of the exploratory testing services they would subcontract.

Rather than stretching the role, capability, and cost of lab testing beyond sensible limits, VVSG should require voting equipment vendors to enable public testing with the following requirements:

> *(a) non-COTS components of voting equipment software should be publicly reviewable;*
>
> *(b) all test-related artifacts (including test plans and test results) for voting equipment software be public records;*

(c) all voting equipment specifications other than proprietary specifications for COTS components should be public records;

(d) that voting equipment tendered for sale to any government in the United States should be made available to the public at a similar cost, so that researchers can obtain and test the voting system;

(e) the license agreement for the voting equipment software must not prohibit publication of any test results, benchmarks, or other results or opinions stemming from evaluation of the equipment; and

(f) the license agreement for the voting equipment software must not prohibit reverse engineering any of the equipment's software (including COTS embedded software) to the extent necessary to discover implementation or security flaws.

Public testing offers several benefits, mitigating several inherent weaknesses in traditional conformance testing.

The underlying problem is that software defects are not like manufacturing defects. There is relatively little value in running the same test over and over. There is great value in testing the software in different ways. Two tests are distinct if the program can pass one and fail the other. For any nontrivial program, the number of distinct tests is virtually infinite. The test design challenge boils down to a selection problem: which tiny subset of the pool of distinct tests should we select?

To a degree, the testing problem can be reduced by thorough code review. However, many potential errors cannot be caught in code review.

Let me start with an example from one of the most successful computer peripherals manufacturers in the United States. Suppose that we have a pool of N automated regression tests, and in a given build, the software passes M of them. Restrict further work in this build to these M tests that the software passes individually. Do not change the software in any way. Now run random sequences of tests sampled from the M until either the software fails or the software has not failed for a criterion period of time. Under all code coverage models and under most or all failure models used to estimate software reliability, the expectation would be that the system should not fail random sequence testing because it has passed each and every test in the sequence. However, this technique exposes timing problems, conflicts involving multiple processors, memory leaks, memory corruption that builds over time, stack corruption that builds over time, and several other serious problems. Early in development, this manufacturer's code does well to survive a test like this for more than a minute of execution. Release to the public, in this company, requires survival (no failures in a long sequence) times of at least 72 hours. Variants of long-sequence automated testing have been used to qualify telephone systems and other embedded software for at least 25 years.

Creating well-documented scripts for this kind of testing would be appallingly expensive. I have never seen a test plan for regulatory independent testing that includes testing of this type.

The long-sequence example illustrates the fact that tests can be distinct in subtle ways. Here are more examples:

- Features tested separately can show no problem, but tested together can yield a memory leak or a corrupted stack.
- Features tested with most data values can pass, but can fail on special cases that are hard to predict a priori. In a famous example from the testing of the MASPAR computer's integer square root function, of the 2^32 possible tests (32-bit integer inputs to the function), all tests were run and only two cases failed. Neither case had an input value that was near any obvious boundary. The underlying cause was a rounding error that had an impact on the final result only twice (the error was rounded out before having an impact in several other cases). Boundary testing provides a heuristic, but one that is far from infallible. A more famous but less easily summarized problem of this class was the Pentium FDIV bug.
- As another illustration, on the software-testing mailing list (2/18/2008), Ross Collard (a well known practitioner / teacher) said, "I have been sifting through archived data on bugs found by extensive date testing (Y2K testing), and correlating bugs with the conditions tested. No matter how I choose

to define the boundaries, so far I have not found statistically valid evidence for that assertion that "errors lurk at boundaries."

- Features that pass on one configuration can fail under a subtly different one. As a classic example of this, Intuit released a version of Quicken that, to their surprise, crashed during a database search if and only if the computer involved was running Microsoft's new IntelliSense keyboard driver. Other early Windows programs failed on configurations that included both resolution (1600x1200) video and high-resolution (600 dpi) printing. As one example (this problem affected several software developers), in a product that I worked on, no failures occurred with high resolution video but lower resolution printing or high resolution printing but lower resolution video, but in combination, some tasks (such as a print preview) corrupted system memory and caused a system crash.

None of these problems stand out in source code reviews. Long-sequence bugs that I personally worked with in telephone systems showed up in code that had been thoroughly reviewed and had been subject to glass-box testing that involved (for all of the code that was eventually implicated in the failure) 100% statement coverage, 100% branch coverage, and testing of every obvious boundary of every obvious (to a person reading the source code OR operating the program black box) variable.

