Part One

Reliability Basics

Chapter 1

Reliability and Availability Concepts

This chapter reviews the basics that underpin system design for reliability:

- Definitions of reliability and availability in the context of systems and services
- How faults are activated to become errors and evolve into failures
- Failure recovery and high availability
- Quantifying downtime and service availability
- Attributing responsibility for downtime and outages
- Overviews of hardware and software reliability

1.1 RELIABILITY AND AVAILABILITY

Reliability is defined by the IEEE as "*the ability of a system or component to perform its required functions under stated conditions for a specified period of time*" [IEEE610]. For example, a failure rate, such as the frequency of software crash or failure, measures reliability; mean time to first failure is a simple reliability metric.

*Availabilit*y is defined as "*the degree to which a system or component is operational and accessible when required for use*" [IEEE610]. For example, the probability that a car will start is a measure of the car's availability. A simple mathematical formula to compute service availability is:

$$Availability = \frac{Uptime}{Uptime + Downtime}$$

Design for Reliability: Information and Computer-Based Systems, by Eric Bauer

Since both the numerator and denominator of the formula are in the same units (time), the units cancel out and the value of availability becomes dimensionless. Because availability is dimensionless, it is generally expressed as a percentage, such as 99.999% or "five 9's." Table 1.1 shows the relationship between number of 9's and down-minutes per system per year.

Table 1.1 Availability as a Function of Number of 9's

Number of 9's	Availability	Annualized down-minutes per system	Practical meaning
1	90%	52596.00	Down 5 weeks per year
2	99%	5259.60	Down 4 days per year
3	99.9%	525.96	Down 9 hours per year
4	99.99%	52.60	Down 1 hour per year
5	99.999%	5.26	Down 5 minutes per year
6	99.9999%	0.53	Down 30 seconds per year
7	99.99999%	0.05	Down 3 seconds per year

Note that although many people speak of "99.999% system reliability," they often actually mean *five 9's service availability*.

Service availability is essentially based on the simple model that a system is either "up" or "down," and transitions between those states are fairly well understood; this abstraction maps well to critical service failure events, such as system crashes. However, failures occur that can cause a few isolated operations to fail without bringing the system down. Readers will be familiar with these service failures, such as when their wireless calls drop or fail to complete, or when they have to use the "reload" button on their web browser to reload a page that did not display properly. Assuming the network or server did not actually go down, the service failure will not be reflected in the service availability metric. Service reliability measures the rate of successful user interactions or transactions when the system is up, such as rate of successful transactions, successful call completions, or successful web page displays. As service reliability is often very good, it is often more convenient to measure service *un*reliability as defective transactions or interactions per million attempts. For example, 15 web page loads per million might fail, or 37 database update transactions might fail per million attempts, or 27 calls per million attempts may fail to be established with acceptable voice quality. Although the servers or network probably did not go down, some individual users experienced unacceptable service. Thus, users experience unacceptable service either when the system is down (which is measured via service *unavailability*) or when the system is up but fails to complete their service request properly (which is measured via service *unreliability*). This book addresses designing systems that deliver both high service availability and high service reliability.

1.2 FAULTS, ERRORS, AND FAILURES

Failure is defined as "*the inability of a system or component to perform its required functions within specified performance requirements*" [IEEE610]. *Error* is primarily defined as "*the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. For example, a difference of 30 meters between a computed result and the correct result*" [IEEE610]. *Fault* is defined as "*(1) A defect in a hardware device or component; for example, a short circuit or broken wire. (2) An incorrect step, process, or data definition in a computer program*" [IEEE610]. Faults are said to be *activated* to become errors, and errors can lead to failures. For example, a software defect (**fault**) in the stopping condition of a *do/while* loop is activated when executed to become an infinite loop **error**, which prevents the system from replying to a user's request within the specified time and thus produces a service **failure**.

