An Alternative Software Reliability Assessment

Herbert Hecht, SoHaR Incorporated, Culver City, California

***Abstract* – UML software development tools facilitate computer aided reliability assessment based on severity of potential failure effects and effectiveness of protection provisions. This assessment is more widely applicable than one based on failure rate.**

INTRODUCTION

The measured failure rate (or one of its first cousins) has long been the "gold standard" for software reliability[1]. In current computing environments it can be inapplicable and even misleading. Consider server software with an expanding number of clients. More users are likely to cause an increase in the failure rate though the software (and therefore its reliability) are not changed. Another example is software controlling a machine tool. The machine tool is aging, causing more exception conditions to be encountered by the program and hence more failures. The machine shop supervisor sees a higher failure rate even though the software remains the same.

Since there are problems with using failure rate as an indicator of reliability in existing software, we looked for alternatives for predicting software reliability during development and that would continue to be valid in operation. The severity of failure effects needed to be taken into account so that preventive steps could focus on avoidance of the most severe failures.

SOFTWARE FMEA

This latter requirement suggested a look at software failure modes and effects analysis (FMEA), a topic that has been under investigation for over 20 years[2]. But while FMEA for hardware is widely used, it is rarely encountered for software. An obvious reason is that hardware is generally made up of parts with well-known failure modes; there is no equivalent of this in software. Instead, software is analyzed by "functions". But these are subjective partitions and there is usually no certainty that all functions that can contribute to failure have been included.

UML-based software development tools permit us to overcome this difficulty. In the UML approach the smallest operational software construct is a *method*. If all methods of a program work correctly, the program will not fail. Conversely, if a method is faulty the program will fail under some conditions. Thus, the method is similar to a part. All methods are listed in the *class chart* of a program, and thus the FMEA will be complete if all listed methods have been analyzed. Because the class charts are computer accessible files, the generation of software FMEA can be partially automated, reducing both the labor required and the potential for errors.

The computer-aided generation of a software FMEA is shown in Figure 1. The entry in the component column is generated automatically from the listing of *methods* by the UML tool. Our program assigns the hierarchically formatted ID (first screen at the top). The second screen shows the assignment of failure modes. We provide three default failure modes for each component: crash, stop (with return of a symptom code), and output error. Other modes can be added by the analyst. The local effect (third screen) is usually assessed at the computer program component (CPC) level. If the CPC incorporates protective code the failure mode may not propagate at all or at a reduced level. The presence of protection measures is noted in the Detection Method column of the FMEA worksheet.

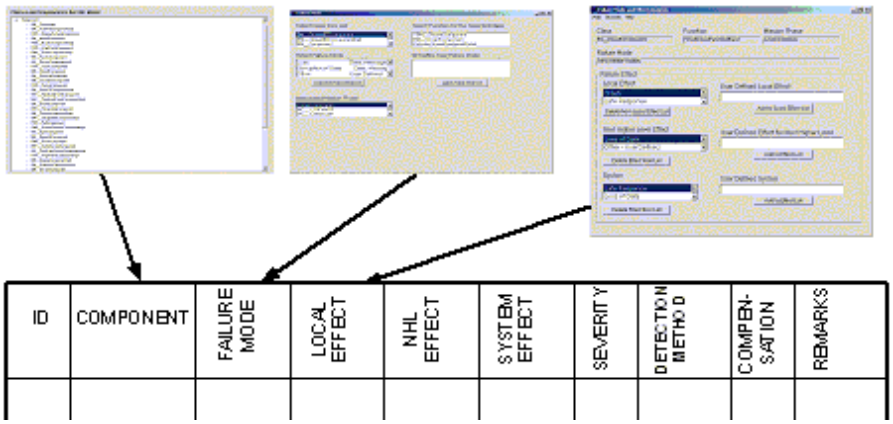

Figure 1 FMEA Worksheet

PROTECTION PROVISIONS

Examples of Detection Methods are assertions, code checks on incoming and outgoing data and sequence checks on operations. Since these are implemented as UML methods their failure modes are covered by the worksheet. Further, they can be tagged as "Detection Methods" by the analyst and when referenced will be automatically entered into the Detection Method column. Once a failure mode is detected, compensating measures can be invoked, including use of default values, repeating an operation and transferring to

an alternate routine. The compensating provisions are also implemented as UML methods and can be tagged and analyzed in the same manner as the detection. Absent these provisions the failure mode will propagate to successively higher levels, eventually causing a serious system level effect. Table 1 shows how the propagation of a crash failure is affected by detection and compensation provisions at the next higher level.

To be effective, the detection provisions should be close to the source of the failure mode. This prevents contamination of the data stream, provides useful diagnostics, and permits invocation of the most appropriate compensation. If an anomaly in the fuel flow monitoring routine is detected, the last value of fuel flow can be used as a default for at least one cycle. But if detection is relegated to a higher hierarchical level, the anomaly will affect the entire engine management, making diagnosis as well as recovery much more difficult.

Table 1. Effect of Detection and Compensation on Propagation of a Crash

Detection and Compensation	NHL Effect
None	Crash
Detection only	Stop
Detection and re-try	Delayed output
Detection and default value	Degraded output
Can call alternate method	None

The failure effects are propagated to the system level, such as the flight management system (FMS), where severity designations are associated with each failure mode. A crash of the FMS will probably cause the mission to be abandoned which is conventionally considered a severity II failure. Crash of a flight control system may jeopardize the safety of the aircraft and will be considered severity I. Failures that impair mission effectiveness (short of abandonment) are designated severity III and all others severity IV.

RELIABILITY ASSESSMENT

The reliability assessment will deal exhaustively with all failure modes that lead to severity I and II failures, and summarize the protection against severity III and IV failures. For the highest severity failure modes it is essential that detection is direct (close to the source) and that compensation is immediate and effective, preferably by access to an alternate routine or standby processor. For the lower severity failure modes detection by effect (removed from the source) can be acceptable, and compensation by default value or re-try can be used.

This approach does not provide any estimate of failure rates and thus makes software FMEA worksheets different from those used for hardware. But if the assessment described above is carried out correctly and shows no gaps in fault coverage it will demonstrate to project management that the software is reliable. Where gaps are found, the required corrective action is in most cases obvious. This assessment, tied to system effects, is appropriate for management review and may be preferred to one using failure rates. Where consistency with hardware FMEA is essential, the assessment format described here can be adopted for hardware.

The assessment has an important legacy to test: once a failure mode is covered by detection and compensation provisions, the emphasis in test can shift to testing these provisions with fewer resources allocated to testing the functional code. Because detection and compensation provisions take a limited number of forms, test case generation is simplified and the cost of test is reduced.

In addition to the computer-aided generation of FMEA worksheets for UML-based programs we are working on computer-aided generation of timed Petri nets for the exploration of timing and sequence related failure modes in real-time systems.

ACKNOWLEDGMENT

This research has been supported by the DARPA MoBIES program, headed by Dr. John Bay, and funded under contract F33615-02-C-3253 for which Ray Bortner, AFRL, is the technical monitor.

REFERENCES

[1] ANSI/AIAA Recommended Practice *Software Reliability,* R-013-1992

[2] Reifer, Donald J, "Software Failure Mode and Effects Analysis", *IEEE Transactions on Reliability*, vol.28, no.3, August 79