Testing is a complex problem. Our current state of the art does not allow us to fully solve it and so we are well served by creating assessment processes that view the code and test the code from widely differing perspectives, in richly different ways.

Independent test lab testing is not well suited to this constant variation. Decisions about what to vary in what ways are often intuitive and hard to justify. Auditing the skill and thoroughness of this style of testing is difficult for a test manager working with a skilled staff who are willing to freely share their private thoughts about their strategies and choices. Auditing under more adversarial circumstances would be much more difficult--obviously bad work could be exposed, but the range from not-awful through excellent would be very hard to assess in the face of testing staff who were responding cautiously (defensively) as is not uncommon in adversarial audits. The labs could spend millions of dollars on testing and vendors and regulators could spend hundreds or thousands of hours arguing about whether a pool of tests (and failure rates) was sufficient, representative, fair, etc.--and at the end, we would still have significant uncertainty at high cost.

Rather than push labs beyond their zone of excellence, the public testing approach relies on the public to generate a wide variety of approaches that supplement the conformance testing done by the labs.

One objection that has been raised to this proposal is that it has been very difficult, in open source projects, to attract sustained testing at this level of skill. Voting systems, though, are special:

- National Science Foundation merit criteria require every proposal to include discussion of the impact of the research for the public benefit. Some researchers will choose to improve the perceived merit associated with their NSF proposals by using voting equipment software as target platforms for the test technique, code review technique, reliability model, or whatever new technology they want to study. As the principal investigator for three NSF projects totaling over a million dollars in funding, and having served on several NSF panels, I would certainly do this whenever possible in my grant proposals and I would expect to see a lot of it in new proposals. These proposals would not be primarily targeting voting systems; they would be using the voting systems as test beds for the ideas that they wanted to explore. However, as they found problems in the voting systems, they would report them.
- Many doctoral students would find it convenient to use voting equipment software as a test bed for their work because the software is thoroughly documented and fully available. Additionally, work on this type of product can neither be dismissed as work on "toy" applications nor as work on low-grade software chosen to misleadingly highlight the strengths of one particular technical approach or unfairly denigrate another. The intimidating oral exam question, "But why did you choose THIS for

your test bed?" would be easy to handle--this would be very motivating for several doctoral students, at least many of the ones I know.

- It is likely that other nongovernmental organizations would raise money to fund testing efforts for this software. Given the enormous public mistrust, there is an enormous opportunity for fundraising for activities that could be perceived as mitigating the risks that lead to that mistrust.
- With the publication of test plans and results, some test labs will be motivated to prove their capability by demonstrating that their approach to testing exposes bugs missed by the NIST-certified, prestigious labs that tested a given voting system. Such demonstrations would make for useful advertising copy. Live demonstrations of flaws in other well-tested software (e.g. Microsoft Office) have been the core of some keynote addresses at software testing conferences--very powerful marketing for the test group (and test techniques) involved.

Another objection raised to this proposal is that the results go nowhere. That is, if a research group does find a defect in a voting system, there is nothing in VVSG that closes the loop, requiring immediate repair by the voting equipment vendor. This might be true today, and it might stay true in all subsequent versions of VVSG, but if members of the public find and publish defects in a voting system, this can affect buying decisions by subsequent potential purchasers and it can also affect the reputation of the test lab that signed off on the system. Over time, systematic weaknesses in the assessment of voting systems will be understood and mitigated.

Part 3, Section 1.1.4.1 (End-to-end testing)

While much testing should certainly be done end-to-end, it is not necessary to run every test end-to-end. Independent testing imposes an enormous cost on the voting equipment. It is important to consider the cost-benefit considerations for individual tests. The more expensive we make each test, the smaller the set of tests we can rationally impose on the vendor.

Allowing the public to obtain these systems and run their own tests will create an important additional source of end-to-end tests. Given public and researcher interest in these systems, such testing will probably be far more varied (and thus a richer sample) than can be achieved by the test lab anyway.

Part 3, Section 1.1.4.3 (Open-ended vulnerability testing)

The more common name for open-ended testing is exploratory testing. This is commonly done in the testing of any attribute of computer software. There is widespread belief among practitioners, including plenty of experience reports at practitioner conferences, that exploratory testing yields many more failures than running the same sequence of regression tests time and again.

