SUPREME COURT OF NEW JERSEY
September Term 2005
Docket No. 58,879

STATE OF NEW JERSEY,

Plaintiff-Appellant,

v.

JANE H. CHUN, DARIA L. DE CICCO, JAMES R. HAUSLER, ANGEL MIRALDA, JEFFREY R. WOOD, ANTHONY ANZANO, RAJ DESAI, PETER LIEBERWIRTH, JEFFREY LING, HUSSAIN NAWAZ, FREDERICK OGBUTOR, PETER PIASECKI, LARA SLATER, CHRISTOPHER SALKOWITZ, ELINA TIRADO, DAVID WALKER, DAVID WHITMAN and JAIRO J. YATACO,

Defendants-Respondents,

and

MEHMET DEMIRELLI and JEFFREY LOCASTRO,

Defendants,

and

DRAEGER SAFETY DIAGNOSTICS, INC.,

Intervenor-Respondent.

SUPPLEMENTAL FINDINGS AND CONCLUSIONS OF REMAND COURT

On remand from the Supreme Court of New Jersey: December 14, 2005

Findings and Conclusions Submitted to Supreme Court: February 13, 2007

On limited remand from the Supreme Court of New Jersey: April 30, 2007

Supplemental Findings and Conclusions Submitted to Supreme Court: November 8, 2007

Christine A. Hoffman, Deputy Attorney General, and John A. Dell'Aquilo, Jr., Deputy Attorney General, appeared on behalf of the State of New Jersey (Anne Milgram, Attorney General, attorney).

Samuel L. Sachs of the firm Sachs & Sachs appeared on behalf of Jeffrey R. Wood and James R. Hausler.

Matthew W. Reisig appeared on behalf of Christopher Salkowitz, Peter Lieberwirth, Raj Desai and Peter Piasecki.

John Menzel of the firm Moore & Menzel appeared on behalf of Anthony Anzano, David Whitman, David Walker, Hussain Nawaz and Jeffrey Ling.

Evan M. Levow of the firm Levow & Costello appeared on behalf of Jane H. Chun, Lara Slater, Elina Tirado, and Frederick Ogbutor.

Jonathan A. Kessous of the firm Garces & Grabler appeared on behalf of Jairo Yataco and Angel Miralda.

Arnold N. Fishman of the firm Fishman, Littlefield & Fishman appeared on behalf of amicus curiae New Jersey State Bar Association.

Jeffrey E. Gold of the firm Gold & Laine appeared on behalf of amicus curiae New Jersey State Bar Association.

Jeffrey Schreiber of the firm Meister Seelig & Fein appeared on behalf of intervenor Draeger Safety Diagnostics, Inc.

KING, P.J.A.D., SPECIAL MASTER

TABLE OF CONTENTS

SUPPLEMENTAL FINDINGS AND CONCLUSIONS OF REMAND COURT

I. PROCEDURAL HISTORY

We filed our initial findings and conclusions on February 13, 2007. The Supreme Court heard argument on April 5, 2007. Consequent upon argument, the Court issued an order on April 30, 2007 (Order) temporarily remanding the matter to the Special Master for the limited purpose of providing defendants the opportunity to conduct at their expense an analysis of the software used in the Alcotest 7110 MKIII-C, NJ 3.11 (Alcotest). The remand was limited to determining whether firmware version NJ 3.11 reliably analyzed, recorded and reported alcohol breath test results. The Court's order further provided the outline of

the protocol for independent source code testing, now that Draeger Safety Diagnostics, Inc. (Draeger) was a party and finally had agreed to cooperate in a scientific inquiry concerning the reliability of its product.

When defendants and Draeger could not agree on an independent software house for source code testing, the Court issued a supplemental order on May 22, 2007 (Supplemental Order) requiring the affected parties to designate their own experts. Elaborate discussions ensued which resulted in examination of the Alcotest source code under non-disclosure agreements by two allegedly independent software houses: (1) SysTest Labs, Inc. (SysTest) designated by Draeger and (2) Base One Technologies (Base One) designated by defendants. The Special Master received the reports of the two experts in due course. They disagreed. The Special Master then scheduled and conducted a testimonial hearing on the experts' reports, pursuant to the Court's supplemental order.

We now advise the Supreme Court that the remand hearing conducted at the Camden County Courthouse for ten days between September 17, 2007 and October 11, 2007, with summations on October 23 and 24, 2007, did not change the Special Master's opinion expressed in the initial findings and conclusions contained in his February 13, 2007 report. We conclude that the

Alcotest, the subject of scrutiny in this proceeding, is scientifically reliable as an evidentiary breath testing instrument, as to both the hardware and software elements, subject to the conditions set forth in the Special Master's initial report and this supplemental report.

Our review of the testimony of the witnesses now follows, along with our more elaborate conclusions.

II. SUPPLEMENTAL ON BURDEN OF PROOF

All agree the burden of proof in this proceeding is on the proponents of the evidence, the State and Draeger, by clear and convincing evidence. The parties also agree that there is no extant New Jersey case which quantifies this burden. All agree that it rests somewhere between the customary civil burden of reasonable probability (51%) and beyond a reasonable doubt (perhaps 98+% or 99+%). One well-recognized authority, Judge Jack Weinstein of the Eastern District of New York, has expressed this view in a criminal law context: "Quantified, the probabilities might be in the order of above 70% under a clear and convincing evidence burden." United States v. Fatico, 458 F. Supp. 388, 404 (E.D.N.Y. 1978), aff'd, 603 F.2d 1053 (2d Cir. 1979), cert. denied, 444 U.S. 1073, 100 S. Ct. 1018, 62 L. Ed. 2d 755 (1980). See United States v. Copeland, 369 F. Supp. 2d 275, 286 (E.D.N.Y 2005), aff'd, 232 Fed. App'x. 72 (2d Cir.

2007) (Judge Weinstein cites Fatico with approval of a 70% burden in "clear and convincing standard" cases).

The Oregon Supreme Court has referred favorably to research disclosing a purported national consensus on the "clear and convincing" burden of proof as a 75% likelihood. Willbanks v. Goodwin, 709 P.2d 213, 218 n.9 (Or. 1985). Personally, we might be inclined to put the burden as high as 85% to 90%. However, we conclude that the State and Draeger have met the clear and convincing burden in this proceeding.

III. EXPERT TESTIMONY

1. Overview

The parties called four expert witnesses: Bruce Geller on behalf of Intervener Draeger; Norman Dee on behalf of the State; and John Wisniewski and Thomas Workman on behalf of the defense. Geller and Wisniewski examined the Alcotest's source code for obvious issues and consistency with the algorithms pursuant to the Court's supplemental order dated May 22, 2007. Dee and Workman drew conclusions on the code's scientific reliability based on their analyses of the static code reviews performed by the other experts. Neither Dee nor Workman saw the actual code, although Dee scanned portions of it shortly before this remand hearing. All four witnesses submitted reports.

This court found each of them qualified in their areas of expertise.

This court called Brian Shaffer, an employee of Draeger who wrote the customized portions of the source code for New Jersey, as a witness. Shaffer testified as both a fact and an expert witness but did not prepare a written report.

Because of the need to analyze many pages of testimony to fully understand each expert's opinion on any particular issue, we provide detailed summaries of their findings and conclusions with comments, where appropriate, on the weight this court placed on their testimony. The order of these summaries corresponds to the appearance of the witnesses on the stand, except for Shaffer who also appeared as a State rebuttal witness. To assist the Court in its understanding of this highly technical evidence, we present the following overview of the testimony.

Geller was a software quality engineer who worked for SysTest of Denver, Colorado, a nationally recognized company which specializes in software testing. Geller and another employee reviewed the Alcotest's source code under the supervision of a senior project director. Although their report was a collaborative effort, Geller fully agreed with its findings.

Geller testified that the source code was written by more than one programmer and evolved over numerous transitions. Although the code did not adhere to usual software design "best practices," he did not find any defects intentionally written to produce inaccurate test results. Geller's review identified three issues with the code: complexity; use of global variables; and the presence of a buffer overflow.

Relying on a software metric known as cyclomatic complexity, Geller measured the number of paths through the code and determined that it was highly complex. He explained that the code's complexity made it more difficult to understand and maintain, which placed an added burden on the programmers. A highly complex code also increased the inherent risks of defects. In Geller's opinion, however, the source code's complexity did not affect the Alcotest's accuracy or cause failures in the interface between software and hardware.

SysTest's review also discovered that the code used a number of global variables. Unlike locally declared variables, global variables were accessible from any function within an application and could be used throughout its duration. Because global variables could be intentionally or unintentionally modified by any function in the application, Geller believed their presence increased the risk of program error and their use

should be limited. Nonetheless, he maintained that the use of global variables did not negatively impact on his opinion that the Alcotest's software was reliable.

Geller, however, did find one serious error in the source code which he identified as a buffer overflow. By attempting to store more bytes or units of information into an allocated variable than space available, the buffer overflow invalidated the reported breath test result on the alcohol influence report (AIR) under well-defined conditions. The error occurred only in very limited circumstances where the first two breath tests were out of tolerance, the subject provided a third breath sample which was within tolerance of each of the other two samples, and the lowest of the six recorded test results was the second breath sample's electrochemical (EC) test result. In these cases, the AIR would not report the lowest breath test result even though it retained and reported the measured alcohol concentration values for the six tests. According to Geller, the buffer overflow error could be easily corrected with one keystroke by replacing the number "four" with a "six" at a particular place in the code.

Wisniewski's review was far more critical of the Alcotest's source code. Base One Technologies (Base One) retained his firm, Winc Research, to determine if the code was scientifically

reliable. Because Wisniewski believed that it was time-prohibitive to test complex software, he relied on industry standards or development methodologies to assess a source code's reliability. In his opinion, the use of these methodologies produced the most error-free and reliable software. Wisniewski found that the Alcotest's source code did not follow any system-wide development methodology. The lack of use of any standards prevented the testing of all the critical paths in the software.

Relying on a program called Lint, Wisniewski identified approximately 19,500 defects in the Alcotest's source code. To insure the code's reliability, he recommended the removal of every defect. While many of the defects reflected poor coding practices or simply bad housekeeping in his opinion, Wisniewski warned that they could cause unintended consequences in other parts of the program. He then identified nine major defects which he claimed could ultimately effect the breath alcohol reading. Wisniewski, however, was unable to say with any reasonable certainty that any of these defects produced a real problem that could influence the test result on an AIR.

Dee testified as a witness for the State at the initial hearing in October 2006, when this court found that he was qualified as an expert in data management business systems and fully credited his testimony. Dee described SysTest as a well-

established company in the computer industry. He was impressed with SysTest's ability to reverse engineer the pseudo source code, and especially with Geller's ability to find the buffer overflow error. Dee did not believe that the source code's complexity affected the instrument's performance and said it simply reflected a tradeoff between performance and ease of maintenance. He concluded that SysTest performed an in-depth review and accepted its finding that there was no evidence of any attempt to maliciously alter the Alcotest's source code.

Dee was not impressed with Base One's analysis. He did not accept Wisniewski's criticism of the code's lack of standards or his use of Lint to quantify errors without considering their quality. Dee continued to maintain, as he did at the initial hearing, that black-box testing was the most appropriate method to determine the Alcotest's reliability. If black-box testing disclosed a problem, then he would examine the source code to see if its logic or a hardware-related error was the cause.

Workman was a licensed engineer before he attended law school. He currently operates a computer forensic business and works as a court-appointed criminal defense attorney in misdemeanor court in Massachusetts, representing clients charged with operating-under-the-influence and other misdemeanors. Like Wisniewski, Workman believed that the source code's complexity

and design made it impossible to test and, therefore, it was not reliable. He agreed with Wisniewski that the source code's reliability would significantly increase if Draeger applied standards to its software development. Workman also criticized Draeger for the lack of any quality assurance organization to test the source code and support Shaffer's programming efforts.

Workman supported Wisniewski's selection of Lint to find source code modules with particular problems. After reviewing the Lint warnings, he concluded that their sheer numbers increased the likelihood of producing a totally wrong result such as an incorrect reporting of a breath test as too high or low, or a sample as insufficient. In his view, the most significant problem identified by Lint involved the Alcotest's use of an unscientific formula in its so-called averaging routine. He understood the routine or algorithm averaged the last breath measurement with the sum of the three previous measurements, minimizing the earlier values. He further supported Wisniewski's finding that the EC and IR sensors did not operate independently despite Draeger's assertions to the contrary.

Workman generally accepted Wisniewski's testimony and concluded that Base One delved more deeply into the source code than SysTest. In his opinion, the Alcotest's source code was

not capable of measuring and accurately reporting the concentration of alcohol in human breath.

Finally, Shaffer testified at the court's request about the technical aspects of writing this source code. He explained that several persons wrote the Alcotest's code, which he also described as complex. Contrary to the defense witnesses, however, he did not believe that a more highly organized and consistently structured code would be more understandable. Shaffer further explained that the instrument's core routines were written in Draeger's offices in Luebeck, Germany, while he was responsible for its customization in the United States including the changes made to the Alcotest after State v. Foley, 370 N.J. Super. 341 (Law Div. 2003). Although the code did not delineate or "wall-off" the core routines, Shaffer was aware of their locations and avoided making any changes to them.

Shaffer performed his own static code review. While acknowledging that there was no dedicated quality assurance in-house designee with respect to software, he stated that Draeger subjected the code to black-box testing by its technical writer, service department, and ultimately the consumer. He disagreed with Base One's assertion that Draeger's lack of use of any standards prevented the testing of critical paths in the software.

