INTEGRATED DIAGNOSTICS FOR SOFTWARE

CAPT JAMES P. WEBER
AIR FORCE WRIGHT LABORATORY
AVIONICS LOGISTICS BRANCH (WL/AAAF)
WRIGHT-PATTERSON AFB, OH 45433

ABSTRACT

This paper introduces the idea of diagnostics and integrated diagnostics for the diagnosis of software errors. It examines the differences between hardware diagnostics and software diagnostics. It defines what is meant by the terms Integrated and Diagnostics to make clear the distinctions between the many ways these terms are often used together. The paper defines what is meant by software diagnostics and integrated software diagnostics, and why diagnostics might be important for future embedded systems. This is done to promote a consistent usage of these terms throughout the software development community. Finally the paper concludes with some ideas about how software diagnostics and integrated software diagnostics can be accomplished to help support the software life cycle update process.

Introduction

Historically, integrated diagnostics (ID) has focused on hardware faults and has completely ignored the diagnosis of faults that exist in software. Of the two problems, diagnosis of software faults is fundamentally more complex. Hardware can break and diagnosis consists of isolating the problem. Software faults, however, are intrinsic to the delivered product. Faults, if they exist, will only appear in special situations based on the software state or timing interactions.

Integrated diagnostics in the Air Force and in the private sector is the result of vastly complex systems being developed to solve evermore difficult problems. Integrated diagnostics techniques are necessary because the systems which are being developed are huge, but the underlying technology remains the same. For hardware the basic units of development are still the transistor, resistor and capacitor. These components are being used in larger numbers than ever imagined twenty years ago. The result - a need to detect and isolate hardware faults or to tolerate faults. For software the basic unit of development is still the ubiquitous "line of code," but once again in much higher numbers than before. Human error present in coding leaves an imperfect product. The result - a need to detect and isolate faults in the software or methods to tolerate the faults.

The Air Force under the Generic Integrated Maintenance Diagnostics program (GIMADS) has written volumes on how to specify integrated diagnostics for weapon systems. Most of this, however, is aimed at the process of detecting hardware faults in large weapon systems. Before we can discuss the differences between diagnostics for hardware and software, we first need to have a common understanding of the terms. [1],[2]

Terminology

The field of Integrated diagnostics is one in which the definition of what is meant can be misunderstood. For example, are both built-in-test (BIT) and automatic test equipment (ATE) part of integrated diagnostics? Well, the answer to this question depends upon whom you speak. For some people the idea of integrated diagnostics is equivalent to built in self test. For them BIT would be part of integrated diagnostics, but depot ATE would be a logistics function. They interpret integrated to mean that diagnostics are part of the equipment. In fact, both of these areas should be considered part of integrated diagnostics. To better understand this we need to look at the meaning of

U.S. Government work not protected by U.S. copyright.
the words. It is simple, but let us start with the
dictionary definitions for the words "Integrated,"
"Diagnostics," "Internal" and "External." These
words as a group and used together will better define
the intricacies associated with the field of Integrated
Diagnostics.

Diagnostics: The art or science of analyzing the cause or
nature of a problem or situation.
Integrated: Organized or structured so that constituent units
function cooperatively.
Internal: Existing, occurring, or found within the limits or
scope of something; intrinsic.
External: Situated outside of something; acting or coming from
without.

After defining these words you can see that
integrated diagnostics is the cooperative functioning
of different units to analyze the cause or nature of
a problem. By its very nature integrated diagnostics
is a process making all diagnostics testing techniques
work together. The power of this concept lies in
perfecting the concept of integration. Integration
will be complete when any question that can be
asked about system performance or a system fault
will have a diagnostic technique which will allow the
question to be answered or the fault to be isolated.

Diagnostics can take on one of two forms: internal
diagnostics or external diagnostics. Figure 1 shows the breakdown of diagnostics
between the two fields of internal and external
diagnostics. Each part of integrated diagnostics must
do diagnostics of hardware and software.

![Figure 1: Breakdown of Diagnostics](image)

Software Diagnostics

Software diagnostics, however, is slightly
different from hardware diagnostics. As stated in
the introduction, software faults are intrinsic to the

Hardware Diagnostics

Hardware diagnostics has been going on for many
years, but the readers needs to know what the goal
of hardware diagnostics is so that they can see the
subtle difference between hardware and software
diagnostics. The goal of hardware diagnostics is to
isolate all known faults in a system, and can best be
summarized by the definition of integrated
diagnostics given in MIL-STD-1814:

"A structured process that maximizes the
effectiveness of diagnostics by integrating
pertinent elements, such as testability,
automatic and manual testing, training,
maintenance aiding and technical information,
as a means for providing a cost effective
capability to detect and isolate unambiguously
all faults known or expected to occur in
weapon systems and equipment in order to

Notice that the emphasis is on finding the faults that
are known or expected. In general, there is a set of
tests and procedures that allows detection and
isolation of hardware faults. These tests may be
completed on a work bench with a large rack of
Automated Test Equipment. Tests can be written in
manuals and given to field maintenance personnel.
They can also be written in software, embedded in
the system and the results recorded for later
analysis. The goal of integrated diagnostics for
hardware is to integrate the results of these tests to
quickly isolate faults.