The need for skilled exploratory testing in the lab will be significantly reduced if public testing is made possible.

The inevitable arguments about whether a test lab's exploratory testing was sufficiently skilled and was run for a sufficient time will also be reduced by public testing, which will be a significant separate source of open-ended tests.

Part 3, Section 2.5.5-C (Critical software defects are unacceptable)

I agree with that the VVSG comment that "the system cannot be considered trustworthy even after the known fault is corrected, because the cases that the test lab does not have the opportunity to test can be expected to conceal similar faults."

However, mitigation of this risk does not merely require "a new application and starting [the testing] over" (the VVSG requirement). Genuine mitigation of this risk requires new tests that look for those "similar faults" in other parts of the code that have not been tested this way for this particular type of error.

Part 3, Section 5.2.3-B.1 (Practical limit on capacity operational tests)

This is a perfect example of a situation in which a test fixture can drive the system to a limit that is impractical to reach with end-to-end testing. We should permit the use of risk-focused tests that are not end-to-end as a more desirable alternative to testing with reduced limits.

Part 3, Section 5.3.1 (Benchmarks: General method)

It is inappropriate to treat software regression tests as if they were a representative sample of the behavior of the system under test because the system is optimized to pass them as they are repeatedly run. The more times they are run, the less predictive power they have with respect to other tests that involve other data, other combinations of functions, or other sequences of events.

Part 3, Section 5.3.1 (Benchmarks: General method)

As it applies to software, this section's terminology is ambiguous or undefined. What is a test event? What is a test volume? What is a test campaign?

Part 3, Section 5.3.1 (Benchmarks: General method)

VVSG should not estimate reliability (or acceptability in any other way) of software by calculating the number of failures (test events?) divided by the number of tests (test volume?). It is too easy to influence this estimator by including large numbers of easy-to-pass tests.

Part 3, Section 5.3.1 (Benchmarks: General method)

Lab tests focused on conformance testing do not model usage patterns in the field and therefore test results based on them cannot estimate failure rates in the field. This is not a defensible method for estimating reliability.

It may be possible to develop operational profiles from which reliability tests could be developed but this will require extensive research that would not be part of the approval process of any particular voting system.

Part 3, Section 5.3.1 (Benchmarks: General method)

VVSG should not model software failure rates with a Poisson distribution or a Poisson process, or with any other distribution or stochastic process unless that distribution or process is derived from a logically and empirically-defensible model.

There is no reason to think that a mixture distribution that combines hardware and software events would be a simple Poisson. It is important to recognize that we are estimating performance in the tails of the distribution. To the extent that the true underlying distribution differs from the Poisson, deviations are particularly likely in the tails, yielding overestimates or underestimates of the significance of a given number of failures in a given period of activity. The statistical tables may have validity for hardware-related failures or for misfeeds, but there is no reason to think they would be valid for software or for the mixture distribution.

Robert Austin [Measuring and Managing Performance in Organizations, Dorset, 1996] wrote a particularly compelling discussion of the risks associating with basing high-stakes decisions on metrics that are not tightly

tied to the underlying attribute that the metric attempts to estimate. Equipment vendors have a strong interest in making their numbers look good, or at least good enough to pass testing, and therefore they have an incentive to optimize their behavior in ways that improve the numbers. They also have an incentive to challenge the ways in which the numbers (total failures, total volume) are calculated, to arrive at a result that is more favorable.

I assume in all that follows that the manufacturers are acting in good faith. When we tell someone that their performance will be passed or failed according to a criterion, there is nothing dishonest in optimizing efforts to meet that criterion. If anything, that is what the criterion is there to accomplish. In particular, you should read the comments that follow with the understanding that I am explicitly and intentionally assuming that the vendors will be factually honest in everything that they do and that they are primarily motivated to achieve a "pass" from the system and not particularly motivated to do so in such a way as to mislead anyone about the underlying quality of the product.