Shaffer agreed with several findings by Base One and SysTest. For example, he agreed with Base One's finding that the source code failed to detect catastrophic errors or illegal opcode traps and recommended resetting the microprocessor whenever such a situation arose to restart the test anew. In fact, Shaffer testified that Draeger already had begun implementing this reset feature with its customers in the United States. He also agreed that the source code relied on global variables, but believed their benefits outweighed any potential risks. He further agreed that the code relied on a weighted average routine, but said it was absolutely necessary to assign the greatest weight to the most recent value in order to get an accurate breath test measurement.

Shaffer readily admitted that in error he created the buffer overflow, that it existed only in New Jersey, and that it should be corrected. For pending cases, he prepared a series of instructions to compensate for the overflow and ascertain the true breath test result. He also readily admitted that the fuel cell slowly depleted over time, but he adamantly disagreed that the code's aging compensation routine affected the analysis of a subject's breath. The formula for the depletion of the fuel cell used in the control tests was in part derived from the constant infrared (IR) value but did not corrupt the breath test

results in any way. For this reason, Shaffer insisted that the Alcotest used two independent technologies to analyze breath samples.

The Court will find a review of the summations of counsel very helpful. All summations were carefully prepared and well delivered, and will help greatly in understanding the testimony.

2. Summary of Testimony of Draeger's Expert, Bruce Geller: September 17, 18 and 19, 2007

Draeger presented the testimony of Bruce Geller, a senior software quality engineer with SysTest Labs in Denver, Colorado (1RT18).[1] Geller graduated in 1978 with a degree in biology from the University of Colorado (1RT28-1RT29). After working as an accountant, he went back to school and, in 1992, earned a Bachelor of Science degree in computer science from Metropolitan State College in Denver (1RT29;1RT40). For the past four years, Geller has worked at SysTest (2RT135).

SysTest performs commercial software testing along with independent verification and validation services for private and public entities (1RT20). It also is one of three laboratories in the country which reviews and tests voting system software to verify that it conforms to the Election Assistance Commission's software standards (1RT19;1RT27;2RT162). SysTest has approximately 100

[1] For designation of transcripts, see Appendix A.

employees, including ten source code reviewers, and maintains strategic partnerships with hardware-specific testing laboratories (1RT20-1RT21;2RT138-2RT139).

Geller and another employee, Dan McNamee, reviewed the Alcotest's source code, version NJ 3.11, under the direction of Geoffrey Pollich, a senior products manager (1RT30-1RT31;1RT67). SysTest also hired a professional translator to assist in the translation of the German-language component of the code (1RT31). Although the report was a collaborative effort, Geller approved the final version before it was released (3RT4-3RT5).

During his career in the software industry, Geller has reviewed more than two million lines of source code (1RT27). Prior to this hearing, however, he had never testified in court about a source code review or any other topic nor did he have any experience with breath testing instruments (1RT35;1RT45). We qualified Geller as an expert, noting that as the trier of fact, we would decide the weight of his testimony (1RT45-1RT46).

Draeger retained SysTest to inspect the Alcotest's source code for the consistency of the application of its algorithms — whether the same inputs produced the same results — and any other observable issues (1RT54;2RT149;3RT6). Draeger did not charge SysTest with testing the hardware nor did it give SysTest

an instrument on which to "run" the code (1RT107;2RT53;2RT150-2RT151).

Geller described source code as the human readable version of a program (1RT47). This consists of statements created by a programmer with a text editor or a visual programming tool which are then saved in a file (SysTest report at 6).[2] As in English, source code is written from top to bottom and reads from left to right (1RT47). There is no requirement regarding a source code's organization; instead, this is an issue of style (2RT6). A compiler translates the source code into machine-readable code (2RT57;2RT194).[3]

Source code is segregated into separate functions within certain files (2RT194). A function is a named block of code that contains the instructions to retrieve a specified text screen in a single file (2RT56-2RT57). A call is made to that function providing the index number of the string within the file and the string is retrieved (2RT57). The Alcotest source code contained 504 or 505 functions (2RT103). Some of the functions were left inactive because certain operational aspects were unused under New Jersey's requirements (2RT92-2RT93).

---

[2] See Assessment Report for Draeger Safety Diagnostics, Inc. Alcotest 7110 MKIII-C. New Jersey Firmware Version NJ 3.11 by SysTest Labs, Inc., August 28, 2007 (I-20).

[3] For a detailed explanation of how to write source code, see 3RT9-3RT17.

Using an analogy to a player piano, Geller explained that the piano (or Alcotest) will not work unless a person (or compiler) transcribes the sheet music (or source code) onto a roll with holes (or a binary file composed of bits or non-human readable series of zeros and ones) (1RT47-1RT51). However, unlike a piano roll with multiple holes making concurrent sounds, the Alcotest's microprocessor is single-threaded (1RT49-1RT50;2RT79;2RT170-2RT171). Because everything must pass through the central processing unit (CPU), which can only process one course of logical instructions at a time, the current state of the running application is placed on hold in a stack frame for later retrieval (1RT110-1RT111).

SysTest performed a static code review of New Jersey firmware version 3.11 (2RT9;2RT39). Unlike a dynamic analysis, static code analysis examines computer software without actually executing programs built from that software (2RT11). The purpose of the review was not to translate all 45,000 lines of code, but to examine things of interest (1RT152;2RT17). A component of the static review included reverse engineering of the code by SysTest (2RT12).

To begin the source code review, Geller used Module Finder, an in-house software program developed by SysTest about a year-and-a-half earlier (1RT55-1RT56;2RT155). SysTest created the

program to determine the number and length of the individual functions, the extent of the comments embedded in the functions, and the number of source code lines (1RT55). Geller described Module Finder as a work-in-progress and noted that it did not adhere to any development standards, which made its software more difficult to maintain (2RT155).

Next, Geller processed the code through a software program called Understand for C++ (1RT60). That program produced illustrations of the invocation tree, which showed the order and sequence in which functions might be executed (1RT62;3RT46). Geller analogized the invocation tree to a street map, where rectangles were given function names and lines showed the logical paths from one function to the next (2RT184). The invocation tree served as a guide for reviewing the code (1RT69-1RT70). SysTest also created control flow diagrams, which illustrated the logical paths through each function's execution (2RT184-2RT185).

Geller also ran the code through two additional software programs: (1) C-Doc, which is a C-language-specific source code review tool; and (2) Fortify SCA (1RT68). C-Doc prints out measurements, function lengths and names, the extent of the comments and lines of code, and some information on complexity (1RT68). Because it reports a lot of the same information, he

used C-Doc mostly to verify the metric results from Understand for C++ (1RT68). Unlike those programs, Fortify SCA is a security-oriented application which looks for vulnerabilities in the code (1RT68-1RT69).

There are a multitude of available software programs, and Geller admitted that he was not familiar with many of them (2RT32). He never heard of the Lint program until he read Base One's report (1RT70-1RT71;1RT73;2RT35). From his on-line research, Geller found a January 1979 paper written by S.C. Johnson which explained that "Bell Labs" created Lint at a time when programming was done on large mainframe computers in shared environments (1RT71;2RT41). By enforcing strict syntax rules, Lint warned programmers of possible mistakes in their code (1RT73). Because some of the "mistakes" were not necessarily errors in the software, Lint was known for verbosity (1RT73-1RT75;1RT82-1RT83;1RT85). Geller believed that (1) Lint was created as a development tool, not a review tool; (2) it was outdated after the development of personal computers; and (3) it produced output not particularly informative to programmers (2RT38;3RT63;3RT69). Thus, he believed that the Lint document, contained in Appendix C of SysTest's report, was irrelevant (3RT71).

Geller's review of the source code revealed that it was written by more than one programmer and evolved over numerous transitions (1RT187-1RT188). It contained comments written in German and English, which he explained served to jog the memory of the programmer who wrote them and to advise future programmers (1RT188;3RT13-3RT14;3RT24;3RT133). Geller used the comments to find his way through the code (1RT190).

The review by SysTest led to the identification of three issues with the Alcotest's software: (1) its complexity; (2) its use of global variables; and (3) a buffer overflow (1RT102;1RT108;1RT114). Regarding complexity, Geller explained that some functions had a fairly large number of comparison operations which resulted in "branching" of the code (1RT102). To measure the number of independent paths through the application's code, he used a software measurement developed by Thomas McCabe called "cyclomatic complexity" (2RT98-2RT99).

Because high complexity increases the risk of inherent defects, coding guidelines recommend keeping the cyclomatic complexity of functions under ten, and or even seven (2RT100-2RT101;SysTest report at 23). SysTest identified more than eighty-one modules whose cyclomatic complexity of functions was in excess of ten and three in excess of a hundred (3RT7;SysTest report at 23-35). While restructuring the code could reduce its

complexity, Geller said that the presence of modules with high complexity indices did not effect the instrument's accuracy (2RT102;3RT7-3RT8). He also said that excessive complexity did not cause failures in interfaces between software and hardware (3RT22). Because complexity related to the ease of understanding and maintenance, its presence did place an increased burden on the programmers who work with the software (3RT21).

Geller also addressed the source code's use of global variables to store all test and result data (3RT117). Unlike locally declared variables designed for specific purposes within small subsets of the code, global variables are accessible from any function within an application (1RT108-1RT109;SysTest report at 10). Also unlike local variables which pass from existence after a single use, global variables can be used throughout the duration of the program (3RT110). Thus, global variables make information available without need for the resources required to pass a value from one function to another (2RT85;3RT116).

However, data contained in global variables are not protected and can be changed intentionally or unintentionally at any time by any function in the application (2RT83). Because they can be potentially modified from anywhere, global variables should not be used to store critical data (2RT83). Given the

increased risk of program error, their use should be extremely limited (2RT83;2RT85-2RT88). Nonetheless, Geller said there was nothing inherently wrong with global variables and their presence did not impact negatively on his conclusion that the Alcotest's software was reliable (1RT114). In fact, he said that some computer languages — such as COBOL and assembly — only use global variables (1RT114;2RT90;3RT107-3RT108). Moreover, the preponderance of the code's variables were locally declared (2RT90).

Geller testified that Fortify SCA found a "real error" in the source code which he referred to as a "buffer overflow" (1RT114;1RT116;1RT163). Buffer overflow occurs when a program attempts to store more bytes or units of information into an allocated variable not large enough to hold them (1RT114). Geller used the analogy of trying to park a full-sized Cadillac in a compact-car parking space (1RT114;2RT119). After the computer program warned of a potential overflow, Geller opened the code to the particular file and function, and saw a situation in the source code where six bytes were being parked into a space allocated to hold four (1RT115;1RT118). The two overhanging values were used by the next two declared variables and thus were overridden (1RT116;1RT126-1RT127).

Geller explained that the buffer overflow affected only one aspect of the AIR, and not the way in which measurements were read (1RT118). Specifically, it arose only in very limited circumstances where: (1) the results of the first two breath tests were out of tolerance; (2) a third breath test was taken; (3) the result of the third breath test fell between the values of the first two and was centered enough to be within tolerance of both; and (4) the ER result from the second breath test was the lowest measured value (1RT119-1RT120).[4] In these cases, the AIR would not report the actual lowest breath test result because the sorting routine would read a ".32" in place of the ER result from the second breath test (1RT121-1RT126;2RT124). The overflow problem, however, could easily be corrected with one keystroke by replacing the number "four" with a "six" at the proper place in the code (1RT129-1RT130;2RT124-2RT125). Base One's report did not identify the buffer overflow problem (1RT117).

Geller acknowledged that the code was not written in a manner consistent with usual software design "best practices" (3RT135-3RT137). While such practices are set out in various

[4] As represented by State's counsel at the remand hearing, the record from the initial hearing established there were no instances where tolerances required a third test in the Middlesex County data (1RT145-1RT146).

publications, programmers often have to adapt them to fit the available resources (3RT141). In cases where a code deviated from best practices, it still was safe to use but this placed an added burden on programmers to understand it (3RT140-3RT141). Geller did not find anything in the code that was intentionally written to skew the results (1RT133-1RT134).

Finally, Geller addressed the key findings in Base One's report (1RT135-1RT163). Among other things, he did not know if there were any "industry standards" that governed source code review (1RT136;1RT140;2RT24;2RT51). In Geller's opinion, quality software could be developed without governing standards and conversely, software could meet standards but still be of questionable quality (1RT137;1RT140-1RT141). He was not personally familiar with any of the standards cited in Base One's report nor did he know if any of those standards applied to the Alcotest (2RT50-2RT51).

Overall, he found there was nothing particularly unusual about the Alcotest software in terms of its style and organization (1RT137;3RT76-3RT77). He was unconcerned that the software did not contain confidentiality or copyright notices, or that some sections of the code were not "walled off" (1RT138). Geller explained that programmers had to exercise

caution, but that he never had any classification restrict his access to all or part of a code (1RT138-1RT139).

He also did not agree with Base One's finding that there was proof of incomplete software testing (1RT143-1RT144). Instead, Geller noted that some branches of the source code would never be executed — such as requirements of other jurisdictions — and did not require any testing (1RT144-1RT145). Moreover, he disagreed with Base One's finding that the instrument produced unreliable results because its catastrophic error detection was disabled (1RT147-1RT148). Geller expected that the instrument would go into an endless wait cycle, meaning it would basically cease to function, and not produce a final result in the event of a catastrophic error (1RT148-1RT149). He also objected to Base One's other findings, noting that some of its criticisms were invalid, undocumented, unimportant, not peculiar to the Alcotest, or normal in software (1RT150-1RT163).

Geller fully agreed with all of the findings and conclusions in SysTest's report (3RT5-3RT6;3RT116). Based on his review and the review by SysTest, Geller concluded that the Alcotest software was reliable and consistent when used in accordance with the instructions in the State Police user manual (1RT164;2RT188;3RT6-3RT7). Geller saw no indication of any inconsistencies with the algorithms as documented in the

software, and considered buffer overflow as the only real error (1RT163;3RT5).