Faults can be residual software defects, design weaknesses, or vulnerabilities of internal components and external elements. Practical realities assure that no large and complex software product is ever completely free of residual software faults; defects are inevitable. Hardware components are vulnerable to physical phenomena like bearing wearing out on hard disk drives that eventually cause them to fail. Documented installation and planning guides, procedural instructions, user interfaces, and so on can be unclear, incorrect, misleading, or otherwise prompt humans who operate, administer, maintain, or provision the system to perform incorrect actions. Unexpected, extraordinary, and malicious inputs and conditions challenge deployed systems, and these can exceed the system's designed parameters or expose residual software defects.

Quality activities focus on reducing the number of residual defects in a system's software, hardware, and documentation through careful development and testing processes. For example, written specifications, designs, source code, test plans, and documentation are methodically reviewed and diligently tested. Best-in-class quality organizations will often analyze defect data to predict and quantitatively manage the number of residual software defects (faults) to assure that a system is of appropriate quality before deploying a software release to customers.

An error is an activation of a fault that causes incorrect behavior. Fault activation is a function of the system's profile of operation, including operating time, operational environment, workload, and other characteristics. Most errors will be minor and not cause notable impact, but some may escalate to cause some or all of a system to become incapable of functioning and thus cause a service-affecting failure. For example, consider a common programming error—not initializing a pointer or an array index variable that is used to store or write data—as a sample fault. This fault would be activated by executing the code that uses this pointer or array index variable. Depending on the previous contents of the uninitialized variable, the value of other input

parameters, the system state, the application logic, and the memory map, the system will attempt to write data to some location in the process's address space. If the computed pointer or array index value is outside of this process's writeable address space or the address is misaligned, then the CPU is likely to raise a processor exception and the operating system is likely to cause abnormal termination of the process, thereby producing a system error. If the computed pointer or array index value is a properly aligned address within the process's writeable address space, then the data write is likely to complete successfully without immediately producing a system error. If the system accesses or executes memory that was compromised by the erroneous data write, then a secondary error is likely.

Some errors will escalate and catastrophically impact system operation, thus causing critical failures. If a system doesn't recover from the initial failure promptly, then a cascade of secondary failures may be triggered. These concepts are illustrated with an ordinary pneumatic tire on an automobile or truck in Figure 1.1. A nail on the road presents a hazard or fault that can be activated by driving over it, thereby puncturing the tire to create a hole that leaks air (an error). Over time, this air leak will cause a repairable tire failure, commonly called a "flat tire." If the driver doesn't stop driving on a failed tire quickly enough, then the tire will become irreparably damaged. If the driver continues driving on a flat tire even after the tire is damaged, then the wheel rim will eventually be damaged.

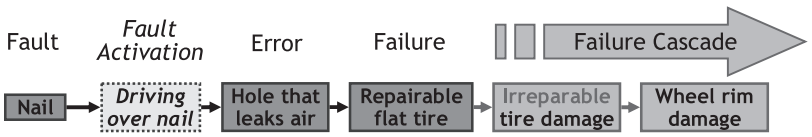

Figure 1.1 Fault Activation and Failures

Robustness is defined as "*the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions*" [IEEE610], which encompasses tolerance of hardware and software faults as well as human and external faults.

1.3 ERROR SEVERITY

Errors are generally classified by severity, with *critical* (often "severity 1"), *major* (often "severity 2"), and *minor* (often "severity 3") being commonly used. For example, consider the definitions of *critical*, *major*, and *minor* severities from the telecommunication industry's TL 9000 quality standard [TL9000]:

> *Problem report—**critical**: Conditions that severely affect the primary functionality of the product and because of the business impact to the customer requires non-stop immediate corrective action, regardless of time of day or day of the week as viewed by a customer on discussion with the* [enterprise] *such as:*

a) product inoperability (total or partial outage),
b) a reduction in the capacity capability, that is, traffic/data handling capability, such that expected loads cannot be handled,
c) any loss of emergency capability (for example, emergency 911 calls), or
d) safety hazard or risk of security breach.

*Problem report—**major**: Product is usable, but a condition exists that seriously degrades the product operation, maintenance or administration, etc., and requires attention during pre-defined standard hours to resolve the situation. The urgency is less than in critical situations because of a lesser immediate or impending effect on problem performance, customers and the customer's operation and revenue such as*
a) reduction in product's capacity (but still able to handle the expected load),
b) any loss of administrative or maintenance visibility of the product and/or diagnostic capability,
c) repeated degradation of an essential component or function, or
d) degradation of the product's ability to provide any required notification of malfunction.