- The VVSG (5.3.1) correctly notes that one of the characteristics of the Poisson model is that the probability of an event occurring stays constant over each "unit of volume processed." It then notes that this is not exactly correct for software because software errors might be nonrandom, that is, they might be triggered every time the same set of conditions is tested. It then dismisses this problem by saying "Thus, when all specific evaluations are used together, the different test methods complement each other and the limitation of the particular test method with respect to nonrandom events is not bothersome." I think this is a novel conclusion. I do not understand how mixing nonrandom events with random ones (to the extent that there are random failures in software) is a good foundation for a model that assumes that all events are random.
- For most software failures, the failure itself is not random at all. Given THESE conditions, THAT failure will occur. What might be thought of as random is whether and when the particular test that includes those conditions is presented to the software. The probability that a given test will yield a failure thus depends on at least two factors: how many problems remain in the software and how powerful the test is with respect to the types of problems that remain. As the software goes through testing, problems are fixed, and so the number of remaining problems diminishes. Therefore the assumption that the rate parameter of the Poisson distribution is stationary is implausible.
- The power of software tests run is not related to the underlying reliability of the software. Test power is (analogous to the power of a statistical test) the ability of the test to detect an error of a certain type if it is there. Note that no test has "absolute" power—a test that is optimized to expose an off-by-one error might be a weak detector of rounding errors. Thus, a lab can achieve a low failure rate (high apparent reliability) by running relatively low-power tests and a high failure rate (low apparent reliability) by running relatively high-power tests. Regression tests lose their power as they are used repeatedly, because the errors they are optimized to detect get found and fixed. (This problem was labeled the "pesticide paradox" by Boris Beizer in Software Testing Techniques, Van Nostrand, 2nd Edition, 1990; see also Kaner, Testing Computer Software, McGraw-Hill, 1987, p. 94). The improvement in apparent reliability with repeated use of regression tests should not be expected to predict improvement of reliability in the field, because users in the field do other things with the software beyond running these particular regression tests. Varying testing, for example by changing parameter values, combining tests, or running the tests in long random sequences, probably does a better job of mitigating operational risk but under the VVSG benchmarking definition, this testing will drive down estimated reliability at the same time as it contributes to the actual improvement of reliability.
- If there are B bugs in the software and we find a bug with 100% certainty if we run a specific test (or ones sufficiently like it), the probability of detecting one of the errors boils down to the probability that an error-revealing test makes it into the test suite. That depends on the sampling strategy (any test design strategy can be seen as a sampling strategy), whose details are under the control of the test lab, with some influence by the vendor and the VVSG. It is not clear what this sampling strategy has to do with the underlying reliability of the software. The VVSG-specified rate parameter probably has more to do with this sampling strategy than with operational reality.

- A Poisson process model for failures makes several assumptions. The first that I noted is that the probability of discovery of a failure is constant over time. This is implausible because the program presumably gets more stable, and the tests (if they are the fully scripted regression tests required by VVSG) get less powerful over time. The second is that instants of time (or units of volume) (that is, tests) are independent. This is also implausible. A widely reported pattern in test data is that some modules are much more error prone than others. Presumably, this is due to the inherent difficulty of some problems, the tremendous variations in individual programmer competence, perhaps a difference in time pressure associated with completing some tasks compared to others, etc. A sensible testing strategy adds new tests to further investigate areas that have shown some failures. If this is done, the probability of these tests exposing problems is relatively high, but that is a conditional probability—test X2 has a high probability of exposing a problem because test X1 did expose a problem. This is precisely the opposite of the assumption of independence between X1 and X2. Of course, one can preserve apparent independence of tests by never adding new tests to more carefully study areas of the program that seem weak. However, if the objective is to check the quality of the software, this restriction (no follow-up with related tests) would be bad testing, in conflict with the objective. Another problem for the idea of independence is the problem of identicality. Do we really think that the same test, run a second or third or fourth time (regression testing) should be treated as an independent sample from the pool of possible tests?
- Another problem with the Poisson process model is that some bugs are inherently harder to detect than others. Thus, if we have B bugs, the probability of detecting each one is not the same. It is usually easier to find a bug if it depends only on one feature or one parameter of a feature. A relatively simple test will do the trick. The only risk of obscurity is the possibility that only one value of the parameter would lead to failure. Special cases do exist, but they are often (and not always) at boundaries that are either visible externally or on review of the code. VVSG requires testing at boundaries and therefore most (and not all) of the single-variable special-case bugs are probably covered. However, some bugs involve combinations of two, three, or more variables or functions. Some of those variables might be relative timing of events (race conditions) or amount of free memory when a given task is attempted or access to some other resource. These are harder to detect with simple tests.
- Even the assumption that one test can only expose one defect is empirically challengeable. Unless a test is so successfully focused on the processing of one variable by one method that multiple problems are impossible (unit testing can achieve this, but not system testing), a given test might trip first over one feature and next over another. A test that combines 10 features might yield 10 (or more!) failures. This is not a merely theoretical possibility. It is a common heuristic in system testing that testing should start with single-feature tests and progress to relatively simple multi-feature combinations and then progress to user-meaningful rich scenarios. This is based on experience: companies that do mainly the multi-function scenario testing often find their tests blocked—a failure in the first steps of the test blocks continuation to the later steps. After the first bug is fixed, the test fails again, in a way that blocks further execution, and then when used again, it fails again. In the practitioner community, there are many anecdotes of bugs that should have been easy to find being found very late in testing because the planned test that finally exposed the bug was blocked by other bugs. Thus, we have commonplace examples of tests exposing many more than one defect.