We found Geller an honest and technically impressive witness, without bias. We fully credit his testimony.

3. Summary of Testimony of State's Expert, Norman Dee: September 19 and 20, 2007

Norman Dee testified as a State witness at the initial Alcotest remand hearing in October 2006. At that time, we qualified him as an expert in data management business systems (30T60-30T61).[5] We incorporate by reference his testimony on that occasion.

At the State's request, Dee analyzed the expert reports by SysTest and Base One and prepared a report addressing their static code reviews (4RT5;4RT9). He also examined the experts' work materials and unpublished appendices and scanned the code, but only after he wrote his report, when the code finally first became available to him, two weeks before this remand hearing (3RT157;4RT17-4RT18).

Dee had performed static code reviews of embedded systems in the late 1970s and early 1980s (4RT13;4RT137). Although he never wrote embedded code, he reviewed and performed problem determinations of embedded code for large IBM mainframes and

[5] See transcript of October 18, 2006 (morning).

mid-range computers (4RT137;4RT239). He defined his "world" today as one of larger, multi-user systems (4RT187).

Dee described SysTest as a well-known, established company in the computer industry with a reputation for thorough and independent testing of source code (3RT157). Dee was particularly impressed by SysTest's reverse engineering of the pseudo source code, a descriptive English narrative of the algorithms without language-specific syntax (3RT159;4RT15;4RT19;4RT169). In other words, SysTest reviewers read the source code, which is not very human-friendly, and turned it into fairly human-friendly statements by looking at the modules ("the logic itself, the post-compiled code") and typing out the flow (4RT170-4RT171;4RT176). By reverse engineering the pseudo code, which usually was written before the actual code, SysTest was able to identify the main core program and recreate the design of the product (3RT159-3RT160;4RT20;4RT175). Dee also was very impressed with SysTest's ability to find the buffer overflow error; this discovery clearly demonstrated the level of SysTest's competence and the independent nature its review (3RT160;3RT175).

Regarding SysTest's key findings, Dee did not believe that cyclomatic complexity affected the instrument's performance (3RT173). To the contrary, he believed that the presence of

large numbers of "decision trees" in one module was better than breaking them into additional modules in terms of system performance (3RT173). He defined decision trees as boxes containing expressions of code from which branched other boxes, with a "calling program" or "main module" deciding which box to proceed to depending on what the code ordered (4RT181-4RT182). Thus, Dee considered cyclomatic complexity as a stylistic issue which implicated a tradeoff between performance and ease of maintenance (4RT189-4RT190;Dee report at 27).[6]

While Dee recognized the potential significance of buffer overflow, he believed this was a limited vulnerability (3RT175). He described the problem here as a situation where six values must be put into four boxes (4RT193;4RT212). Because of the closed nature of the application in this embedded system, Dee explained that the situation in which this "three tests" error occurred was quite uncommon (3RT175;Dee report at 27).[7] For that reason, he surmised that the problem never arose in field testing and that it took a program tool specialized in exposing vulnerabilities to raise it as an issue (4RT134-4RT135). That program tool, Fortify SCA, checked the source code for

[6] See Comments on the Source Code Reviews by SysTest Labs, Inc. and Base One Technologies, Inc., by Norman Dee, The CMX Group, September 4, 2007 (SR-4).

[7] Dee defined an embedded system as "the combination of hardware and software to perform a specific function" (4RT6).

approximately 150 different kinds of exposure and code issues (4RT236). Dee said, as did Geller, this flaw could easily be fixed (3RT175). He recommended correction in a future release (Dee report at 27).

Dee acknowledged that SysTest's report identified numerous unused or uncalled modules, which consumed memory and space in the source code (3RT187;4RT74-4RT75). While their presence could be due to the carelessness of programmers, he believed they were more typical of software development projects where code was developed and used for multiple customers, was decommissioned, or was developed for future releases (3RT187;4RT74-4RT75;Dee report at 29). Dee explained that it was more convenient to leave these unused functions in the code, than to remove them (3RT187-3RT188;4RT74). In any event, Dee thought their presence was a matter of style and did not effect the reliability or results of the executed program (3RT188).

Dee accepted SysTest's conclusion that its static code review (or "desk checking") did not find anything in the source code that would cause irrational or unreliable results (3RT176). Because static review works with the real code and hypothetical values, it is considered "white-box" testing (3RT167). Dee, however, reiterated his prior testimony that "black-box" testing was the most appropriate method to determine the Alcotest's

reliability and accuracy (3RT197-3RT198;4RT151). If black-box testing disclosed a problem, Dee would want to look into the source code to see if a logic-related or hardware-related error was the cause (4RT43). He assumed that black-box testing did not identify the buffer overflow issue because the underlying circumstances never actually arose in the field (4RT134-4RT135;4RT151-4RT152).

Dee, however, seriously questions many of the findings in Base One's report. For example, he did not think it was impossible to fully test the source code given the singular function of this application (3RT177). He also said it was standard operating procedure in mature systems to disable the capabilities of the processor which detect catastrophic errors (3RT178-3RT179). Dee explained that these aborts were disabled and replaced with software which "captured" the errors so that a determination could be made as to whether the error was recoverable or not, or whether a more meaningful message should be written (3RT179;4RT56).

Dee also disagreed with Base One's statement that the source code ignored or suppressed error messages unless they occurred a large number of consecutive times (3RT179-3RT181). In his opinion, this was normal in embedded systems in order to wait for the coordination of the application with the operating

system or hardware (3RT181;Dee report at 7-8). Thus, the purported error did not mean something was wrong but rather something simply was not ready (3RT181). For example, there might be slight timing differences between internal components which needed adjustment (3RT180-3RT181;Dee report at 8).

Dee also addressed SysTest's and Base One's criticism of the source code's extensive use of global variables (3RT182-3RT184). He agreed that a programmer must be careful to avoid overwriting a global variable (3RT183;4RT204). If an error occurred, a global variable would remain until the system was reset, re-initialized or re-powered whereas a local variable remained only for the specific calculation and then vanished (4RT233-4RT234). On the other hand, the use of global variables conserved memory (as the only place in memory where that variable was found) and increased efficiency by reducing performance time (eliminating the need for multiple copies) (3RT183;4RT70). Also, it is more expensive to change over to all local variables (4RT70). In this event, the presence of global variables did not concern Dee because the source code made extensive use of local variables and only retained those common to all modules at the global level (3RT183-3RT184).

Moreover, Dee disagreed with Base One's criticism that the routines were written in C language rather than assembly

language (3RT186-3RT187). He believed the use of C language would not cause problematic delays or effect the results in any way (3RT187). Also, despite Base One's assertion to the contrary, he found a copyright notice in the first module which he opened (4RT153).

Dee also did not agree with Base One's criticism of the Alcotest's lack of standards (3RT192-3RT193). In his opinion, such standards usually referred to the design and documentation of the code; they rarely addressed the tasks the code actually performed (3RT193;4RT217-4RT218). Moreover, he objected to Base One's reference to standards without citing specific instances in the code where standards were violated (4RT93-4RT94;4RT96).

Finally, Dee was "outraged" when he reviewed Appendix C in Base One's report which purported to find "errors" in 19,000 of the 45,000 lines of code (3RT188-3RT189). Wisniewski found these errors using a pre-compiling syntax checking program called Lint (3RT190). In Dee's view, Lint was a product of the 1970s and was not a commonly-used program today (4RT24;4RT206). He believed that interactive development environments (IDEs) replaced the need for Lint by keeping a programmer within the parameters of the proper syntax during the coding process (3RT190;4RT205-4RT206).

Dee objected to Base One's attempt to quantify the errors (3RT189). For example, Lint generated approximately 7657 lines of warnings based on its misunderstanding that the "U_Byte" variable was undeclared or used incorrectly (3RT189). Dee later explained these lines might have values which truncated the lower digits and retained the higher values (4RT167-4RT168). Also, Lint ignored the quality of the errors, and improperly flagged "comments within comments" (3RT189-3RT190). Based on these alleged errors, Dee believed that Lint did not understand some of the specific code needed for embedded systems (4RT24). He would not have used Lint to review the Alcotest's source code (4RT48).

Dee understood that the source code was written between 1993 and 1997 (4RT78). Although he only scanned the code, he saw about three or four different writing styles (4RT97-4RT98). He did not know if Draeger gave its programmers requirements documents with instructions on how to code (4RT99;4RT180).

Based on his review of the reports, the testimony, and his own experience, Dee was of the opinion that SysTest performed an in-depth review of the source code and produced a professional report (3RT194-3RT195). He found that SysTest was able to reverse engineer, conduct a "fairly good accounting" of the system, and expose the "overflow" error not previously found

through actual field usage of the instrument (3RT195). He also supported SysTest's finding that there was no evidence of any attempt to maliciously alter the code (4RT177). Dee, in the overall, was completely unimpressed with Base One's analysis (3RT195).

We find Dee an impressive witness, as we did at the initial hearing. We give considerable weight to his opinion and find that he was fair and even-handed in all respects.

4. Summary of Testimony of Defendants' Expert, John Wisniewski: September 25, 26, October 9 and 10, 2007

John Wisniewski has a Bachelor of Arts degree in computer science from the State University of New York at Potsdam (6RT197-6RT198). For the past thirty-one years, he has worked as a computer professional primarily in the areas of programming and software development (6RT177;6RT183;6RT202-6RT203;Wisniewski report at 47).[8] In June 1991, he became a free-lance, independent contractor and established Winc Research which he currently operates from his home in Lakeview Terrace, California (6RT188;6RT222;7RT21;7RT37).

Base One Technologies (Base One) retained Winc Research to review the Alcotest's source code for obvious defects and inconsistencies (7RT20;7RT34;7RT44). Base One is a "virtual"

[8] See Report: Alcotest 7110 MK IIIC, by John J. Wisniewski, Base One Technologies, undated (DR-30).

consulting company with its main office in New Rochelle, New York (6RT187;6RT190-6RT191;7RT37). Most of its personnel work from their homes in different cities throughout the country (6RT188). After writing his report with the assistance of a translator in Germany, Wisniewksi sent his draft to Base One's technical writer in Colorado for formatting and editing (7RT37;7RT39-7RT40;8RT215).

The defense offered Wisniewski as an expert in software and hardware development, specifically C language, programming, source code reviews, software troubleshooting, computer interfacing, and embedded systems (7RT19). Based on Wisniewski's education and experience, the court found he was qualified to testify (7RT35).

Wisniewksi maintained that it was time-prohibitive to thoroughly test a complex computer program (7RT190). For that reason, he relied on industry standards to test the reliability of software (7RT190-7RT191). Various standards initially governed how programmers wrote each individual line or segment of code, but Wisniewski described "current standards" as more akin to software development methodologies which applied to the whole system (7RT100-7RT101;8RT216-8RT217). In his opinion, these methodologies produced the most error-free and reliable software (7RT101). They also made software easier to maintain,

produced more robust overall systems, and allowed for testing of all the critical paths (6RT196;7RT98;Wisniewski report at 3-4).

Wisniewski recognized that private industry was slow to adopt the new methods of producing software because they took time to develop (7RT102-7RT103). He claimed, however, that "standards" saved money in the long run (7RT102). For that reason, he said the United States military, some federal agencies, and the European community had developed their own methodologies (7RT103).

Wisniewski identified five widely used software development methodologies: (1) IEC 6158 Functional Safety International Standard, regarding safety of electrical devices and software; (2) ISO 9001 international standard for requirements, regarding the software life cycle; (3) IEC 62304, regarding Federal Drug Administration (FDA) standards for software in medical devices; (4) DOD-178B, regarding Federal Aviation Administration (FAA) devices used on commercial and private aircraft; and (5) DOD-STD-2167 and MIL-STD-498 regarding software used by the military and some government law enforcement agencies (7RT99;7RT191-7RT196;Wisniewski report at 34). Wisniewski recommended that the State of New Jersey refuse to accept any device unless the manufacturer followed one of these five methodologies or developed one of its own (8RT131-8RT132;8RT217;9RT86;9RT125).

According to Wisniewski, the Alcotest's source code did not adhere to any software development methodology (7RT104-7RT105). On cross-examination, he conceded that the code followed a known, function-oriented methodology, but claimed that methodology applied only to software, not the "whole system" (8RT218;8RT224-8RT225). He was unaware of the national breath testing standards promulgated by NHTSA, but understood they did not apply to software (8RT218-8RT219).

Wisniewski described early software development as "bottom-up programming" which focused on details rather than the overall picture (7RT181). Eventually, the industry adopted "structured programming" or the "top-down" approach which started at the highest level and then added more complicated and lower-level functions (7RT180-7RT181). In Wisniewski's view, the Alcotest used a "little bit" of structured programming with a lot of low-level detail (7RT181;Wisniewski report at 33). However, because Draeger did not employ any software development methodology, it was Wisniewski's opinion that the Alcotest's source code was not reliable (7RT193;7RT201).

Wisniewski described source code as human-readable statements which were written in programming language such as assembly or C (7RT63-7RT64;7RT80;DR-15). In C language, a single line of English-looking code could have multiple

instructions (7RT79). In assembly language, however, each instruction was assigned a word called an operation code or opcode which generated one instruction for every line (7RT79). According to Wisniewski, the Alcotest's source code was written in C and assembly languages (7RT159;9RT79-9RT80).

The purpose of source code was to correctly implement the algorithms or mathematical formulas (7RT67). A compiler translated the source code into object code, which consisted of a binary set of machine-readable instructions consisting of ones and zeros (7RT64;DR-15). The microprocessor executed the instructions (7RT79).

