SUMMARY OF THE SOFTWARE HOUSE FINDINGS
FOR THE
SOURCE CODE OF THE
DRAEGER ALCOTEST 7110 MKIII-C

After two years of attempting to get the computer based source code for the Alcotest 7110 MKIII-C, defense counsel in *State v. Chun* were successful in obtaining the code, and had it analyzed by Base One Technologies, Inc.

By making itself a party to the litigation after the oral arguments in April, Draeger subjected itself to the Supreme Court's directive that Draeger ultimately provide the source code to the defendants' software analysis house, Base One.

Despite Draeger's protestations that the code was proprietary, Base One found that the code consists mostly of general algorithms arranged in a manner to implement the breath testing sequence. "That is, the code is not really unique or proprietary."

In a report released August 28, 2007, Base One determined:

> As a matter of public safety, the Alcotest should be suspended from use until the software has been reviewed against an acceptable set of software development standards, and recoded and tested if necessary. An incorrect breath test could lead to accidents and possible loss of life, because the device might not detect a person who is under the influence, and that person would be allowed to drive. The possibility also exists that a person not under the influence could be wrongly accused and/or convicted.

Draeger reviewed the code, as well, through its software house, SysTest Labs, which agreed with Base One, that the patchwork code that makes up the 7110 is not written well, nor is it written to any defined coding standard. SysTest said, "The Alcotest NJ3.11 source code appears to have evolved over numerous transitions and versioning, which is responsible for cyclomatic complexity."

The best thing SysTest said about the machine was, "The translation from German to English of the comments within the major components shows the logical intent of the programmers to produce reliable and valid test results. SysTest was unable to find any evidence of any intention to mis-direct or re-direct the test results or report anything other than valid results."

SysTest only looked for "mal-ware", not for functioning of the code.

Base One, however, did an extensive evaluation, finding 19,400 potential errors in the code.

Among its findings are:

1. **The Alcotest Software Would Not Pass U.S. Industry Standards for Software**

Development and Testing: The program presented shows ample evidence of incomplete design, incomplete verification of design, and incomplete "white box" and "black box" testing. Therefore the software has to be considered unreliable and untested, and in several cases it does not meet stated requirements. The planning and documentation of the design is haphazard. Sections of the original code and modified code show evidence of using an experimental approach to coding, or use what is best described as the "trial and error" method. Several sections are marked as "temporary, for now". Other sections were added to existing modules or inserted in a code stream, leading to a patchwork design and coding style.

The software development life-cycle concept is governed by one of the nationally and internationally recognized development standards to prevent defects from entering the software during the design process, and to find and eliminate more defects as the software is coded, tested, and released to the field. This concept of software development using standards requires extensive and meticulous supporting data, and notations in source files, and a configuration management system. None of this methodology is evident in the Alcotest code. Further, the decision method of how to allocate the architecture and assignment of tasks does not match any of the software standards. This further substantiates that software development standards were not used to verify or test the software, including the ISO 9000 family of standards.

It is clear that, as submitted, the Alcotest software would not pass development standards and testing for the U.S. Government or Military. It would fail software standards for the Federal Aviation Administration (FAA) and Federal Drug Administration (FDA), as well as commercial standards used in devices for public safety. This means the Alcotest would not be considered for military applications such as analyzing breath alcohol for fighter pilots. If the FAA imposed mandatory alcohol testing for all commercial pilots, the Alcotest would be rejected based upon the FAA safety and software standards.

2. **Readings are Not Averaged Correctly**: When the software takes a series of readings, it first averages the first two readings. Then, it averages the third reading with the average just computed. Then the fourth reading is averaged with the new average, and so on. There is no comment or note detailing a reason for this calculation, which would cause the first reading to have more weight than successive readings. Nonetheless, the comments say that the values should be averaged, and they are not.