In sum, there is no reason to think that any of the assumptions underlying a Poisson process model apply.

The VVSG provides a table for determining critical values associated with the ratio of the number of test events to the test volume. The idea is that even if the Poisson distribution is not a perfect estimator, perhaps it is a good first approximation. I am not a professional statistician, but I do have about 8-12 semesters of probability/statistics courses, a few more on modeling, and some practical research experience. My understanding is that if two distributions are similarly shaped, using one as an approximation of the other is possible—but the relative differences are likely to grow as you go out to the tails. That is, similar distributions often differ most in their assignment of probabilities to lower-probability events. A 90% criterion value is pretty

far out in the tail of the distribution. If none of the assumptions of the Poisson model apply to software testing, it is hard to believe that numbers taken from the tail of that distribution accurately predict much about the system under test.

Here are some other problems associated with the VVSG's estimator of software reliability:

- As far as I can tell, in its treatment of reliability estimation, the VVSG assumes that the test volume is a fixed value, not itself a random variable. This is only true if one set of tests is run once and no other tests or other events are considered. Given that there will be regression tests, this is not true. Even if we count each regression tests only once, no matter how many times it is run (but that is unfair if the same test later exposes a different bug), a competent test lab does additional testing around any bug reported fixed. That is, if a given set of test conditions exposes a failure, and the equipment vendor fixes the bug and returns a new version of the software, a competent tester will not only test the fix with the original test that exposed the bug but will create new tests to see whether the fix actually covered the underlying problem. These can expose new problems and so they must be new units of test volume. Of course, to preserve a fixed volume of testing, we could choose not to allow such testing. However, as in many cases considered above, it might be highly undesirable to allow an incorrect model to be used as an excuse to constrain the power of testing.
- The VVSG assumes that the test results obtained in conformance testing can be used as a sound statistical estimator of the population reliability (see 5.3.1). This assumption is unreasonable. The reliability of the voting equipment software, in the field, will depend on how the software is used in the field. The tests designed for conformance testing are not designed with an objective of mapping to field usage. They are designed to achieve a level of simple coverage of the code, another level of simple coverage of documented requirements, another level of coverage of boundary values of individual variables, and so on. There are sound statistical methods available for estimating the reliability of the software in the field (see, for example, Musa, Software Reliability Engineering, McGraw-Hill, 1998), but they start from development of operational profiles—profiles of ways in which people will actually use the software. The next task is estimation of relative frequency of occurrence of each profile—a usage pattern twice as likely in the field should be involved in twice as many of the reliability tests. From here, one generates a large pool of tests, deriving each test from one of the profiles (varying specific parameters, or sequences of operation in ways consistent with the profile). Ideally, that generation should be itself driven by a random process that reflects usage patterns. From there, failure rate over the sample of tests might well be a valid estimator of field failure rate. If it is important to have a failure rate estimator, it is important to have a number that bears a defensible relationship to the underlying parameter.
- Development of operational profiles is an expensive proposition. Some vendors (such as AT&T and Microsoft) have access to customer usage patterns and, at significant expense, can develop profiles on their own. It is not clear that voting equipment vendors have this level of access to customer usage of their own equipment. In addition, the better study might be of usage of voting equipment generally, across vendors. If the profiles are essentially the same across vendors and equipment models, the same profiles can be used with new models as they are introduced, rather than requiring a hugely burdensome (in time and money) research program for each new model. Rather than requiring voting equipment vendors to do this type of research, it might make more sense for NIST (or some other agency) to fund independent (e.g. university-based) research to develop such profiles and assess their commonality across devices. This will take some number of years. Until those profiles are developed and usable, I think it is inadvisable to predicate any decisions on estimators of software reliability.
- If we assume that the TEST CAMPAIGN includes all tests done by the independent test lab, then the campaign includes all regression tests, no matter how many times these are repeated. Suppose that a given test is repeated ten times. When we compute the TEST VOLUME of the campaign, is this 10 tests or 1?