When Wisniewski examined the Alcotest's source code, he was unable to distinguish between its core and customized sections (8RT17-8RT18). He believed that programmers should be able to touch the core software in order to learn more about it (8RT17-8RT19;9RT44;9RT46). If Draeger wanted to protect the core, however, he recommended taking the core routines out of the regular code and converting them into libraries of object modules (8RT176-8RT177). The programmers would then get a reference with the library routines, and they would be protected from change (8RT177).

Wisniewski determined that at least three programmers worked on the source code based on stylistic variations

(7RT178-7RT179;8RT103-8RT105). Although the code was created between 1993 and 1997, Wisniewski said it was written in a style reminiscent of the 1970s and 1980s (7RT56;7RT179). He did not view that style observation as a criticism, and acknowledged that Draeger had done a "great job" of adhering to an older style methodology (7RT56-7RT57;7RT193). He further acknowledged that it was not necessary to always adopt the newest technology, and that many people preferred to stay with the familiar (7RT180).

Wisniewski used a variety of tools to review and analyze the Alcotest's source code (7RT158-7RT161). For example, he used the "Understand C Code Analyzer" to find fifty-one uncalled functions, which were either empty pending future release, temporarily disabled, or full but forgotten (7RT161-7RT162;8RT30;8RT167). In Wisniewski's view, these uncalled functions were confusing and untidy (7RT163). Given the possibility that they could confuse future programmers or be executed accidentally, Wisniewski said they should be purged from the executable code (7RT163-7RT165;7RT167). He also observed that there were 475 active functions in ninety-five source files with 26.5% in seven files (8RT175;8RT178). He preferred one function per file (8RT175).

Wisniewski selected Lint to analyze the source code's C language syntax, data initialization, and data management (7RT158;8RT19). Specifically, he used a cost-free program tool derived from Lint called Splint, version 3.1.2, which raised warnings or flags (8RT235;9RT6-9RT7;9RT41). Wisniewski called these warnings "defects" because they required action whether they were serious or simply flaws (8RT26;8RT235). Splint allowed Wisniewski to customize the warning messages by selecting which ones to show (9RT7). He selected the option which displayed all of them (8RT8-8RT9). Wisniewski used Lint to check the source code because it looked across the boundaries of several modules whereas IDEs looked for errors one module at a time (7RT154-7RT155;9RT126).

Wisniewski recognized that Lint was quite verbose because it tended to produce a number of defects including repetitive examples of the same coding style (8RT19-8RT21;9RT39). Despite its "voluminous" output, he maintained that disciplined coders would want to know about the defects and remove them to avoid confusion or any chance the code might not work correctly (8RT19-8RT21). Although the presence of these defects did not prove the software program would fail to execute, they indicated a disregard for use of industry coding standards (Wisniewski report at 37).

Lint found approximately 19,500 defects in the Alcotest's source code, which Wisniewski described as consisting of 35,000 lines after eliminating "comments and other things like that" and 3200 decision paths (8RT29;8RT180;8RT211). To insure the source code's reliability, Wisniewski said every defect should be removed (8RT23-8RT24). He would undertake an aggressive, ongoing campaign to find and dispose of them as part of what he called the software life cycle (8RT27;8RT127-8RT128;9RT123-9RT124). Wisniewski estimated that it would take about one year to fix all the "defects" in the Alcotest's source code (8RT126).

Wisniewksi estimated there were defects in three out of every five lines of the code, ranging from substantive problems to variations in programmer style and organization (8RT180). Some of the defects appeared numerous times, like print interrupts which were flagged about 2000 times (8RT64). The Lint program did not categorize the warnings or flags, nor did it quantify any of them (9RT16;9RT19). Wisniewski did not attempt to fix any of the defects identified in the source code nor did he check to see if they applied to functions actually used in New Jersey (9RT29-9RT30;9RT70).

Some defects simply reflected poor coding practices, in his opinion, such as using a variable as a character in one place and a number in another (8RT31-8RT32). Another example involved

mismatched function argument types where the code expected to see a variable with a plus or minus value, but received only a positive variable (8RT43-8RT44).[9] While not as serious as other defects, Wisniewski said their presence could cause unintended consequences in other parts of the program (8RT32-8RT33;8RT44-8RT45).

Other defects flagged by Lint included the use of a local variable as a global and vice versa, and the assignment of the same name to local and global variables (8RT48-8RT49). Wisniewski described these cases as confusing and inconsistent, and expressed concern that they might influence some other operation in the program such as calibration (8RT49-8RT50). He acknowledged, however, that identically named local and global variables would not confuse the compiler because the local would take precedence over the global declaration (9RT37-9RT38).

Wisniewski also found such defects as: mismatched types (where the computer assigned integers to floating-point variables); memory leaks (where unused memory was taken from the system and not returned); variables assigned different types depending on conditions (where the types of values assigned were inconsistent); and arrays initialized with too many variables

[9] Wisniewski defined an argument as "something passed to a function to allow it to take different paths or make different decisions" (8RT64).

(where there were too many variables to fit into the declared space) (8RT50-8RT59).[10] Wisniewski believed these defects could ultimately effect some calculation which, in turn, could effect the breath alcohol reading (8RT58-8RT59). However, he was unable to determine, by desk checking alone, if these defects had corrupted any critical values (8RT74). He would need an Alcotest instrument and an emulator to "run" the code to see how it performs (8RT74-8RT75). In any event, he said many of the defects were simply bad housekeeping and extraneous, and should be removed (8RT73-8RT74).

Wisniewski also testified about inconsistencies between the code and corresponding comments (8RT118). For example, he found a comment in the code stating that a conversion to "%BAC" needed to be performed, but this was not done (8RT116-8RT117;Wisniewski report at 17). He also found comments which said values should be averaged when, in fact, he claimed the source code performed weighted averages or successive averaging routines (8RT118). Wisniewski said such comments could affect the breath test result if they were unintentionally executed; otherwise, they reflected a sloppy coding style (9RT18). He agreed, however, that comments were not compiled, never reached

[10] For a discussion of errors detected by Lint, see 8RT43-8RT98;Wisniewski report at 37-43.

the object code, and did not effect the performance of the Alcotest (9RT18). Therefore, they would not affect the Alcotest's performance if placed correctly within the source code (9RT18).

Of the many defects identified by Lint, Wisniewski selected nine with the greatest impact on the Alcotest's test results: (1) the software would not pass industry standards for development and testing; (2) the lack of use of industry coding standards prevented the testing of all critical paths in the software; (3) the catastrophic error detection was disabled, making it difficult to detect if the software was executing indefinite branching or invalid code; (4) the implemented design lacked positive feedback; (5) the diagnostic routines were performed during data measurement cycles, allowing the substitution of arbitrary data values when a routine failed; (6) the air flow readings were adjusted at the beginning of the measurement, causing defective measurements when the baseline value was corrupted; (7) the error detection logic failed to flag an error unless it occurred thirty-two times; (8) the heavy use of global variables failed to insulate software modules; and (9) the software instructions were out-of-phase with the continuously operating timer interrupt routine, which went off

every 8.192 milliseconds (8RT110-8RT116;8RT120-8RT125;8RT134-8RT141;8RT152-8RT166;Wisniewski report at 3-6).

At this remand hearing, however, Wisniewski was unable to find an illustration of diagnostics adjust/substitute data readings (8RT137;9RT66-9RT67). He also admitted on cross-examination that the use of global variables was a tradeoff, stating that fewer globals would result in more functions with arguments passed but more variables protected (9RT76-9RT78). He further admitted that time constraints prevented him from determining if a global variable was misused and could actually change the result on an AIR (9RT78). Indeed, he was unable to identify anything in the code that posed a real problem that would effect a result on the AIR (9RT86-9RT87).

Wisniewksi also raised an issue regarding the independence of the IR and EC measurements (8RT182-8RT198). He found a section in the code where the IR reading mathematically modified the EC reading (8RT183-8RT184;8RT191). That section could be called or activated from seven different paths in the code (8RT190). Thus, under certain conditions, the code would take the results of the calculations under the EC curve and divide them by the IR average (8RT187;8RT191). In Wisniewski's opinion, the Alcotest's source code should not be accepted as scientifically reliable in his field (8RT199).

We were not particularly impressed with Wisniewski's testimony. He was very negative and deconstructive. He said many things were wrong but did not convince us that these negatives made the Alcotest unreliable. We doubt that he was as experienced as he portrayed.

5. Summary of Testimony of Defendants' Expert, Thomas E. Workman, Jr.: October 10 and 11, 2007

Thomas E. Workman, Jr. has Bachelor and Master of Science degrees in electrical engineering from the University of Texas at Austin (1970 and 1974), and a JD degree with a high technology law concentration from Suffolk University Law School in Boston, Massachusetts (1997) (9RT151;9RT176). He is a licensed patent attorney and admitted to practice before the U.S. Patent Office (9RT151).

Workman was an engineer for over twenty years with various technology-based companies including Thinking Machines Corporation, Digital Equipment, Hewlett-Packard, Xerox Corporation, Austron Corporation, and Texas Instruments (9RT152-9RT155). He also worked as an independent consultant on projects which developed embedded systems primarily for law enforcement and communication customs software for remote job emulators (9RT155). Workman has experience in software engineering, quality assurance, systems verification, and standards (9RT163-9RT164;9RT168-9RT172).

For example, at Hewlett-Packard, Workman was co-chair of a working group for the Institute of Electrical and Electronic Engineers (IEEE) which promulgated a standard for measuring software reliability (9RT170;DR-15). That standard, IEEE 982.1, was voluntary and recommended a classification scheme for severity and class of defect (10RT199). At a presentation to one of the IEEE standards boards, Workman observed that unless some step was taken to improve the reliability of software, the number of problems would double every two years as a result of the computer operating twice as fast (9RT174-9RT175). His observation became codified within the IEEE as Workman's law of software reliability (9RT175).

Workman currently practices law, provides expert testimony, and operates a computer forensic business (9RT183;10RT206). He primarily works as a court-appointed criminal defense attorney in misdemeanor court in Massachusetts for clients charged with operating-under-the-influence (OUI), assault and batteries, and other misdemeanors (9RT183). In his "spare time," he performs as a classical singer at such venues as Carnegie Hall (9RT185;10RT206).

Workman has qualified as an expert in multiple subject areas in fifteen to twenty cases, two on behalf of the prosecution (10RT201-10RT202). He has testified as an expert on

source code for breath testing instruments in Arizona and Georgia, and prepared for a case in New Hampshire (9RT180-9RT181). He is scheduled to testify in Tennessee, South Carolina, Georgia, Arizona and California (9RT181). Except for here in New Jersey, most of his other testimony involved issues relating to the production of source code for discovery (10RT214). In Arizona and Florida, he worked on cases where the court ordered CMI, Inc. to produce the source code for Intoxilyzers 5000 and 8000 (9RT204).

The defense here offered Workman as an expert in source code review and the application of standards (9RT186). The court found him qualified to offer testimony in engineering by education and in the other areas by work experience (9RT186-9RT187).

Based on his education, background, experience and understanding of the Alcotest from exhibits introduced at the remand hearing, Workman offered the opinion that the Alcotest was not a reliable instrument on human subjects (9RT187). He believed the source code's complexity and design made it impossible to test (9RT195-9RT196). However, because he would not enter into a non-disclosure agreement with Draeger, Workman never saw the source code except for several snippets introduced in evidence at this remand hearing (10RT59;10RT183;10RT198).

Both SysTest and Base One identified the software's complexity as a major issue (9RT195). Whereas Base One concluded that the software was too complex to test, SysTest relied on the cyclomatic complexity metric developed by Thomas McCabe in 1976 to measure the number of potential paths through the code (9RT195-9RT196;9RT199-9RT200;9RT211). While acknowledging that McCabe wrote in 1976 that a cyclomatic complexity of ten was not a magical upper limit, Workman maintained that modules with excessive McCabe metric scores were overly complex (9RT196-9RT197;9RT199-9RT201;10RT187;Workman supplemental report at 4).[11]

Workman also described another software metric called the Halstead Metric, which measured a routine's data complexity (9RT200). This metric measured the number of operands (things that are operated on) and operators (how the operands or data are manipulated) to provide a single number (9RT200;DR-15).

In Workman's view, the Alcotest's software was "far too complex" to test (9RT200-9RT201). Therefore, its reliability could only be determined retrospectively based on the occurrences of failures (9RT201). He said the problem could be corrected by re-partitioning the routines so there was a more

[11] See Supplemental Report by Thomas E. Workman, Jr., October 4, 2007 (DR-31).

manageable number of paths through particular functions (9RT211).

Workman acknowledged that it was impossible to write perfect source code (9RT201;10RT215). First, human beings, by nature, were fallible (9RT201). Second, specifications changed over time in response to new regulations and legislation (9RT201). Third, codes — like the Alcotest's — contained trillions of paths which made it impossible to find and fix all the errors (10RT33). Indeed, Workman estimated it would require all of mankind for the rest of time to test all the paths in the Alcotest's source code (10RT33). Nonetheless, Workman thought it was possible to achieve 99.98% reliability by applying standards to software development (9RT202-9RT203;10RT33-10RT34). He did not know if any breath testing instruments on the market had achieved that level of reliability (9RT204).

To make the Alcotest reliable, Draeger would need to develop standards that would dictate the complexity of the modules and discourage the use of global variables (10RT33-10RT34). Such standards also would establish testing processes and procedures, which Workman believed would have detected the buffer overflow problem (10RT34).

Programmers make mistakes all the time (10RT34). For that reason, companies relied on their quality assurance

organizations to test source codes and determine if they adhered to standards (10RT34-10RT35). While he considered Shaffer a good programmer, Workman saw no evidence that Shaffer had the support of such an organization within Draeger to review and test the code (10RT34).