3. **Results Limited to Small, Discrete Values**: The A/D converters measuring the IR readings and the fuel cell readings can produce values between 0 and 4095. However, the software divides the final average(s) by 256, meaning the final result can only have 16 values to represent the five-volt range (or less), or, represent the range of alcohol readings possible. This is a loss of precision in the data; of a possible twelve bits of information, only four bits are used. Further, because of an attribute in the IR calculations, the result value is further divided in half. This means that only 8 values are possible for the IR detection, and this is compared against the 16 values of the fuel cell.

4. **Catastrophic Error Detection Is Disabled**: An interrupt that detects that the microprocessor is trying to execute an illegal instruction is disabled, meaning that the Alcotest software could appear to run correctly while executing wild branches or invalid code for a period

of time. Other interrupts ignored are the Computer Operating Property (a watchdog timer), and the Software Interrupt.

5. **Implemented Design Lacks Positive Feedback**: The software controls electrical lines, which switch devices on and off, such as an air pump, infrared source, etc. The design does not provide a monitoring sensory line (loop back) for the software to detect that the device state actually changed. This means that the software assumes the change in state is always correct, but it cannot verify the action.

6. **Diagnostics Adjust/Substitute Data Readings**: The diagnostic routines for the Analog to Digital (A/D) Converters will substitute arbitrary, favorable readings for the measured device if the measurement is out of range, either too high or too low. The values will be forced to a high or low limit, respectively. This error condition is suppressed unless it occurs frequently enough.

7. **Flow Measurements Adjusted/Substituted**: The software takes an airflow measurement at power-up, and presumes this value is the "zero line" or baseline measurement for subsequent calculations. No quality check or reasonableness test is done on this measurement. Subsequent calculations are compared against this baseline measurement, and the difference is the change in airflow. If the airflow is slower than the baseline, this would result in a negative flow measurement, so the software simply adjusts the negative reading to a positive value.
If the measurement of a later baseline is taken, and the measurement is declared in error by the software, the software simply uses the last "good" baseline, and continues to read flow values from a declared erroneous measurement device.

8. **Range Limits Are Substituted for Incorrect Average Measurements**: In a manner similar to the diagnostics, voltage values are read and averaged into a value. If the resulting average is a value out of range, the averaged value is changed to the low or high limit value. If the value is out of range after averaging, this should indicate a serious problem, such as a failed A/D converter.

9. **Code Does Not Detect Data Variations**

10. **Error Detection Logic**: The software design detects measurement errors, but ignores these errors unless they occur a consecutive total number of times. For example, in the airflow measuring logic, if a flow measurement is above the prescribed maximum value, it is called an error, but this error must occur 32 consecutive times for the error to be handled and displayed. This means that the error could occur 31 times, then appear within range once, then appear 31 times, etc., and never be reported. The software uses different criteria values (e.g. 10 instead of 32) for the measurements of the various Alcotest components, but the error detection logic is the same as described.

11. **Timing Problems**: The design of the code is to run in timed units of 8.192 milliseconds, by means of an interrupt signal to a handler, which then signals the main program control that it can continue to the next segment. The interrupt goes off every 8.192 ms, not 8.192

ms from my latest request for a time delay. The more often the code calls a single 8.192 ms interrupt, the more inaccurate the software timing can be, because the requests from the mainline software instructions are out of phase with the continuously operating timer interrupt routine.

12. **Defects In Three Out Of Five Lines Of Code**: A universal tool in the open-source community, called Lint, was used to analyze the source code written in C. This program uncovers a range of problems from minor to serious problems that can halt or cripple the program operation. This Lint program has been used for many years. It uncovered that there are 3 error lines for every 5 lines of source code in C.

While Draeger's counsel claims that the "The Alcotest [7110] is the single best microprocessor-driven evidential breath tester on the market", Draeger has already replaced the antiquated 7110 with a newer Windows® based version, the 9510. The computer code in the 7110 is written on an Atari®-styled chip, utilizing fifteen to twenty year old technology in 1970s coding style.

There is no doubt that the Supreme Court should declare this machine to be unreliable. If this happens, based on an agreement entered into over 4 years ago between the State and Draeger, the taxpayers of New Jersey can recover the almost $7 million spent on these machines.