- It is one thing to say that the lab cannot qualify a device based on the testing of a prototype. It is another thing to bar the vendor from submitting a prototype to the lab for evaluation. Evaluating prototypes gives the lab an opportunity to build expertise with the system under test, making its ultimate testing of the final version more effective. And it gives the vendor an opportunity to discover the weaknesses it is blind to, enabling it to fix problems earlier in the development cycle. It is widely believed in the software engineering community that earlier testing improves quality and reduces costs. While VVSG should not require vendors to submit early versions to the lab (there may be more cost effective ways to evaluate early versions), surely it should not ban it. If a vendor does submit an early version for testing, do those tests count as part of the test campaign? Do those failures count in the ultimate total of test events?
- Suppose two equipment manufacturers have equivalent internal processes, in terms of the quality and functionality of their software, and (for simplicity) equivalent products. One submits its software to the independent test lab a little earlier in development than the other. The first submitter goes to the lab with a few more bugs and goes through one or more rounds of regression testing. Ultimately, the same bugs are found and fixed in both systems. Thus, at the end of testing, we have two equivalently reliable systems. What is the effect on the numbers? If the test campaign counts each time a regression test is run as a separate test, then the first submitter is increasing the measured test volume enormously by submitting early. If the product has only a few more bugs than the product submitted by the late submitter, then even though the first submitter's test event total will be higher, its ratio of test events to test volume will be lower. In contrast, if regression tests are not counted twice but fixed bugs are counted as test events, then the incentive will go to the vendor who waits until the last possible minute to submit product to the lab. If we want VVSG to drive this strategy as a matter of policy, VVSG should explicitly consider and state the policy and the policy choice should be publicly reviewed. Instead, the method of calculation creates an implicit policy.
- To the extent that test volume is left loosely defined, the estimated reliability will vary enormously depending on how the test lab (paid by the vendor) computes the test volume. A rational vendor would spend effort advocating for the largest possible interpretation of volume, so as to make the denominator as large as possible.
- Consider applying a high-volume test strategy to the testing of the device. High-volume strategies have been used effectively for automotive software, telephone switching software, firmware in office automation products, and undoubtedly many other contexts. I will emphasize my own work below, and other work I am personally familiar with, not because I think it is the best in the field but because I can write with authority about the underlying observations. High-volume testing is a well-funded, fashionable area of work. Examples are state-model based testing that execute long sequences of sub-tests, each involving a controlled state transitions; testing using genetic algorithms; search-based testing, in which the test sequence involves test values chosen to be different from each other in a specified way (e.g. maximally dissimilar from the previous tests in the sequence); random-input tests or random-event tests in which a random source generates data or traffic for a long period or until the system crashes; and various types of extreme value attacks (heavy load, big input, extreme combinations, corrupted files) that string many individual tests into one long, grueling sequences of harsh tests. These are often done as security tests today, but they were seen as tests of robustness twenty years ago. These are not fundamentally new ideas. The concerns I raise below in the context of the testing types that I mention applies just as well to all of these other methods. Given that preamble, consider a specific example that I know well: suppose the lab applies long-sequence randomized regression testing (LSRRT), which McGee & Kaner ("Experiments with High Volume Test Automation," Workshop on Empirical Research in Software Testing, ACM SIGSOFT Software Engineering Notes, 29(5) 1-3 2004 discussed under the label "extended random regression"). In LSRRT, you take a set of tests that a particular build of the software has passed individually and string them together in an arbitrarily long random sequence. The key advantage of LSRRT over many other tests that push a device through very long sequences of tests is that the expected results