Software Diagnostics

Software diagnostics, however, is slightly
different from hardware diagnostics. As stated in
the introduction, software faults are intrinsic to the
The goal of software diagnostics is to isolate and correct the software fault. This is different because once the fault is corrected it should be fixed for good. Of course other faults can be introduced by the correction of a software fault, but that is a different discussion. The resulting difference:

**Hardware diagnostics** are aimed at finding faults that have occurred before or will likely reoccur in the future.

**Software diagnostics** are aimed at finding faults that have never been seen before.

This results in different goals for the diagnostics techniques. For hardware you would follow a predetermined path along a fault tree. This test strategy could be determined by experience, with the aid of an expert system or with conventional ATE equipment. With software, the techniques must be different. For software you need tools that can answer questions for the software maintainer. In other words diagnostics for software must answer the questions: "What if?", "What happen when ...?" and "If I change ... what happens?" So software diagnostics is similar to software debugging and testing. While testing is usually done to make sure the software works correctly, diagnostics is testing to see what caused software to fail. This process of asking questions is one of the reasons that software maintenance is very expensive.

**The Need for Software Diagnostics**

The expense of software systems is one of the reasons we need integrated diagnostics for software. Figure 2 shows that 17% of the tasks associated with software maintenance are for correcting of faults. Another 10% of the tasks being accomplished to improve efficiency, and 27% being done to add new capabilities. All of these tasks need diagnostics. This means that over 54% of the tasks associated with software support require a large and well integrated set of diagnostic tools and techniques. [4],[5]

Another need for integrated software diagnostics can be seen very graphically in Figure 3. Normally people who refer to a chart of this type say that we must improve the software development process so we can find and eliminate software faults very early in the software development cycle. From the figure that is very obvious and this paper should not detract from the critical nature of what they are saying. The other factor that should be obvious from this chart is that we need to decrease the cost of fixing errors found late in the development and test process. While we can eliminate the majority of the software faults by better use of engineering discipline and through the use of well-integrated development environments, the fact remains that we will still have latent software faults during testing and deployment. This leads to the conclusion that the process by which we detect, isolate and correct
those faults should receive the same attention that is occurring to improve software development environments. [6]

**Internal Software Diagnostics**

Many of the problems associated with large embedded systems are directly related to the lack of knowledge we have with respect to real time performance of embedded software. Needed is a method for software developers to check performance and accuracy of software when it is running in its real environment. Current practice is to use simulation facilities to create an environment in which we can test embedded software while monitoring performance on specialized equipment. While this is very effective for most problems and faults, it will not catch the problems which only happen in a real environment. Since these faults can not be duplicated with simulation facilities, locating the fault is practically impossible.

The two types of faults that are most likely to exist are logic faults and temporal interaction faults. Logic faults are those in which combinations of external or internal conditions terminate in unanticipated and incorrect results (i.e. errors). These typically exist because testing cannot be complete over every possible executable path. These are logic faults and algorithm changes are required to fix the problem(s). Temporal based faults of this kind are very difficult to isolate. Even with correct algorithms being executed, the timing interactions between data and executing code segments can produce errors. Both of these fault conditions can exist and may only be seen under extreme conditions in real use.

A possible solution to this problem is the development of real time embedded monitor software. This software would become part of the Operational Flight Program in an avionics system. In the same way that we obtain performance and diagnostics data on hardware, this additional software (and hardware if needed to make the system perform as expected) would provide diagnostics and performance measurements on the embedded software. A unique aspect of this concept is that communication does not have to simply be a report mechanism. I envision that the aircraft or other embedded system could be uploaded with specific monitor request data. In this way the software maintainer would be able to get feedback on various aspects of system performance.[3]

There are many types of functions that this diagnostics system could perform. Below I will discuss some of the possible functions and some of the ways in which they might prove useful. This is not meant to be a complete list. Part of the reason for this paper is to allow others to add to the ways in which we can instrument and diagnose software faults.

**Timing Analysis:** The software monitor could keep track (upon request) of the timing of any of the procedures or tasks (Ada terminology) used in the embedded software. It could keep track of minimum and maximum execution times, or could be told to trigger a trouble report if the execution of a procedure exceeded a certain threshold. Other state information could be collected and reported at that time.

