6.7 Software Reliability

Software reliability, unlike many other quality factors, can be measured directed and estimated using historical and developmental data. *Software reliability* is defined in statistical terms as "the probability of failure-free operation of a computer program in a specified environment for a specified time" [MUS87]. To illustrate, program *X* is estimated to have a reliability of 0.96 over eight elapsed processing hours. In other words, if program *X* were to be executed 100 times and require a total of eight hours of elapsed processing time (execution time), it is likely to operate correctly (without failure) 96 times.

> **"The unavoidable price of reliability is simplicity."**
>
> **C.A.R. Hoare**

Whenever software reliability is discussed, a pivotal question arises: What is meant by the term *failure?* In the context of any discussion of software quality and reliability, failure is nonconformance to software requirements. Yet, even within this definition, there are gradations. Failures can be only annoying or catastrophic. One failure can

be corrected within seconds while another requires weeks or even months to correct. Complicating the issue even further, the correction of one failure may in fact result in the introduction of other errors that ultimately result in other failures.

26.7.1 Measures of Reliability and Availability

Early work in software reliability attempted to extrapolate the mathematics of hardware reliability theory (e.g., [ALV64]) to the prediction of software reliability. Most hardware-related reliability models are predicated on failure due to wear rather than failure due to design defects. In hardware, failures due to physical wear (e.g., the effects of temperature, corrosion, shock) are more likely than a design-related failure. Unfortunately, the opposite is true for software. In fact, all software failures can be traced to design or implementation problems; wear (Chapter 1) does not enter into the picture.

KEY POINT
Software reliability problems can almost always be traced to defects in design or implementation.

There has been debate over the relationship between key concepts in hardware reliability and their applicability to software (e.g., [LIT89], [ROO90]). Although an irrefutable link has yet to be established, it is worthwhile to consider a few simple concepts that apply to both system elements.

If we consider a computer-based system, a simple measure of reliability is *mean-time-between-failure* (MTBF), where

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$

The acronyms MTTF and MTTR are *mean-time-to-failure* and *mean-time-to-repair*,[6] respectively.

KEY POINT
It is important to note that MTBF and related measures are based on CPU time, not wall clock time.

Many researchers argue that MTBF is a far more useful measure than defects/KLOC or defects/FP. Stated simply, an end-user is concerned with failures, not with the total error count. Because each defect contained within a program does not have the same failure rate, the total defect count provides little indication of the reliability of a system.

In addition to a reliability measure, we must develop a measure of availability. *Software availability* is the probability that a program is operating according to requirements at a given point in time and is defined as

$$\text{Availability} = [\text{MTTF}/(\text{MTTF} + \text{MTTR})] \times 100\%$$

ADVICE
Some aspects of availability (not discussed here) have nothing to do with failure. For example, schedule downtime (for support functions) causes the software to be unavailable.

The MTBF reliability measure is equally sensitive to MTTF and MTTR. The availability measure is somewhat more sensitive to MTTR, an indirect measure of the maintainability of software.

26.7.2 Software Safety

Software safety [LEV86] is a software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail. If hazards can be identified early in the software

6 Although debugging (and related corrections) may be required as a consequence of failure, in many software will work properly after a restart with no other change.

process, software design features can be specified that will either eliminate or control potential hazards.

> "I cannot imagine any condition which would cause this ship to founder. Modern shipbuilding has gone beyond that."
> E. I. Smith, captain of the Titanic

A modeling and analysis process is conducted as part of software safety. Initially, hazards are identified and categorized by criticality and risk. For example, some of the hazards associated with a computer-based cruise control for an automobile might be:

- Causes uncontrolled acceleration that cannot be stopped.
- Does not respond to depression of brake pedal (by turning off).
- Does not engage when switch is activated.
- Slowly loses or gains speed.

Once these system-level hazards are identified, analysis techniques are used to assign severity and probability of occurrence.[7] To be effective, software must be analyzed in the context of the entire system. For example, a subtle user input error (people are system components) may be magnified by a software fault to produce control data that improperly positions a mechanical device. If a set of external environmental conditions are met (and only if they are met), the improper position of the mechanical device will cause a disastrous failure. Analysis techniques such as fault tree analysis [VES81], real-time logic [JAN86], or Petri net models [LEV87] can be used to predict the chain of events that can cause hazards and the probability that each of the events will occur to create the chain.

Once hazards are identified and analyzed, safety-related requirements can be specified for the software. That is, the specification can contain a list of undesirable events and the desired system responses to these events. The role of software in managing undesirable events is then indicated.

Although software reliability and software safety are closely related to one another, it is important to understand the subtle difference between them. Software reliability uses statistical analysis to determine the likelihood that a software failure will occur. However, the occurrence of a failure does not necessarily result in a hazard or mishap. Software safety examines the ways in which failures result in conditions that can lead to a mishap. That is, failures are not considered in a vacuum, but are evaluated in the context of an entire computer-based system and its environment. Those readers with further interest should refer to Leveson's [LEV95] book on the subject.