Dynamic Software Measurement

Computer Measurement Laboratory, Inc.
info@cmlab.biz

From the outside looking in, the field of computer science and software engineering is a real puzzlement. At the core of every other science and engineering discipline is the notion of measurement. Yet, the software field is virgin territory in this regard. It is pretty easy to see how we got that way. Computer science spun off from mathematics, which, in turn, spun off from Philosophy. Measurement is not a key player in either of these disciplines.

In order to measure something, there must be an instrument to do this. A ruler is an example of such an instrument. To build a ruler there must be a standard against which this ruler can be built. The National Institute of Standards and Technology maintains such standards. Let us examine their standards holding for a simple software metric such as Lines of Code, LOC. After a very frustrating search through their standards, we discover that there are no software standards at NIST. What this means is that two people conducting research on two different software systems cannot share even the most fundamental information about the systems that they are working on. It may well be that one researcher is counting LOC by enumerating <CR>s. Another may be counting only non-blank, non-commented lines. That one researcher found 1024 LOC in one system is a fact that cannot be shared with anyone else in the research literature.

The lack of software measurement standards means that science is simply not possible in "Computer Science". To fill this void, researchers in the discipline have turned to "Formal Methods". Unfortunately, while mathematical rigor is a necessary condition for science it is not a sufficient condition. The scientific method is fundamental to science and measurement is a vital component in the conduct of scientific inquiry. Instruction in the scientific method is not part of a mathematics curriculum.

What do you measure?

Enough. Different measurement tools are used for different purposes. A micrometer is best used to test manufacturing tolerances of machine parts -- a ruler to measure a piece of paper. The objective is to determine the necessary precision needed and then match the measurement tool to that requirement.

So it is with software. The attribute space of an executing software system is very large. There are quite a few different things that can be measured. The objective is to determine exactly what you will need to see and then measure for that.

There are two distinct software measurement domains. There are static software measures and there are dynamic software measures. Static software measures can be taken directly from the software source code. Dynamic software measures must be taken on the compiled code as it is executing. The focus of this discussion is on dynamic software measurement and the attribute space is very broad. We can measure, the frequency of transitions from one module to another.

We can measure the elapsed time interval between to arbitrary points in the code. We can measure the usage of execution paths in the control flow graph of a program.

How do you measure?

It is possible to measure executing software through software probes. These are instrumentation points in the code that will generate telemetric events when they are encountered. Unfortunately, software instrumentation is highly visible, very intrusive and almost unusable for real time applications.

A monitor co-processor is the very best way to measure and executing software system. This co-processing unit will be permitted to observe the flow of the instruction though the CPU(s). Thus any or all of the instructions in an executing program may be observed. The advantage of measurement using the co-processor concept is that the measurement process is totally transparent. This is important for two reasons. First, the measurement process will have no impact of the performance of the system being monitored. Second, it cannot be detected or manipulated by software executing on the monitored process.

How Might Measurement be Applied?

There are many different domains in software development and deployment that are ready candidates for dynamic measurement. The measurement process should begin very early in the software lifecycle and should remain an integral part of any software system throughout its life. The monitor co-processor is, in fact, the telemetry engine for a software process control system. This monitor co-processor measures the activity of a software system while it is running. The central issue here is that once we understand the notion that a program exhibits activity that can be measured, we can begin to assert control on the program execution. A fundamental notion of process control is that there is a standard model against which the operation of a system is continually compared.

Testing

When a software system is being tested, it is, de facto, being certified for a range of different functionalities. There are really two distinct objectives in this testing process. First, is to measure the progress of the testing to understand exactly which functionalities have been exercised appropriately and which have not. Second, the range of test activity should be captured to form the standard model or certificate that will describe the certified behavior of the system. In the testing role, then, the monitor system will provide status reporting on the evolution of the test process. It will also build a certificate for tested activity.

Reliability

Software systems do not fail or wear out as a function of time. They do, however, have a tendency to fail when a user of the system expresses new and uncertified behavior. Again, from the software test activity, a record of that activity is embodied in the software certificate. With the continuous monitoring of a software system, it is possible to observe, in real time, when the

software is moving into new and uncertified activity domains. In this role, the monitor system can provide a real time reliability assessment of the monitored software in terms of exactly what the software is doing right now.

Availability

It is one thing to monitor the reliability of a system in real time. It is quite another to take action based on that information. If a user community is beginning to use a piece of software in an uncertified, unreliable manner, it is possible to detect that departure with the monitor engine immediately. Then the precise nature of the departure can be identified when compared against the standard model of the execution certificate. In the high availability mode, then, the monitor engine will capture the precise departure from uncertified activity on a user's system and then transfer this departure information to the vendor. Then software vendor may then test the software in then new activity domain, recertify the software for the new activity, and perhaps hot-patch software modifications back to the operating software. The objective of the monitor in this role is to identify the departure of the software from its certified range, capture the nature of the departure and then call home to report the departure. In this manner we may fix the software before it breaks.

Security

Software security is really a corollary of the notion of software availability. In the case of reliability, software may be driven into uncertified territory in that the software was used in a manner that was not tested. In the case of a software exploit, the software is deliberately driven into a region where it will either fail or allow the attacker to compromise the system on which the software is running.

In the security role, the monitor engine will serve a slightly different role. In this case, the monitor engine will observe the departure and then take action to transfer control to an adaptive engine running on the host system. This adaptive component can then manage the attack by suspending the software, killing the software process, sandboxing the application, identifying the user and/or source of the attack, or a range of other possible actions. The second thing that the monitor engine might do is to record the circumstances that lead to the departure from the certified behavior. These data may then be turned over to a forensic system, again running on the host, for vulnerability analysis.

Survivability

It is one thing for a software system to execute a software fault (vulnerability). It is quite another for the entire system to fail because of this one execution event. In the survivability application, the monitor engine will observe the failure event and then take action on the host machine to insure that the software continues to execute. In this case, the adaptive engine running on the host machine will gain control and modify the software system that just failed, to eliminate the functionality that led to the failure event. The software may then be returned to service operating, now, at a reduced level of functionality but operating nonetheless.

Software Black Box Recorder

The software black box recorder is really an extension of the availability engine. In this guise, the monitor engine will trace the execution path through the execution certificate, writing the execution events into a circular queue. This queue will be preserved in nonvolatile memory. In the avionics realm, this memory may be retrieved and analyzed to determine the precise status of the avionics system in the event of an aircraft crash.