Workman defined source code review as an inspection method for identifying and documenting problems (9RT209-9RT211). Unlike desk checking, source code review was a more rigorous process typically performed by someone other than the author (9RT210). To review code, Workman would: (1) use a static tool, like Lint, to find source code modules with particular problems that needed to be investigated; (2) evaluate the build process to determine how the code was assembled and identify what source code went into the modules; and (3) methodically test the sections of code which were most likely to have problems and yield useful results (9RT213;9RT218;10RT38-10RT39). For example, Wisniewski looked at the interrupt handlers, timing routines, and the algorithms purporting to average the samples, and found significant problems (10RT39). Workman described this type of review as "static code analysis" because it did not involve the execution of the code (9RT218).

Workman described Lint as a generic term for a class of tools that performed static code analysis (9RT220-9RT221). Lint

is designed to find problems in source code; IDEs facilitated the writing and testing of code for a particular environment by providing tools such as a programming editor, a compiler, a link editor, and often a debugger (9RT223). Most Lint programs were shareware, meaning they were cost-free, while others had fairly modest fees (9RT222). Splint was a variant of Lint that focused primarily on security issues relating to coding errors (9RT222). Lint and Splint functioned on C language source code (9RT222).

Workman explained that people committed errors in writing code by acts of commission or omission (9RT229). Errors resulted in defects which existed in the lines of source code (9RT229). When the microprocessor executed a defect in the code, a fault occurred (9RT230-9RT231). A fault meant that the computer was doing something unintended or wrong, which could result in a failure to perform the desired specification (9RT231;9RT235-9RT236). Because the software development process was imperfect, there were always some defects and failures (10RT7).

Workman reviewed the warnings or errors flagged by Lint on the Alcotest's source code (9RT224-9RT225). Among other things, Lint found prolific "u_byte" errors, which meant that a byte variable was being loaded with a number too large to fit within eight bits (10RT59-10RT60). For example, in base two, the

number of values that could fit into a byte was 255 or 2 to the eighth minus one which accommodated for zero (10RT60-10RT61). When too much data was assigned to a byte, Lint raised a warning because of the risk of losing data and causing a wrong reading (10RT63-10RT65).

Lint also found errors involving mismatched functions against type, meaning a function was expecting a variable of one type and was passed a variable of a different type (10RT66). In Workman's opinion, such errors might produce totally wrong results (10RT67-10RT68). He also agreed with Base One's finding that the source code had timing problems, explaining the difficulties arose from the use of two different clocks within the instrument plus a realtime clock which kept track of date and time (10RT88). As an example, Workman mentioned that Draeger did not add a new daylight savings variable in its code to anticipate the recent legislative change (10RT88-10RT91).

He further agreed with Base One's finding that the lack of positive feedback in the hardware did not give the source code the proper tools to do its job successfully, citing the inability to confirm that the pump worked properly (10RT98-10RT99). In Workman’s view, the absence of positive feedback made it impossible to demonstrate the Alcotest's reliability (10RT99-10RT100). He also considered the notion of ignoring

thirty-one consecutive errors before reporting an error message as "junk science" (10RT130).

Workman recognized that there was an error re-insertion rate in the industry between twenty to seventy percent, but estimated it was on the high end for the Alcotest because of its lack of use of standards (10RT96). There also was a greater probability of creating new problems given the complexity of the source code (10RT97-10RT98). He cited the case where Draeger added a new capability to find the minimum value of six breath samples and inadvertently created the buffer overflow problem (10RT95;DR-4).

Workman believed the most significant problem uncovered by Lint was the Alcotest's averaging routine (10RT69-10RT70). Instead of computing a simple average by adding together a set of numbers and dividing by the number in the distribution, a weighted average took into account the number of times each value was present (10RT74;10RT80;DR-15). While Draeger claimed to average the data points from the continuum of IR measurements of the alcohol content in human breath, it actually averaged the last measurement along the continuum with the sum of the three earlier measurements (10RT71-10RT74). By minimizing the earlier values (1/6 each) and giving half the weight to the final value, Workman said the formula was scientifically unreliable as an

average or weighted average (10RT77;10RT83;10RT85-10RT87;10RT121-10RT122;10RT173).

While acknowledging that the final point was a valid reading, Workman maintained it should have no more weight than the earlier values (10RT177). He recognized, however, other instances where later values were given greater weight because they were more important, referring to the Bayesian formula used to predict the future based on past events (10RT159-10RT160).

Workman also agreed with Base One's finding that the EC and IR sensors did not operate independently as represented by Draeger (10RT129). He explained that fuel cells were very common devices which deteriorated over time until at some point they ceased to function (10RT142;10RT144). He described this process as a function of time and the fuel cell's use (10RT145). As fuel cells drifted, they did not give the same output, just as a battery flashlight becomes weaker with time (10RT142).

When the Alcotest's fuel cell drifted out of tolerance, the instrument used an IR value to compute an electrochemical result (10RT132-10RT133;10RT136;DR-14). Workman believed that this adjustment was made in the first and last control tests based on Wisniewski's finding that it was called from seven different places within the code (10RT141). Even if the adjustment was made only in the first control test, it would affect everything

that followed it because the adjusted EC was used in the ambient air blanks and succeeding breath tests (10RT149). Workman could not find any warning about fuel cell drift in Draeger's operator manual (10RT144;10RT154). He would "fix" the problem by stopping the test, shutting the machine down, and putting out a message that the fuel cell had drifted out of tolerance and required replacement (10RT144-10RT145).

Although the probability that any one problem would result in a failed AIR was small, the large number of warnings identified by Lint increased the likelihood of such an outcome (9RT224-9RT225). Workman raised the probability that an error could incorrectly report a breath test as too high or low, or a sample as insufficient (9RT225). It also could incorrectly find a third test was not necessary or result in global variables being overwritten so that AIRs were printed without such information as the expiration date or solution control lot as in the Longport example (9RT225-9RT226;10RT102;AB-2). Moreover, an AIR could appear valid on its face, when it really was invalid (10RT152).

For example, Workman testified about a series of AIRs from the East Brunswick, Milltown, and South River Police Departments

(10RT9-10RT11;D-129).[12] All three tests were administered by the same officer on May 15, 2006, on three different instruments (10RT14;10RT17-10RT18). The East Brunswick and Milltown readings were taken at 4:03 a.m. and 4:36 a.m., and both results were zero (10RT13-10RT14). The third test was given in South River, at 5:14 a.m., with a reading of .14 BAC (10RT14-10RT15). Workman believed a software defect caused the underreporting on the two AIRs (10RT15). While the problem could be hardware-related, he believed the coincidence had to be very high for hardware to fail in exactly the same way in two different instruments (10RT42-10RT43). While he acknowledged the problem also could be caused by operator error, Workman posited that even if there was a sucking-back problem, the software should have detected it and produced an error message (10RT44).[13]

Instead of Lint, Workman observed that SysTest relied on a different tool called Fortify to look for security defects (10RT39-10RT40). Fortify was designed to look for malware or viruses that might exist in the software (10RT40). Workman

[12] Exhibits with a "D" designation refer to exhibits marked into evidence by the defense at the initial hearing before the Special Master in this matter.

[13] State's counsel represented that at the initial hearing, Sergeant Kevin Flanagan said the underreporting was caused by the subject who was sucking air into the instrument (10RT29-10RT30).

thought this was an inappropriate tool because the user did not touch the interface to the software (10RT40). Both SysTest and Base One used another tool called Understand C++, which provided information about the complexity of routines such as the number of global variables and uncalled modules (10RT40). Workman thought this tool was appropriate (10RT40).

Unlike other computer-dependent industries, Workman expressed concern that there was no easy access to Draeger when problems occurred in the field (10RT20-10RT21). There was no button on the Alcotest which could be pressed to alert the manufacturer of a problem and no evidence of data logs (10RT21). He acknowledged on cross-examination, however, that he did not review the actual code and seemed unaware of the data log functions in the instrument, as related by Shaffer (10RT183-10RT184).

Moreover, New Jersey did not maintain a centralized data base in contrast to Alabama, which logged over 200,000 breath tests, or Massachusetts (10RT27-10RT28). Workman observed that the forensic breath testing field did not encourage the reporting of problems, that state organizations had limited skills in software and computer science, and that police officers often had even fewer skills (10RT23-10RT25). He also was highly critical of New Jersey's operation (10RT24-10RT25).

Workman found nothing really wrong with Wisniewski's testimony, although he might have done things a "little bit" differently (10RT200). He said that Wisniewski delved significantly deeper into the code than Geller (10RT224-10RT225). While both experts identified the code's use of global variables and its excessive complexity, he thought Wisniewski was the only one who properly testified about their consequences (10RT225). Because Wisniewski concluded that the Alcotest was not scientifically reliable, he could not contemplate any method to distinguish between correct and incorrect test results in the pending cases (10RT147).

The court heard Workman's testimony under R. 1:7-3, which provides in relevant part that in actions tried without a jury, a court shall permit the evidence to be taken down by a court reporter in full unless it was not admissible on any ground, a valid claim of privilege was asserted or the interest of justice required otherwise. This court found that Workman was qualified to voice his opinion on technical, computer and legal matters. The weight was for the court.

We did not find Workman's testimony persuasive on the point of Alcotest's unsuitability. He did not convince us that its hardware or software was inappropriate. His suggestion that the EC cell should be replaced on an indication of depletion may

have some merit and could be considered by the Court as a correction of the current practice and program.

6. Summary of Testimony of Court's Witness, Brian Shaffer: September 24 and 25, 2007, and October 11, 2007

Brian Shaffer received a Bachelor of Science degree in electrical engineering in 1992 from the University of Pittsburgh (5RT5). After working nine years in the semiconductor industry as a product test engineer, he spent one year as a design engineer for an electronics company which served the hobby industry (5RT5). In July 2003 Shaffer joined Draeger in Durango, Colorado as a software engineer (5RT5;5RT56;5RT38). He currently works with evidential table-top instruments, primarily the Alcotest 7110 and 9510 (5RT38;5RT124).

Shaffer has written source code for Alcotest instruments used in California, Massachusetts, Alabama, and New Jersey (6RT32-6RT33). In New Jersey, he wrote the post-Foley changes into the code which appeared in version NJ 3.11 (5RT5-5RT6). Norbert Schwarz is his primary Draeger colleague in Luebeck, Germany (5RT19). Shaffer prepared no written report, as he was called by the court as a witness (5RT4;5RT6). Although he was a fact witness, the court also found him qualified as an expert on source code writing (5RT53).

The Alcotest's source code consists of core routines and customized tasks designed around them (5RT10-5RT11;5RT175-5RT176). A compiler translates the source code into instructions which the microprocessor follows to complete the sequence and print a result (5RT8).[14] The software, however, cannot function without the critical hardware (5RT11).

The core routines in the Alcotest relate directly to the measurement of alcohol (5RT178-5RT179). Because these analytical algorithms were tested many times in different applications around the world, Shaffer avoided altering them (5RT18-5RT19;5RT93). The instrument also was tested by the National Highway Traffic Safety Administration (NHTSA), which would require re-testing if any changes were made to the core routines (5RT18-5RT19;5RT93-5RT95). While the code did not delineate or "wall-off" these sections of code, Shaffer was alerted to their presence by comments from previous developers, and discussions with Ryser and Shaffer's own engineering supervisor (5RT93-5RT94). Shaffer acknowledged it would be easier to find the core routines if they were documented in the code, but did not believe this was necessary (6RT28-6RT29).

[14] The Alcotest uses three microprocessor chips: Motorola; Toshiba; and a low-voltage version of the Motorola device (6RT136-6RT137).

These routines were the same in instruments used in New Jersey, Alabama, Massachusetts, and California (6RT41).

Shaffer described the Alcotest's source code as complex with various styles of syntax (5RT148;6RT28). While a highly organized and consistently structured presentation would make the source code more readable, Shaffer did not believe this would make the code more understandable (6RT28-6RT30).

The source code was written in assembly and C program languages (5RT36). The core algorithms were written in Germany, and the customized ones here in the States (5RT93;5RT98). Shaffer customized various tasks for New Jersey including display prompts, external printouts, removal of internal printout functions, modifications of tolerance agreements, test sequence changes, and data memory (5RT179-5RT180).

In Shaffer's opinion, there was nothing proprietary about a very common algorithm (5RT13). However, software developers and scientists worked very hard and invested a lot of time and money to develop routines to create breath test measurements (5RT13-5RT14). If the Alcotest's source code was openly available, competitors could use Draeger's hardware and software to create "knock-offs" (5RT14). They also could use Draeger's technology to create their own products (5RT14). For example, anyone who marketed products with fuel cells might be interested in the way

that Draeger captured data from the Alcotest's electrochemical sensor (6RT133). Shaffer was unaware of any companies that openly published their source codes or of any instance where he personally shared code that he wrote (5RT14-5RT15).

After writing source code, Shaffer performed his own static code review or "desk checking" (5RT192). He also conducted "black-box testing with white-box knowledge" by exercising certain logical paths through the code to confirm that these paths worked as he intended (5RT192-5RT193). A technical writer then conducted black-box testing to find out how the test sequence performed and if it met the customer's requirements, and documented the procedures (5RT193-5RT194;5RT197). The service department next performed black-box testing to determine if the code supported its service capabilities (5RT194;5RT197). Finally, the customer performed user acceptance verification testing (5RT197).

Draeger did not have a dedicated quality assurance person or anyone who functioned in that role with respect to software (6RT38). Shaffer admitted that he would have a higher degree of certainty about the source code if another person participated in the code review process (5RT195-5RT196).

Shaffer was unaware of any single industry standard for software development (5RT15-5RT16). Instead, he referred to

"industry standards" as collections of techniques and common-sense wisdom which had proved effective over time (5RT16). During his career, Shaffer collected his own set of development standards, albeit unwritten (5RT16;5RT144-5RT145). He was not familiar with the ISO 9000 standards for software (6RT38).