of each test are known and therefore failures can be detected in terms of unexpected responses rather than waiting until the software crashes. A unexpected responses might be unexpected data, but it also might be unexpected behavioral timing. Oracle Corporation used a method like this in its early qualification of its database, for example. If a test that took T1 time to complete at one point in testing took T2 (much longer or much shorter) time a bit later, system engineers investigated the cause of the difference, often finding coding errors. (Unpublished oral personal communication from Bob Miner, 1987) As McGee & Kaner reported in their short case study summary, LSRRT exposed a large number of serious problems that were not being exposed by the individual tests themselves. Similarly, I have seen serious failures exposed in a different type of long-sequence testing by a PBX manufacturer whose code had gone through thorough unit testing. Stack corruption that built up over time, memory corruption triggered by particular subsequences of events or particular combinations of data, race conditions involving unexpected busy-ness of one of the processors in a multiprocessor system--these are examples of the kinds of problems exposed by long-sequence testing that are much harder to find by testing with one distinct functional test at a time. During an election, a voting system has to run without failure for many hours. Long-sequence testing addresses the question of operation over that long period. One-functional-test-at-a-time testing does not. Should a test lab employ this style of testing? If so, how should we count the test volume? At “Mentsville” (a fictitious name, requested by the well-known equipment manufacturer whose processes McGee and Kaner studied), LSRRT was often restricted to 25 distinct tests that were repeated in a random order. Fewer than 25 wasn’t seen as diverse enough. Many more than 25 made troubleshooting a failure a nightmare. (Why? Remember that the system can pass each test on its own, so the secret of the failure lies somewhere in the sequencing. If failure occurs after 48 hours of apparently-trouble-free operation, analysis of that sequence can be very complex. Limiting the number of distinct tests in the sequence was one way to limit that complexity.) Suppose that the test lab runs a 50-test LSRRT for 12 hours, i.e. a sequence that repeats 50 regression tests in random order until testing is terminated by failure or by successful completion of a 12-hour run. Suppose that on average, each of the 50 tests runs 100 times. Is this test volume 1, 50, or 5000? If the test volume is 1, equipment vendors will have a strong incentive to argue that very little of this testing should be done, because this is a very harsh style of testing. If test volume is 5000, equipment vendors will have a strong incentive to encourage the lab to do lots of this type of testing. I submit that the decision to apply this style of testing, the amount of testing to be done, and the characteristics of the tests combined in each suite should be based on other factors than the calculation of test-events/test-volume, but that calculation will drive potentially harsh debates. In practice, I have been told by testers of regulated products that they don’t do long sequence testing specifically because the metrics are impossible to agree on. Benchmark-estimation rules should NEVER drive decisions about what style of testing would be most effective for illuminating the risks associated with a product.

Part 3, Section 5.4.2-A (OEVT resources and level of effort)

This is an enormous amount of material to review. For this to be meaningful, the OEVT team needs sufficient time with the material to understand it and understand how to apply that knowledge to the actual testing.

Individual differences in background knowledge and skill are so enormous that we cannot tell, by specification, whether a given level of access and time will be sufficient. Rather, the OEVT team members should each certify that they have had sufficient time and experience with the materials to do a competent OEVT.

Part 3, Section 5.4.2-G (OEVT level of effort -- commitment of resources)

With a team of at least 4 members, 12 staff weeks doesn’t allow any individual much time for learning the details of a particular system. A careful review of the supplied documentation could take 3 weeks per person, or substantially more than that, without a single test being run.

Additionally, every time the OEVT team finds a bug, it has to take time to troubleshoot the bug and to craft an explanation of the plausibility of the test: "Bug X is serious because it could come into play in a genuine election under the following circumstances." For a complex scenario, this task alone can sometimes take (I have seen it take, with database applications) 2-3 days per bug. Commitment of resources should thus expand with the number of bugs found, including bugs that turn out to be low-operational-risk, because it takes troubleshooting time to determine that such a bug is NOT likely to cause problems in the field.

Part 3, Section 5.4.3-A (Rules of engagement -- context of testing)

This section asserts that OEVT must be conducted in the context of a model of plausible threats associated with the voting system. Who creates such a model? In what document is it published? Where and when is it reviewed? And is it not part of the OEVT team's responsibility to discover new threats that were not imagined in the development of the conventional tests created by the test lab?

Part 3, Section 5.4.3-B (Rules of engagement -- adequate system model)

The task of the OEVT should not be to verify anything; it should operate from the assumption that what has been handed to it is defective. The OEVT is tasked with discovering new types of problems which were missed in conventional testing. Focusing the OEVT effort on the materials already relied on by the other testers is a good way to encourage OEVT failure. If you aren't going to unchain the OEVT team, don't waste the manufacturer's money on them.

Part 3, Section 5.4.5 (OEVT reporting requirements)

The more reporting, the less testing.