**Exception Recording:** Exception handlers are available as part of the Ada run time system. These internal checks on data integrity and system performance should be monitored. The software monitor should record all exceptions that occur during program execution and report them back to the software developer.

**Execution Buffer:** Pertinent data about the last few minutes of execution could be continuously stored in a buffer. If the pilot (operator) detects a fault, he pushes a button and that state information is recorded. This information, in conjunction with the pilot's report, would greatly increase the possibility of finding the fault.

**State Monitor:** The monitor software could be set, via an integrated network, a set of condition statements. If the software receives a certain combination of data or executes a special combination of procedures, then the monitor will record the state information of the system for later analysis.

**Fault Injection:** When used in laboratory conditions this software monitor could be used to inject faulty data. The system response to faulty data could then be monitored. In this way the
software maintainer could ensure proper execution of the code using nominal and off-nominal data.

**Fault Tolerance:** As fault tolerance routines are incorporated into embedded software, a serious question arises. Since the fault tolerant software routines mask faults that do exist, the software maintainer will never become aware of the fault. On the surface this may seem acceptable since no faulty data is ever used. On the other hand, the fault in the algorithm that is caught by the fault tolerant algorithm may corrupt data in areas not caught by the fault tolerant algorithm. The software monitor could collect fault information from all fault tolerant algorithms when and if a fault occurs. This information would be recorded and returned to the software maintainer. Since fault tolerant software may also catch hardware fault information, this information could be fed back into the hardware maintenance system. [7],[8]

In addition to the recording of fault tolerant software performance, this monitor could act as a testbed for fault tolerance techniques. The monitoring software could be told to intercept the input or output of a function and then to check it for validity. Data values could also be checked to ensure that they stay within physical limits. For example, elevation can be checked to make sure it doesn't go below the elevation of the Dead Sea. The aircraft role rate could be checked to ensure that it does not change by more than a certain amount over a specified period of time. These types of validity checks are the same type that might be found in fault detection algorithms; and this software could perform those tasks to see if a fault exists in the software. If it does, it can be fixed and it may not be necessary to use fault tolerant techniques to obtain the needed reliability. [9]

**Built-In Support Function (BISF):** Another laboratory project related to this concept is the idea of a Built-In Support Function. The concept proposed by Mr Duane O. Hague Jr. is the creation of a special group of instructions for the microprocessor. [10] They would appear to be no-op instructions to the rest of the code, but would send information out of the processor on a specialized output port. This port could be buffered and then monitored by either exterior probes or by the internal diagnostics software. With this information, output about code during execution could be obtained, analyzed and stored as needed to locate a fault. With the use of processors with on-board cache and other performance enhancement technologies, it is no longer possible to see what is happening in the processor by monitoring the system bus. As a result a BISF may become very important to system debugging in the near future. An interesting side note to this technique would be the ability of the diagnostics software to monitor itself.

Internal diagnostics can also be applied to laboratory conditions where the embedded computer is not in an operational environment. This is currently done under the titles of: verification and validation; testing; integration test stands; dynamic test stands; symbolic debuggers and many more.

**External Software Diagnostics**

The area of external software diagnostics has already been explored by various techniques. My point in mentioning it is that it is not normally called external software diagnostics. In addition, external and internal diagnostics need to be used together to develop a complete integrated diagnostics system for software.

A technology that has not been fully developed is source code analyzers. There are many possible ways that tools could be developed in order to determine if a major fault exists. Tools of this type can take on two forms. The first looks strictly at the source code to check for typical programming faults. The second is to analyze embedded comments to verify that the programmer was consistent in what he did and how he documented what was coded. [11]

At Wright Laboratory we have been examining the concepts involved in developing a complete support system for software development.

**Advanced Multi-Purpose Support Environment (AMPSE):** This Wright Laboratory development effort is aimed at creating a well-integrated development and testing environment. Currently it is largely a simulation test environment, but future work will provide a integrated environment where design, development, testing and diagnostics are fully integrated.
**Processor Monitor and Control (PMAC):** The processor monitor and control project is aimed at monitoring software execution in real time. This requires special hardware to be placed in the embedded computer that can monitor all bus functions and data transactions that the computer. As part of the AMPSE, it provides the basis for real-time monitoring and analysis of software in a laboratory environment.

**Integrated Software Diagnostics Process**

Integrating software diagnosis into the development processes will be based on understanding what information can be obtained. The new information obtained by using these techniques can be used to improve software quality and reliability. The integration process will require the use of technologies not used in the past for the purpose of diagnostics.