Shaffer did not agree with Base One's assertion that the failure to use industry coding standards prevented the testing of critical paths in the Alcotest's software including 3200 lines of code designed to make decisions (5RT17-5RT20). Because the Alcotest in the United States was highly configured to meet the requirements of specific applications, all of the 3200 lines of decision code — as calculated by Base One — were not relevant (5RT18;6RT152-6RT153). Shaffer also said there were by design many unused or uncalled modules or sections of code (5RT17).

Shaffer, however, agreed with Base One's finding that the source code failed to detect catastrophic errors (5RT20-5RT23). He explained that when the microprocessor encountered a command or a memory location that it did not recognize, such as when an instruction in the stack or temporary memory area became corrupted, the microprocessor would lose its place in the script and jump to another section of code (5RT20;5RT150;5RT216;6RT44-6RT45). When the microprocessor attempted to execute the code

at the new location, it would become confused and fail to respond appropriately (5RT22;5RT216-5RT217).

Shaffer used the term "illegal opcode trap" to describe this scenario and considered it highly likely that the hardware would "lock up" or freeze so that it would be impossible to finish a breath testing sequence (5RT21-5RT22;5RT217-5RT218;6RT37). Because the operator would immediately become aware of the situation, there would be no risk to the subject of a false reading (5RT22-5RT23). Nonetheless, Shaffer recommended resetting the microprocessor whenever the instrument detected an illegal opcode trap by clearing the memory and starting anew as if the instrument had been turned off (5RT20-5RT21;5RT215;6RT36-6RT37;6RT98). Based on Shaffer's discussions with his engineering colleagues in Germany, Draeger already has begun to implement this reset feature with its customers in the United States (6RT36;6RT40).

Shaffer disputed Base One's finding that the implemented design lacked positive feedback (5RT23). He explained that there was a direct or indirect way of monitoring the functioning of every circuit, sensor or electrochemical device in the Alcotest (5RT23-5RT24). For example, a problem with the IR detector on either end of the cuvette would be directly observable because during the operational cycle every 8.192

milliseconds, the measurements would immediately drop below the minimum threshold and the instrument would flag a hardware error (5RT24). Likewise, if the pump or the solenoids were not in the proper position, the problem would be indirectly observable because air would not flow past the sensor at the appropriate times (5RT25).

Shaffer was uncertain about Base One's findings on diagnostic adjustments and substitute data readings (5RT26-5RT27). He explained that the Alcotest performed diagnostic checks every 8.192 milliseconds or 122 times a second, including when a subject was blowing and results were analyzed (5RT26;5RT82-5RT83;5RT141). Unlike other customers, New Jersey did not ask Draeger to take "diagnostic snapshots" and store them in the instrument's memory as part of the data log (5RT26). Nonetheless, if a diagnostic routine failed in New Jersey's firmware version, the instrument would generate a hardware error which would halt its operation and make further tests impossible (5RT27). Contrary to Base One's assertion, Shaffer never saw an instance where a diagnostic routine failed and the Alcotest substituted "canned" or arbitrary data values (5RT27).

Shaffer also disagreed with Base One's criticism about flow measurement (5RT27-5RT29). At the beginning of the power-on cycle which started a breath test measurement, the Alcotest

assumed that the airflow was zero without conducting a "reasonableness check" (5RT27-5RT28). This "zeroing" of the instrument, however, was offset by many real-time checks which made certain that the instrument was working within its prescribed ranges (5RT28-5RT29).

Shaffer said it was common practice in electrical engineering to ignore error messages unless they occurred a large number of consecutive times (5RT30-5RT31). Indeed, he mentioned several advantages of the Alcotest's requirement that measurement errors had to occur thirty-two consecutive times before they were reported (5RT30-5RT31). Shaffer explained that all sensors had a natural range of values and that it would be surprising for them to rely on only one decision point (5RT31). The use of thirty-two events also allowed Draeger to set tighter tolerance ranges to avoid falsely triggering errors (5RT31-5RT32). For example, this meant that if a subject blew into a breath hose before the operator pressed the button to start the test, the instrument would not flag "blowing not allowed" if the subject blew for less than one-quarter of a second (thirty-two times 8.192 milliseconds), but would display the error message if the subject blew one-quarter second or longer (5RT30-5RT31).

Shaffer estimated that the Alcotest's source code contained approximately 200 global variables and 1500 local variables,

which he described as a lot of variables in general (5RT32). He described the use of global variables as a tradeoff (5RT33). On the downside, they placed an additional burden on programmers to exercise caution when adjusting the code to avoid unintended consequences such as overriding local variables or assigning to a new module a name already used by a global variable (5RT33;6RT14). On the upside, global variables were easily accessible in all modules of the program, and contained information for the use of such things as calibration which Shaffer wanted the instrument to remember after it was powered off (5RT33-5RT34). Their use also resulted in far less complexity and overhead, and made the code easier to design (5RT117).

Shaffer believed the advantages of global variables outweighed the risks (5RT33). Moreover, to reduce their number, he would have to add more code and functions which, in turn, would create higher complexity (6RT135). He also would have to touch more portions of the code than otherwise necessary (6RT135). Shaffer emphasized that the decision to use global variables was made in the design process and that he did not consider their presence as a liability for the product (5RT34).

Shaffer also took exception with Base One's identification of timing problems (5RT34-5RT36). Because the Alcotest had a

separate real-time chip on the motherboard that could be accessed for any evidential time stamp, he considered the clock free-running and independent of the microprocessor (5RT34). He also explained that the clock was used for administrative functions, and was not absolute (5RT34-5RT35). He specifically objected to the characterization of the external interrupt routines as very lengthy, stating they handled many functions by design, and to the representation that they were written exclusively in C language when, in fact, portions were written in assembly language (5RT36).

Shaffer agreed that headers could be used to identify the last time a section of code was modified and the name of the programmer who made the change (5RT119). However, he did not consider them a priority because he usually worked alone or as part of a very small engineering team (5RT120). If he needed to determine when a module was changed, he simply would compare previous versions of the code using other tools (5RT120;5RT190). For example, he used "diff programs" which highlighted the lines of code that were changed and the way they were changed, allowing him to interpolate file creation dates (6RT22). If he worked with a larger engineering team, Shaffer acknowledged that the header comments would have far more value (5RT120).

Shaffer defended the Alcotest's use of weighted averages, stating it was absolutely appropriate to assign the greatest weight to the most recent value (taken from the sample with the deepest lung air) for the purpose of making a very accurate breath test measurement (5RT136-5RT138;6RT144-6RT147;10RT229). He explained that individual samples of breath from the IR measurement were taken every 8.192 milliseconds but that the weighted average routine only considered the points derived from them every .25 seconds (10RT228). By relying on samples taken at .25-second intervals, the weighted average was actually less than the value of the last reading (10RT227-10RT230).

Likewise, Shaffer took issue with Base One's finding that results were limited to small discrete values (5RT139-5RT140). Specifically, Base One found that there were only eight values possible for the IR detector and sixteen for the EC sensor (5RT140). Shaffer, however, said the range of possible values was significantly higher, with the IR about 12,000 and the EC as low as 100 and as high as the thousands (6RT139-6RT140). By multiplying 4096 — all possible values that can be observed from the IR system — by a sine wave, the IR value could be as high as 22,000, which gave tremendous precision (5RT139).

Shaffer defined "a defect" as anything which does not work in accordance with the specifications (5RT126;6RT30). He

expected to see defects in the development process, and estimated an average of one defect in each version of the source code sent to customers (6RT31-6RT32). By his definition, defects might include typographical errors, misunderstandings about specifications, or anything else which caused some undesired result (6RT32-6RT35).

Shaffer was the creator of the buffer overflow defect (5RT39-5RT40;5RT154). He was quite surprised and impressed that SysTest found this problem which had remained undetected despite significant white-box and black-box testing (5RT49-5RT50). Shaffer inadvertently introduced the buffer overflow when he implemented the post-Foley third-test changes requested by New Jersey (6RT19-6RT20). Specifically, he added a section of code without changing the initialization of the variable to allow it to accommodate six instead of four values (6RT19;6RT94). Thus, the code allocated four spaces for data when it needed six (6RT95). The sorting routine that created the buffer overflow occurred only in New Jersey, not in other jurisdictions (10RT231-10RT232;10RT246).

Buffer overflows can have far-reaching effects (5RT50). After studying the specifics of this particular overflow, Shaffer concluded that it occurred only in limited situations where (1) breath tests one and two were not in tolerance with

each other, (2) a third test was required, and (3) the EC result of the second breath test was the lowest of the six values (5RT46-5RT48). In these cases, the code only allowed the instrument to allocate to its "temporary scratch pad" four of the six values within the locally declared variable (5RT40-5RT42). By overwriting the true lowest value as a .32, the instrument was unable to recognize that EC second test value as the lowest when the software went through the sorting routine (5RT43-5RT46). Consequently, the instrument reported the second-lowest value for the breath test result (5RT46). The overflow error did not affect the six alcohol breath test results printed on the AIR, which came from global variables (5RT42). To correct this buffer overflow, Shaffer simply would change the number four to six at the appropriate place in the code and then recompile the code (5RT49).

Shaffer did not think it necessary to exclude the use of the AIR in all pending third-test cases (6RT120-6RT121). Instead, he crafted a series of instructions to determine if the buffer overflow had an effect and to find the true reported breath test result (6RT121-6RT122;10RT234-10RT235;CR-3). The instructions consisted of twelve discrete mathematical operations shown in green on exhibit CR-3 involving addition, subtraction, multiplication and division, and some other steps

involving basic comparisons or copying from other lines (10RT235). Shaffer explained that these calculations were necessary to determine if breath test three was within tolerance of breath tests one and two, and that it would be incorrect to select the lowest of the six unaffected test results on the AIR (10RT237-10RT238).

Shaffer understood there were no cases in Middlesex County in 2005 which required a third breath test due to lack of tolerance (5RT46-5RT47;5RT155-5RT156). He further understood that some third tests would be generated when New Jersey tightened its tolerance requirement (5RT156). Shaffer, therefore, recommended that New Jersey include the buffer overflow correction on its change request list (5RT184). According to Shaffer, the only other defect pending in New Jersey occurred in very specific circumstances where an instrument did not wait quite the full two-minute period between subject samples (5RT132-5RT133). He also recommended correcting this defect (6RT116).

In contrast to Workman's testimony, Shaffer stated that data logs were part of the Alcotest's source code and were enabled in version 3.11 (10RT230). These logs stored data within the instrument's memory such as the time stamp of each event which occurred within the breath testing sequence, the

individual IR and EC results, and the aborted tests (10RT230-10RT231). This data could be retrieved from the memory of each instrument (10RT231). New Jersey also could retrieve this data on a statewide basis, but it has chosen not to do so up to the present (10RT231).

Shaffer also testified that the fuel cell changes or depletes throughout its life (6RT104). Because older fuel cells tended to underreport the ethanol level, engineers in Germany inserted into the source code an algorithm or aging compensation routine to address this drift over time (5RT222-5RT223;6RT105). Because IR detection remains unaffected by age, the algorithm performed a fine-tune adjustment in the EC value (6RT108-6RT109). Depending on the IR result, there could be an adjustment, but only up to 25% of the difference between the IR and EC results (6RT108-6RT109;6RT126;10RT243). In other words, if during the first control test, the EC reading was out-of-target with the IR reading, the EC could be corrected up to 25% of the EC-IR difference (6RT126). This algorithm compensated for the inevitable aging which took place during the twelve-month calibration cycle (10RT233).

The aging compensation routine occurred several times in the source code (10RT233-10RT234). Except in two cases, the routine was "commented out" and in one of the remaining cases it

was disabled (10RT234). Thus, the routine occurred or was "called" only once in the code, which represented two instances, both of which involved control tests (10RT234).

Shaffer stressed that the aging compensation routine occurs only during a control test under certain special circumstances, and not during the analysis of a subject's breath (10RT232). Thus, he believed the Alcotest used two independent technologies to analyze breath samples (10RT232-10RT233).

In response to the court's questions, Shaffer explained the process this way:

> THE COURT: This discussion, I read your earlier testimony about the circumstance where the EC may be borrowed by the IR under certain conditions and influence the IR reading. You heard Mr. Workman describe that.
>
> Do you have any comment on that further than what I've read? You remember what you said —
>
> THE WITNESS [Mr. Shaffer]: I do. Not to the word, but certainly the concept.
>
> I heard prior testimony from other witnesses that misstated the circumstances in which this occurs. The truth is that this occurs only during a control test, and even at that time it only occurs under certain circumstances. It never occurs during the analysis of a subject's breath sample. There are two independent technologies analyzing that sample at any time that we're collecting a breath sample where the instrument says please blow.

> THE COURT: What is the point of this borrowing from the IR — I mean EC value?
>
> THE WITNESS: The main thing that I want to clarify is that this is an aging compensation routine. The — some electronics or some radios even have a macro or a big tuning adjust knob and there's a fine adjust knob. Think of it in terms of this. The macro, the big adjustment, is being performed by the fact that we are integrating that area underneath the EC curve. That takes care of the aging compensation almost in its entirety. There is this algorithm in place to account for the fine tuning, the adjustment, that is required to compensate for the aging that does occur in between the 12-month calibration cycle.
>
> [10RT232-9 to 10RT233-15.]

When asked to explain anomalistic discrepancies between an AIR and a new solution report from Longport, Shaffer refused to speculate on the cause without more information such as the data log from the instrument (6RT117-6RT118;AB-1;AB-2). In that case, the solution control lot was left blank on the AIR but not on the solution report; the expiration data also was left blank and the reported bottle number was zero (6RT118). Shaffer said the problem could be related to the software or hardware (6RT119). In any event, this court specifically finds that such incomplete AIRs should <u>never</u> be used for evidentiary purposes.

