Shooman Ch 5.9.3

[1990, pp. 582–587].) Another case is computed in which the initial value of $p_i = (1 - 1{,}368/1{,}000{,}000) = 0.998632$ is much higher. In this case, TMR produces a reliability of 0.99999433 for an improvement in unreliability by a factor of 241. However, the same estimate of common-mode failures reduces this factor to only 1.1! Clearly, such a small improvement factor would not be worth the effort, and either the common-mode failures must be reduced or other methods of improving the software reliability should be pursued. Although this data varies from program to program, it does show the importance of common-mode failures. When one wishes to employ redundant software, clearly one must exercise all possible cautions to minimize common-mode failures. Also, it is suggested that modeling be done at the outset of the project using the best estimates of independent and common-mode failure probabilities and that this continue throughout the project based on the test results.

5.9.3 Space Shuttle Example

One of the best known examples of hardware and software reliability is the Space Shuttle Orbiter flight control system. Once in orbit, the flight control system must maintain the vehicle's altitude (rotations about 3 axes fixed in inertial space). Typically, one would use such rotations to lock onto a view of the earth below, travel along a line of sight to an object that the Space Shuttle is approaching, and so forth. The Space Shuttle uses a combination of various large and small gas jets oriented about the 3 axes to produce the necessary rotations. Orbit-change maneuvers, including the crucial reentry phase, are also carried out by the flight control system using somewhat larger orbit-maneuvering system (OMS) engines. There is much hardware redundancy in terms of sensors, various groupings of the small gas jets, and even the use of a combination of small gas jets for sustained firing should the OMS engines fail. In this section, we focus on the computer hardware and software in this system, which is shown in Fig. 5.19.

There are five identical computers in the system, denoted as Hardware A, B, C, D, and E, and two different software systems, denoted by Software A and B. Computers A–D are connected in a voting arrangement with lockout switches at the inputs to the voter as shown. Each of these computers uses the complete software system—Software A. The four computers and associated software comprise the primary avionics software system (PASS), which is a two-out-of-four system. If a failure in one computer occurs and is confirmed by subsequent analysis and by disagreement with the other three computers as well as by other tests and telemetered data to Ground Control, this computer is then disconnected by the crew from the arrangement, and the remaining system becomes a TMR system. Thus this system will sustain two failures and still be functional rather than tolerating only a single failure, as is the case with an ordinary TMR system. Because of all the monitoring and test programs available in space and on the ground, it is likely that even after two failures, if a third malfunction occurred, it would still be possible to determine and switch