Using network technology to integrate the user community, the field maintenance personnel, and depot maintenance personnel is a first step towards integrated diagnostics for software. This will allow two-way communications between all levels of the maintenance system. This type of network must permit human to human and human to computer interaction in a way that is simple and straightforward. The network must track and control testing and diagnostics taking place in various locations and keep a central data base of the results. If there are embedded software diagnostics being executed during flight test, then the result must be immediately available to the software maintainer when the flight is finished. In addition the pilot and the software developer should be able to talk so that flight tests can be used productively. It should be remembered that this type of testing is for the hard-to-find problems, and should not be used for every fault discovered in the software. The key is real time interaction between people and machines.[12]

At the Avionics Software Support Activity (ASSA), integration must take place to ensure that testing of embedded code can take place quickly and without delays. This may require better compiler technology, better configuration control, increased computer resources, better networking technology and workstation-based development environments. This entire system should then be integrated into the network described above.

As new development environments and testing techniques are developed, it is critical that they be built to work within the environment that has been developed. This is true for each particular development environment. New development environments should use the latest technology.

**Conclusions**

The development of integrated diagnostics must be done in conjunction with other traditional testing and development methodologies. In fact by definition integrated diagnostics is the incorporation of all the other traditional functions plus new techniques into a unified whole. This means that more than anything else people need to work together. In essence it requires a Total Quality Management (TQM) approach to the development, testing and maintenance of software.

Greg Boone has a complete presentation of the applicability of TQM to software development. One of his important points is that we need to keep our focus on the following principles:

- We need to be process focused not product focused.
- We need a quality focus not a productivity focus.
- We need to stress high employee involvement.

The goal is to successfully develop diagnostic systems that are integrated into hardware and software. This will require the use of software to diagnose hardware faults and the use of hardware to diagnose software faults and all other combinations of hardware and software. This will mean that all of our test equipment, BIT, maintenance personnel, hardware and software depot personnel, and training material must work together. Developing such systems will fundamentally change the way we view the development process. When designing and implementing a system in which the development, test and diagnosis capabilities are more important than the product under development, we will know...
that we have arrived. That will be an integrated
diagnostic and integrated development environment
which can quickly and inexpensively make changes
to a system. That support environment defines how
well you can use your research, development and
maintenance dollars. Since change is the name of
the game in the development of any and every
system built by man, it only makes sense that we
build a development system that can handle change
in the system it is built to support.

At the heart of the integrated diagnostics concept
is the idea of the unanswered question. The true test
of an integrated diagnostics system is to see how fast
and how cheaply it can answer the next question.
Faults in software are very difficult to locate.
Integrated diagnostics for software are needed to
support weapons systems and industry systems of the
future.

References
Diagnostics, 30 April 1991.
[3] Nichols, David L., Vicen, Paul M., Marcu, Bruce R.,
"Software - Missing Piece in the Integrated Diagnostics
Puzzle" IEEE/AESS 1989 National Aerospace Electronics
Management Ernest R. Keucher, Editor. Air Force
Institute of Technology School of Systems and Logistics,
Department of System Acquisition Management, WPAFB
Operational Flight Programs" IEEE/AESS 1989 National
Aerospace Electronics Conference Proceeding, May
Software Failure Characterization," IEEE AES Magazine,
November 1990, p. 38-44.
[7] Leveson, Nancy G., Cha, Stephen S., Knight, John C.,
Shimeall, Timothy J. "The use of Self Checks and Voting
in Software Error Detection: An Empirical Study, O"IEEE
Transactions on Software Engineering, Vol 16, No.4,
April 1990, p. 432.
Empirical Comparison of Software Fault Tolerance and
Fault Elimination,"IEEE Transactions on Software
[9] Technical Meeting #3 for contract F33615-89-C-1138
presentation materials, "Avionics Fault Tolerant
1750A Computer", In-house technical paper, WL/AAAF-
3, WPAFB OH.
Programming Errors," IEEE Transactions on Software
or Hindrance?" IEEE/AESS 1989 National Aerospace
Electronics Conference Proceeding, May 1989, Dayton
OH, p. 1316.
Development", ASQC International Software Quality
Conference Proceedings. 7-9 October 1991, Dayton, OH,
p. 212.
Howard "Tools for Integrated Diagnostics" Final
Technical Report for Rome Air Development Canter
contract number F30602-84-C-0106, RL/BRBA, Griffis
Diagnostic Software for SMARTNet-2", IEEE/AESS 1989
National Aerospace Electronics Conference Proceeding,
Diagnostics Model", Proceedings of the International
Automatic Testing Conference, Nov 1987, San Francisco
CA, p. 447-450.
Not Inspire Confidence," IEEE Transactions on Software
[18] C. R. Kime "FAULT-TOLERANT COMPUTING - Theory and
Techniques, Volume II" Chapter 8, pages 577-632,
Prentice-Hall, Edited by Dhiraj K Pradhan.