Finally, Shaffer testified that a subject should not be able to suck air into his lungs from the breath hose if the hardware worked as intended and the flapper valve was sealed properly (10RT245-10RT246). But even in this unlikely scenario — sucking air out through the instrument — the reading would be a nonincriminating .000 in this event, as in D-129 (the three AIRs from Middlesex towns — Milltown, South River, and East Brunswick).

Shaffer recommended that New Jersey's next firmware version consider: (1) updating for current daylight savings time; (2) allowing a full 120-second delay between the collection of two subject breath samples; (3) forcing the instrument to reset upon encountering an illegal opcode trap; (4) correcting the buffer overflow defect; and (5) tightening the tolerance between breath tests by half (6RT116-6RT117;6RT123-6RT124).

This court was most impressed by Shaffer's candor, cooperation, careful explanations, and dignified demeanor. We found his testimony completely reliable and forthright.

IV. FINDINGS AND CONCLUSIONS OF LAW

1. The Beginning Of The End

Our charge in this limited remand was to determine whether software in the Alcotest "reliably analyzes, records and reports alcohol breath test results" (Order at 2). That order requested

us to advise the Court of the "effect, if any" of the expert opinion rendered on the "findings and conclusions contained" in our original February 13, 2007 report (Order at 4).

We now conclude that the proofs presented at the original hearing and at the remand hearing combine to satisfy this court that the Alcotest is scientifically reliable, both as to software and hardware, in reporting alcohol breath testing results for evidentiary purposes. We make this finding by the clear and convincing evidence burden of proof placed on the State.

The proofs at the limited remand hearing on the software and the source code aspect did not change our opinion on reliability and trustworthiness of the instrument but reinforced our initial view. We are also so convinced based on the assumption that the recommendations we made in our original report and in this report are followed in the future to ensure the continuing and possibly improved accuracy of breath test results (see Initial Findings and Conclusions).

We are firmly convinced that the Alcotest is much more reliable than the prior state-of-the-art breath testing instrument, the breathalyzer, which has been used in the past in New Jersey, and is still used in four counties. The Alcotest essentially functions independently of operator influence,

unlike the breathalyzer, which is very dependent on the operator and produces no objective and permanent record of test results. The Alcotest is also much more precise.

Based on the testimony with respect to the source code which we heard at this twelve-day remand hearing we make these further findings and recommendations, supplementing our original thoughts. Quite obviously, developing source code in this context is a dynamic, evolutionary process, not a static undertaking. The process should be re-examined and re-evaluated periodically and neither the legal nor the forensic community should fear improvement of the accepted wisdom when necessary. We should fear stagnation; we should not create an idolatry of the status quo. And simply because a procedure can be improved, does not necessarily mean the older model was illicit or worthless.

2. The Critical Issues

We now summarize the critical issues raised at this second hearing and provide our recommendations.

A. Fuel Cell Drift

Wisniewski: He found a section in the code where he said the IR reading mathematically modified the EC reading. He said that section could be called or activated from seven different paths in the code. Thus, under certain conditions, the code

would take the results of the calculations under the EC curve and divide them by the IR average.

Workman: He agreed with Wisniewski's finding that the EC and IR sensors did not operate independently as represented by Draeger. He explained that fuel cells were very common devices which depleted over time until at some point they ceased to function. He described the depletion as a function of time and the fuel cell's use. As fuel cells drift, they did not give the same output — just as a battery flashlight becomes weaker with time.

When the Alcotest's fuel cell drifted out of tolerance, the instrument used an IR value to compute an electrochemical result. Workman thought that the adjusted EC value then was used in the ambient air blanks and succeeding breath tests. Workman believed that adjustment was made in the first and last control tests based on Wisniewski's finding that it was called from seven different places within the code. Even if the adjustment was made only in the first control test, it still would affect everything that followed it. Workman could not find any warning about fuel cell drift in Draeger's operator manual. He would fix the problem by stopping the test, shutting the machine down, and putting out a message that the fuel cell had drifted out of tolerance and required replacement.

Shaffer: Fuel cells changed overtime. Because older fuel cells tended to underreport the ethanol level, the engineers in Germany inserted into the source code an algorithm or aging compensation routine to address the drift over time. Because the IR detection remained stable and unaffected by age, the algorithm performed a fine-tune adjustment in the EC value. Depending on the IR result, there could be up to a 25% adjustment of the difference between the IR and EC results. In other words, if during the first control test, the EC reading was out of target with the IR reading, the EC could be corrected up to 25% to bring it into tolerance with the IR. This EC depletion algorithm compensated for the aging whih occurred during the twelve-month calibration cycle.

The aging compensation routine occurred several times in the source code. Except in two cases, the routine was "commented out" and in one of the remaining cases it was disabled. Thus, the routine occurred only once in the code, which represented two instances both of which involved control tests. Because the adjustment never occurred during the analysis of a subject's breath, Shaffer maintained that the Alcotest employed two independent technologies to analyze breath samples.

Recommendation: We accept Shaffer's testimony and explanation. He clearly explained this issue, which we have quoted at 77-78 supra and we fully credit his testimony in this regard. This explanation may reflect on Draeger's marketing claim that it uses two completely independent technologies. We conclude that this depletion explanation does not undermine the scientific reliability of the breath measurement. The standard of measurement is adjusted for fuel cell depletion, not for any alcohol content. We recommend that the Alcotest should be calibrated every six months rather than every twelve months and the fuel cell replaced at that time, if necessary.

B. The Buffer Overflow

Geller: The buffer overflow occurs when a program attempts to store more bytes or units of information in an allocated variable which is not large enough. Geller used the analogy of trying to park a full-sized Cadillac in a compact-car parking space. After Fortify SCA warned of a potential overflow, Geller opened the code to the particular file and function, and saw a situation in the source code where six bytes were stored into a space allocated to hold only four. The two overhanging values then were used by the next two declared variables and thus were overwritten.

Geller explained that the buffer overflow affected one small part of the AIR, and not the way in which the Alcotest 7110 made any of its breath test calculations. Specifically, it arose only in these limited circumstance: (1) the results of the first two breath tests were out of tolerance; (2) a third breath test was taken; (3) the result of the third breath test fell between the values of the first two and was centered enough to be within tolerance of both of them; and (4) the ER result from the second breath test was the lowest measured value. In such cases, the AIR would not report the lowest breath test result. The overflow problem easily could be corrected with one keystroke by replacing the number "four" with a "six" at this array in the code. Geller also correctly testified that Base One's report did not discover and describe the buffer overflow problem.

Dee: While Dee recognized the potential significance of a buffer overflow, he believed it was a limited vulnerability. He described the problem as a situation where six values must be put into four boxes. Given the closed nature of the application, Dee explained that the situation in which the error occurred — three tests with similar values — was uncommon. For that reason, he surmised that the problem never actually arose in field testing and that it took a program tool specialized in

exposing vulnerabilities to raise it as an issue. That program tool, Fortify SCA, checked the source code for approximately 150 different kinds of exposure and code issues. Dee said the error could be easily fixed and recommended that it be corrected in a future release.

Shaffer: He inadvertently introduced the buffer overflow when he implemented the post-Foley changes requested by New Jersey. Specifically, he added a section of code without changing the initialization of the variable to allow it to accommodate six instead of four values. Thus, the code allocated four spaces for data when it really needed six. The sorting routine that created the buffer overflow occurred only in New Jersey, not in other jurisdictions.

The buffer overflow occurred only in limited situations where (1) breath tests one and two were not in tolerance with each other, (2) a third test was required, and (3) the EC result of the second breath test was the lowest of the six values. In these cases, the code only allowed the instrument to copy four values within the local variable causing the other two values to be overridden. By overwriting a .32 for the EC value from the second test, the instrument was unable to recognize that EC value as the lowest as it went through the sorting routine and instead, reported the second-lowest value for the breath test

result. The error did not affect the six alcohol results from the three breath tests printed on the AIR, which were never overwritten.

To correct the buffer overflow, Shaffer simply would change the number four to six at the appropriate place in the code and then run the code through the compiler. For pending cases, he did not think it was necessary to prohibit the use of the AIR in all third-test cases. Instead, Shaffer crafted a series of instructions to determine if the buffer overflow had an effect and to find the true reported breath test result. The instructions included twelve discrete mathematical operations involving addition, subtraction, multiplication and division, and several other steps. Shaffer explained that these calculations were necessary to determine if breath test three was within tolerance of breath tests one and two, and that it would be incorrect to just select the lowest of the six unaffected test results on the AIR. Shaffer recommended that New Jersey correct the buffer overflow defect.

Recommendation: As to pending cases, either prohibit the use of the BAC evidence in all third test cases or use Shaffer's formula, which the State agrees is appropriate to correct the overflow error. Because the buffer overflow is a real error in the source code, this must be corrected.

C. Weighted Averages

Workman: He believed the most significant problem uncovered by Lint was the Alcotest's averaging routine. Instead of computing a simple arithmetic average by adding a set of numbers and dividing by the total number in the distribution, he said a weighted average takes into account the number of times each value is present. Draeger claimed to use a weighted average when the Alcotest processed the IR measurements of the alcohol content in human breath. It actually averaged the final measurement on the continuum with the sum of the three previous measurements. By minimizing the earlier values and giving half the weight to the final value, Workman said the formula was scientifically unreliable as an "average."

While acknowledging the final point was a valid reading, Workman maintained it should have no more weight than the three previous values. He recognized, however, other instances where later values were given greater weight because they were more important, referring to the Bayesian probability formula used to predict the future based on past events.

Shaffer: He defended the use of weighted averages, stating it was absolutely appropriate to assign the greatest weight to the most recent value for the purpose of making a very accurate breath test measurement. He explained that individual samples

of breath from the IR measurement were taken every 8.192 milliseconds but that the weighted average routine only considered the points derived from them every .25 seconds. By relying on samples taken at .25-second intervals, the weighted average was really less than the value of the last reading.

Recommendation: None. We accept Shafer's tstimony and use of the weighted average which accurately and fairly measures blood alcohol content in the subject.

D. Lack of Standards

Wisniewski: Wisniewski said that standards or development methodologies produce the most error-free and reliable software. They also made software easier to maintain and produce more robust overall systems. They saved money in the long run. He recommended that Draeger adopt one of five he listed in his report or develop its own. However, he conceded on cross that the present code followed a known, function-oriented methodology which he said applied to the software only.

Workman: It is possible to achieve 99.98% reliability by applying standards to software development. He did not know if any breath testing instruments on the market had achieved that level of reliability. Use of standards would dictate the complexity of the modules, discourage the use of global variables, and establish testing processes and procedures.

Geller: He did not know if there were any industry standards which governed source code review. In Geller's opinion, quality software could be developed without standards and conversely, software could meet standards but still be of questionable quality. He was not personally familiar with any of the standards cited in Base One's report nor did he know if Draeger applied any standards to the Alcotest source code.

Dee: He did not agree with Base One's criticism of the Alcotest's lack of standards. In his opinion, such standards usually referred to the design and documentation of the code, and rarely addressed the code's performance. He was unaware of any standard against which the United States evaluated software. Moreover, he objected to Base One's reference to standards without stating which specific provisions were violated. He said it was possible to fully test the source code given the singular or specialized function of this application.

Shaffer: He was unaware of any single industry standard for software development. He referred to "industry standards" as collections of techniques and common-sense wisdom which had proven effective over time. Shaffer did not agree with Base One's assertion that the failure to use industry coding standards prevented the testing of critical paths in the Alcotest's software including 3200 lines of code designed to

make decisions. Because the Alcotest in the United States was highly configured to meet the requirements of specific applications, all of the 3200 lines of decision code — as calculated by Base One — were not relevant. Shaffer also said there were many unused or uncalled modules or sections of code by design. Shaffer did say standard style would be helpful but was not necessary.

Recommendation: None. The testimony of Geller, Dee and Shaffer discussed this topic persuasively and we see no need to recommend any particular style or standard.

E. Cyclomatic Complexity

Geller: He relied on the cyclomatic complexity metric developed by Thomas McCabe in 1976 to measure the number of potential paths through the code. Because high complexity increases the risk of inherent defects, coding guidelines recommend keeping the cyclomatic complexity of functions under ten, and or even seven. The SysTest report identified more than eighty-one modules in excess of ten and three in excess of a hundred. While the report recommended restructuring the code to make it less complex, Geller said the complexity indices did not influence the instrument's accuracy. Nor did excessive complexity cause failures in the interfaces between software and hardware. However, the higher complexity made the code more

difficult to understand and maintain, placing an increased burden on the programmers who worked with the software.

Dee: He did not believe that cyclomatic complexity affected the instrument's performance. To the contrary, he believed that the presence of large numbers of "decision trees" in one module was better than breaking them up into additional modules in terms of system performance. He defined decision trees as boxes containing expressions of code out of which branched other boxes, with a calling program or "main module" directing which box to go to depending on what the code said to do. Thus, Dee considered cyclomatic complexity as a stylistic issue which implicated a tradeoff between performance and ease of maintenance.

Workman: He said the code was too complex and could not be demonstrated as reliable.

Recommendation: None, because this goes to style and not the accuracy of the Alcotest. We accept Geller's and Dee's testimony as persuasive that the Alcotest performs accurately at this level of complexity.

F. Design and Style

1. Older Style

Wisniewski: Although the code was created between 1993 and 1997, it was written in a style reminiscent of the 1970s and

1980s. Wisniewski did not consider that a criticism, and acknowledged that Draeger had done a "great job" of adhering to the older style. He further said that it was not necessary to always adopt the newest technology, and that many people preferred to stay with the familiar.