*Problem report—**minor**: Other problems of a lesser severity than "critical" or "major" such as conditions that have little or no impairment on the function of the system.*

Note that some organizations formally or informally include a severity above *critical* such as *emergency* to capture extraordinary events of extreme scope or duration—for example, a failure that is large enough to be reported as a news item in trade publications or general media (e.g., *Wall Street Journal*, Cable News Network, etc.), or that will require external reporting (e.g., for regulatory or contractual compliance), or that triggers liquidated damages payments to customers for violating service-level agreements.

1.4 FAILURE RECOVERY

Failures are typically recovered or repaired by a process that follows three simple steps: failure detection, failure isolation, and failure recovery. Consider recovery illustrated in Figure 1.2 from the road hazard example of Figure 1.1. The example begins with the error of a hole in a tire that leaks air. Some cars will automatically alert the driver when the tire pressure drops below a minimum threshold, while other drivers must rely on a change to the feel of the car's handling, or increased road noise, or perhaps the honking and waving of other drivers. If the road is rough or the driver is distracted or for other reasons, then the driver may not detect the failure right away. After detecting the failure, the driver must safely stop the car. After stopping the car, the driver inspects the tires to diagnose if a tire is flat. While isolating a flat (failed) tire visually is often trivial, isolating failures to a specific repairable or replaceable unit is often nontrivial. The failure is "recovered" by manually replacing the failed tire with the spare. After installing the spare tire and stowing the

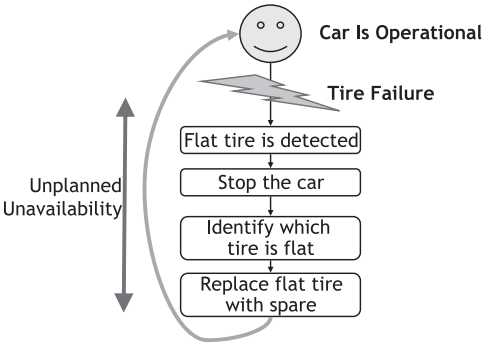

Figure 1.2 Tire Failure as Simple Robustness Example

flat tire, the car is again operational. This completes the *unplanned* downtime associated with the tire failure. Eventually, a planned activity of repairing the flat tire, reinstalling the repaired tire and returning the spare tire to its stowage compartment, must be completed. As the car is unavailable while the tire is repaired and reinstalled, and the spare is returned to its compartment, that period may be considered as planned downtime or planned unavailability.

Generalizing the failure recovery example of Figure 1.2, one can see the three basic robustness steps:

- Failure occurs.
  1. Error is **detected** by system or human being.
  2. Error is diagnosed to **isolate** failure to a repairable or recoverable unit. While it is easy for a human being to visually diagnose which tire has failed, most failures are not trivially isolated by simple visual inspection. For example, consider the challenge of diagnosing typical automobile engine failures to the specific component that must be replaced. The mechanic may follow troubleshooting procedures that rely on both on-board and off-board diagnostics to hypothesize which replaceable unit has failed and must be replaced. If the mechanic incorrectly diagnoses the failure (typically referred to as a *diagnostic failure*), then troubleshooting continues and another likely component failure is hypothesized and that component is repaired or replaced. Ineffective diagnostics can lead directly to higher repair costs and longer unavailability incidents, as any car owner with an intermittent or hard-to-diagnose failure can attest.
  3. Fault is repaired or recovered. Hardware failures are typically repaired by replacing the failed module; software failures are often repaired by restarting a process or the entire system; damaged data may be restored from backup or repaired/rebuilt using system tools, and so on.
- Service is restored.

Sometimes additional actions are required to restore the system to full operational redundancy and readiness (e.g., repairing the failed hardware and/or restocking of spare equipment), but these repair actions are usually completed on a nonemergency basis, and often are not service impacting.