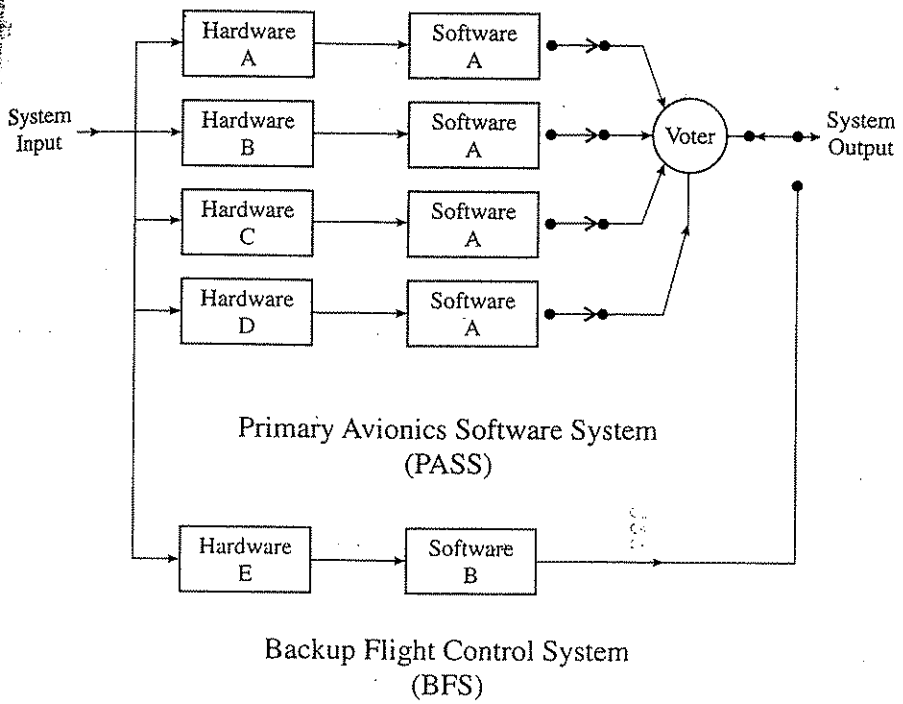

Figure 5.19 Hardware and software redundancy in the Space Shuttle's avionics control system.

to the one remaining good computer. Thus the PASS has a very high level of hardware redundancy, although it is vulnerable to common-mode software failures in Software A. To guard against this, a backup flight control system (BFS) is included with a fifth computer and independent Software B. Clearly, Hardware E also supplies additional computer redundancy. In addition to the components described, there are many replicated sensors, actuators, controls, data buses, and power supplies.

The computer self-test features detect 96% of the faults that could occur. Some of the built-in test and self-test features include the following:

- Bus time-out tests: If the computer does not perform a periodic operation on the bus, and the timer has expired, the computer is labeled as failed.
- Comparisons: Check sum is computed, and the computer is labeled as failed if there are two successive miscompares.
- Watchdog timers: Processors set a timer, and if the timer completes its count before it is reset, the computer is labeled as failed and is locked out.

To provide as much independence as possible, the two versions of the

software were developed by different organizations. The programs were both written in the HAL/S language developed by Intermetrics. The primary system was written by IBM Federal Systems Division, and the backup software was written by Rockwell and Draper Labs. Both Software A and Software B perform all the critical functions, such as ascent to orbit, descent from orbit, and reentry, but Software A also includes various noncritical functions, such as data logging, that are not included in the backup software.

In addition to the redundant features of Software A and B, great emphasis has been applied to the life-cycle management of the Space Shuttle software. Although the software for each mission is unique, many of its components are reused from previous missions. Thus, if an error is found in the software for flight number 76, all previous mission software (all of which is stored) containing the same code is repaired and retested. Also, the reason why such an error occurred is analyzed, and any possibilities for similar mechanisms to cause errors in the rest of the code for this mission and previous missions are investigated. This great care, along with other features, resulted in the Space Shuttle software team being one of the first organizations to earn the highest rating of "level 5" when it was examined by the Software Engineering Institute of Carnegie Mellon University and judged with respect to the capability maturity model (CMM) levels. The reduction in error rate for the first 11 flights indicates the progress made and is shown in Fig. 5.20. An early reliability study of ground-based Space Shuttle software appears in Shooman [1984]; the model predicted the observed software error rate on flight number 1.

The more advanced voting techniques discussed in Section 4.11 also apply to *N*-version software. For a comprehensive discussion of voting techniques, see McAllister and Vouk [1996].

5.10 ROLLBACK AND RECOVERY

5.10.1 Introduction

The term *recovery technique* includes a class of approaches that attempts to detect a software error and, in various ways, retry the computation. Suppose, for example, that the track of an aircraft on the display in an air traffic control system becomes corrupted. If the previous points on the path and the current input data are stored, then the computation of the corrupted points can be retried based on the stored values of the current input data. Assuming that no critical situation is in progress (e.g., a potential air collision), the slight delay in recomputing and filling in these points causes no harm. At the very worst, these few points may be lost, but the software replaces them by a projected flight path based on the past path data, and soon new actual points are available. This is also a highly acceptable solution. The worst outcomes that must be strenuously avoided are from those cases in which the errors terminate the track or cause the entire display to crash. Some designers would call such recovery techniques *rollback* because the com-