Geller: The source code was written by more than one programmer and evolved over numerous transitions. It contained comments written in German and English, which comments he explained served as memory joggers for the programmer who wrote them and as advice to future programmers. Geller used the comments to find his way through the code.

Geller acknowledged the code was not written in a manner consistent with usual software design "best practices." While such practices are described in various publications, programmers often have to adapt them to fit the available resources. In cases where a code deviated from best practices, it still is safe to use but this places an added burden on programmers to understand it. Geller did not find anything in the code that looked intentionally written to skew the results. Overall, he found there was nothing particularly unusual about the Alcotest software in terms of its style and organization.

Dee: He understood that the source code was written between 1993 and 1997. Although he only scanned the code, he

saw about three or four different writing styles. He did not know if Draeger gave its programmers requirements documents with instructions on how to code.

Workman: Workman believed it was impossible to write perfect source code because (1) human beings, by nature, were fallible, (2) specifications changed over time, and (3) codes — like the Alcotest's — contained trillions of paths which made it impossible to find and fix all the errors.

2. Global Variables

Geller: The code uses global variables to store test and result data. Unlike locally declared variables designed for specific purposes within small subsets of the code, global variables are accessible from any function within the application. Also unlike local variables which pass out of existence after their use, global variables can be used throughout the duration of the program. Thus, they make information available without the resources required to pass a value from one function to another.

However, data contained in global variables is not protected and can be changed intentionally or unintentionally at any time by any function. Because they can be potentially modified from anywhere, global variables should not be used to store critical data. Given the increased risk of program error,

their use should be extremely limited. Nonetheless, Geller said there was nothing inherently wrong with global variables and their presence did not impact negatively on his conclusion that the Alcotest's software was reliable. In fact, he said that some computer languages — such as COBOL and assembly — use only global variables.

Wisniewski: He said Lint flagged defects including the use of a local variable as a global and vice versa, and the assignment of the same name to local and global variables. He described these as confusing and inconsistent, and expressed concern that they might affect some other operation in the program such as calibration. However, he acknowledged that identically named local and global variables would not confuse the compiler because the local would take precedence over the global declaration. He also admitted that there were legitimate reasons to use global variables. Time constraints prevented him from determining if a global variable actually was misused and changed the result on an AIR. Indeed, he was unable to identify anything in the code that posed a real problem that would affect a result on the AIR.

Dee: He addressed SysTest's and Base One's issue of the source code's extensive use of global variables. He agreed that a programmer must be careful to avoid overwriting a global

variable written by someone else. If an error occurred, a global variable would remain until the system was reset, re-initialized or re-powered whereas a local variable remained only for the particular calculation and then was gone. On the other hand, the use of global variables conserved memory and increased efficiency by reducing performance time. It also would be expensive to change everything to local variables. In any event, the presence of global variables did not concern Dee because the source code made extensive use of local variables and only kept what was common to all modules at the global level.

Shaffer: There are approximately 200 global variables in the Alcotest's source code and 1500 local variables. The use of globals is a tradeoff. On the downside, they placed an additional burden on programmers to exercise caution when adjusting the code to avoid unintended consequences such as overriding local variables or assigning to a new module a name already used by a global variable. On the upside, global variables were easily accessible in all modules of the program, and contained information for the use of such things as calibration which Shaffer wanted the instrument to remember after it was turned off. Their use also resulted in far less complexity and overhead, and made the code easier to design.

Moreover, to reduce their number, Shaffer would have to add more code and functions which, in turn, would create higher complexity. He also would have to touch more portions of the code than otherwise necessary. Shaffer emphasized that the decision to use global variables was made in the design process and that he did not consider their presence as a deficiency in the product.

3. Headers

Wisniewski: He was unable to determine when sections of the code were modified because there were no headers to track that information. The lack of headers made the code unreliable.

Shaffer: Headers can identify the last time a section of code was modified and the name of the programmer who made the change. He did not consider them a priority because he usually worked alone or as part of a very small engineering team. If he worked with a larger engineering team, Shaffer acknowledged that the header comments would have far more value. To determine when a module was changed, Shaffer compared previous versions of the code using other tools. For example, he used "diff programs" which highlighted the lines of code that were changed and the way they were changed, allowing him to interpolate file creation dates.

4. Core Routines

Geller: He was not concerned that the software did not contain confidentiality or copyright notices, or that core sections or routines were not "walled off." Geller explained that programmers must exercise caution, but he never had an arbitrary classification which restricted access to any part of a code.

Wisniewski: The code did not distinguish between core and customized sections. He believed that programmers should be able to touch the core software in order to learn more about it. If Draeger wanted to protect the core, however, he recommended taking the core routines out of the regular code and filing them in accessible "libraries" of object modules. The programmers would then get a reference with the library routines, and the core algorithms would be protected from change.

Shaffer: The core routines related directly to the measurement of alcohol. Because these algorithms were time-tested, field-tested, and NHTSA-tested, Shaffer avoided them. While there was no black-and-white designation of these "walled-off" sections of code, he was alerted to their presence by comments from previous developers, and discussions with Ryser and Draeger's engineering supervisor. Shaffer acknowledged it

would be easier to find the core routines if they were documented in the code, but did not believe it was necessary.

5. Comments

Wisniewski: He found several inconsistencies in the code's comments, which must be fixed. For example, he found a comment in the code stating that a conversion to "%BAC" needed to be done, but it was not done. He also found comments which said values should be averaged, but found that the source code instead performed weighted averages or successive averaging routines. Wisniewski said such comments could affect the breath test result if they were unintentionally executed; otherwise, they merely reflected a sloppy coding style. He acknowledged, however, that comments were not compiled and never reached the object code.

6. Uncalled Functions

Wisniewski: He identified fifty-one functions in the source code which were not used. Wisniewski said these uncalled functions were confusing, untidy, and unnecessary, and should be purged from the executable code.

Dee: Dee acknowledged that SysTest's report identified numerous unused or uncalled modules, which took up memory and space in the source code. While their presence could be due to the sloppiness of programmers, he believed they were typical of

software development projects where code was being used by multiple customers, decommissioned, or developed for future releases. Dee explained that it was more convenient to leave these unused functions in the code than to remove them. In any event, Dee said their presence was a question of style, and did not effect the reliability or results of the executed code.

Shaffer: There were many unused or uncalled modules or sections of code by design.

Recommendation: These design and stylistic issues are not within the scope of our recommendations. They are matters of the creator's preference and do not relate to the efficacy of breath testing in our view.

G. Catastrophic Error Detection or Illegal Opcode Trap

Wisniewski: Draeger disabled an interrupt that otherwise would detect when the microprocessor was trying to execute an illegal instruction or indefinite branching. By turning off this safeguard, the Alcotest possibly could produce unpredictable results.

Geller: He disagreed with Wisniewski's finding that the instrument could produce unreliable results because its catastrophic error detection was disabled. When this situation occurred Geller expected that the instrument would go into an

endless wait cycle, meaning it would basically cease to function, and not produce a result.

Dee: It was standard operating procedure in mature systems to disable the capabilities of the processor that detect catastrophic errors. Dee explained that these aborts were disabled and replaced with software which "captured" the errors so that a determination could be made as to whether the error was recoverable or not, or whether a more meaningful message must be written.

Shaffer: He said that when the microprocessor encountered a command or a memory location that it did not recognize — such as when an instruction in the stack or temporary memory area became corrupted — the microprocessor could lose its place in the script and jump to another section of code. When the microprocessor attempted to execute the code at the new location, it could become confused and fail to respond appropriately.

Shaffer said it was highly likely that the hardware would "lock up" or freeze so that it would be impossible to complete a breath testing sequence in these cases. Because the operator would immediately become aware of the situation, there would be no risk to the subject of a false reading.

Recommendation: We recommend that Draeger reset the microprocessor so that whenever the instrument detects an illegal opcode trap the memory will clear and start anew, as if the instrument was turned off. Based on Shaffer's discussions with his engineering colleagues in Germany, Draeger already has begun to implement this reset feature with its other customers in the United States. He did not want to touch New Jersey's program until this case is concluded.

H. Error Detection Logic

Wisniewski: He claimed the software design detects measurement errors but does not report an error message unless the errors occur thirty-two times. In the court's view, that means the instrument will report the 32nd error. Wisniewski maintained this meant an error could occur thirty-one times, but remain unreported.

Dee: He disagreed with Wisniewski's statement that the source code "ignored or suppressed" error messages unless they occurred a large number of consecutive times. In his opinion, it was normal in embedded systems to wait for the coordination of the application with the operating system or hardware. Thus, the purported error did not mean that something was wrong but rather that something was not ready. For example, there might

be slight timing differences between internal components that need adjustment.

Shaffer: Measurement errors in the Alcotest must occur thirty-two consecutive times before they are reported. He said the common practice in electrical engineering was to ignore error messages unless they occurred a large number of consecutive times. This technique has several advantages: (1) all sensors have a natural range of values and it would be surprising for them to rely on only one decision point; (2) the use of thirty-two events also allows Draeger to set tighter tolerance ranges to avoid falsely triggering errors.

Recommendation: We accept the Dee and Shaffer view that an error message is communicated effectively when stabilized, accurate and reliable. We see no need for a change.

I. Software Program Tool - Lint

Wisniewski: Wisniewski selected Lint to find defects in the source code's C language syntax, data initialization, and data management. He used a cost-free program tool derived from Lint called Splint, version 3.1.2, which raised warnings or flags, which Wisniewski called "defects" because they required action whether they were serious or harmless flaws. Wisniewski customized Splint to display all the warning or defect messages, and found about 19,500.

Lint was wordy or verbose because it tended to produce a number of defects, including repetitive examples of the same coding style. Despite its "voluminous" output, he maintained that disciplined coders would want to know about these defects and remove them to avoid any possible confusion or chance the code might not work correctly. Although the presence of these defects did not prove the software program would fail to execute, they indicated the lack of use of industry coding standards.

To insure the source code's reliability, Wisniewski said every defect should be removed. He would undertake an aggressive, ongoing campaign to find and dispose of them as part of what he called the software life cycle. Wisniewski estimated that it would take about one year to fix all the defects in the Alcotest's source code.

Defects ranged from substantive problems to variations in programmer style and organization. Some of the defects appeared numerous times, like print interrupts which were flagged about 2000 times. The Lint program did not categorize the warnings or flags, nor did it quantify any of the messages. Wisniewski did not propose corrections to any of the defects he identified in the source code nor did he check to see if they applied to functions used in New Jersey.

Geller: Lint was created as a development tool, not a review tool. It was outdated after the development of personal computers and it produced output not particularly informative to programmers.

Dee: Lint was a product of the 1970s and was not commonly used today. He believed that IDEs replaced the need for Lint by keeping a programmer within the parameters of the proper syntax during the coding process.

He was "outraged" when he reviewed Appendix C in Base One's report which purported to find "errors" in 19,000 of the 45,000 lines of code. Dee objected to Base One's attempt to quantify the errors. For example, Lint generated approximately 7657 lines of warnings based on its misunderstanding that the "U_-byte" variable was undeclared or used incorrectly. Dee later explained these lines might have values which truncated the lower digits and retained the higher values. Also, Lint ignored the quality of the errors, and improperly flagged "comments within comments." Based on the alleged errors, Dee believed that Lint did not understand some of the specific code needed for embedded system. He would not have used Lint to review the Alcotest's source code.

Recommendation: Notably, Lint did not disclose the buffer overflow error but Fortify SCA, used by Geller, did disclose

this error. The alleged hypothetical probability of irregularities raised by Lint are much too speculative and unreliable to recommend abandonment of the Alcotest on these grounds. See State v. Harvey, 151 N.J. 117, 171 (1997) (holding general acceptance of scientific evidence "does not require complete agreement over the accuracy of the test or the exclusion of the possibility of error").

We accept the testimony of Geller, Dee and Shaffer that the source code in these respects is reliable and reject Wisniewski and Workman's claims as too speculative.

J. Source Code Writing and Review

Although perhaps not critical, Shaffer's following recommendations should be considered as helpful in improving the product: (1) it is easier to find core routines if they are documented in the source code, but not necessary; (2) a more highly organized and consistently structured presentation would make the source code more readable and easier to sort, but would not make the code more understandable; and (3) a dedicated quality assurance person or outside expert who functioned in that role with respect to the source code review process would give Shaffer and Draeger a higher degree of certainty about the code.

V. FURTHER CONCLUSION

If any of the categories of data fields in the AIR are incomplete in any respect, e.g., missing calibration data, no part of the AIR can be used by the State for purposes of finding guilt. A BAC finding of .08 or above in such circumstance may not be admitted into evidence.

Foundational materials should be provided in all contested cases, not just in pro se or unrepresented cases as per our initial opinion. With reference to Addendum A in our initial opinion, and in the public interest, the State Bar, through its counsel Jeffrey E. Gold of Cherry Hill, is entitled to written notice of any proposed software revisions.

*****

We again express our gratitude for the very valuable work in this matter by our Appellate Division Staff Attorney Olga Chesler, Esquire, and for her excellent contribution to completing this difficult task both throughout the twelve-day hearing and the supplemental opinion preparation process. Many thanks, Ms. Chesler.

APPENDIX A — TRANSCRIPTS

1RT — transcript of September 17, 2007
2RT — transcript of September 18, 2007
3RT — transcript of September 19, 2007
4RT — transcript of September 20, 2007
5RT — transcript of September 24, 2007
6RT — transcript of September 25, 2007
7RT — transcript of September 26, 2007
8RT — transcript of October 9, 2007
9RT — transcript of October 10, 2007
10RT — transcript of October 11, 2007
11RT — transcript of October 23, 2007
12RT — transcript of October 24, 2007