1.5 HIGHLY AVAILABLE SYSTEMS

To reduce cost, typical consumer and commercial systems are permitted to experience some unavailability during failure recovery or repair. For example, tire failure, battery exhaustion of a portable electronic device, or software crash of a PC application all require recovery or repair procedures that include a period of service unavailability. In some commercial, industrial, public safety, and other applications, service unavailability is so costly that it makes business sense to invest more in system hardware, design, and development to minimize service unavailability following failure. Highly available systems are designed so that no single failure causes unacceptable service disruption. To accomplish this, systems must be designed to detect, isolate, and recover from failures very rapidly. Practically, this means that failure detection, isolation, and recovery must be both automatic and highly reliable, and hardware redundancy must be engineered into the system to rapidly recover from hardware failures. A basic robustness strategy for a highly available system is illustrated in Figure 1.3.

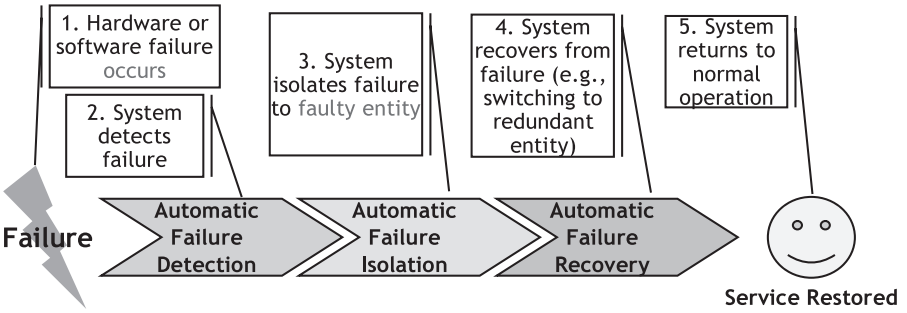

Figure 1.3 Simplified View of High Availability

Consider each step in Figure 1.3 separately:

1. **Failure.** Hardware, software, human, or other failures will inevitably occur.
2. **Automatic failure detection.** Modern systems are designed to detect failures via myriad mechanisms ranging from direct hardware mechanisms like parity checks, to direct software mechanisms like return codes or expiration of timeouts, to environmental sensors like temperature or moisture sensors, to sophisticated indirect mechanisms like

throughput monitors. Highly available systems will have several tiers of failure detection so that if one detection tier misses the event, then another tier will catch it some time later.

3. **Automatic failure isolation.** The system must correctly diagnose or isolate the failure to the appropriate recoverable module so that proper recovery action can be initiated. Fault isolation should be as fast as possible so that failure recovery action can be promptly activated to shorten service outage, but not so hasty as to incorrectly isolate the failure and activate a wrong recovery action. In addition to prolonging the outage event, activating the wrong recovery mechanism (e.g., restarting the wrong software module or rebooting the wrong processor) may unnecessarily impact end users who were not affected by the failure event itself. The situation when a failure is not isolated to the correct recoverable or repairable module is called a *diagnostic failure*.
4. **Automatic failure recovery.** After isolating the failure to the proper recoverable module, then highly available systems will automatically activate the recovery action, such as switching service to a redundant module or restarting a software module.
5. **Service restored.** The system returns to normal operation when service is restored onto the redundant module.

In high availability systems, failure detection, isolation, and recovery occur automatically, and the duration of impact to service should be minimal. Typical high availability systems will automatically detect, isolate, and recover from failures in seconds, but some special purpose systems like optical transmission equipment will detect, isolate, and recover from failures in milliseconds.

If a failure is not automatically detected by the system, then a so called "silent failure" situation will exist in which service is not delivered but recovery actions are not activated because neither the system nor the human maintenance engineers are aware of the failure. A simple example of a silent failure is a frozen water pipe: the pipe freezes, cracks, thaws, begins leaking water silently, and will continue leaking until it is manually detected and the water is manually shutoff. Silent failures are sometimes euphemistically called *sleeping* failures to indicate that the system hasn't noticed the failure because it is "asleep." For example, an underinflated or flat spare tire may be sleeping in an automobile for months or years before being detected. In contrast to sleeping failures, a system might be "dreaming" that a module is operational when it has actually failed, thus misleading surveillance and maintenance engineers. For example, a defective fuel gauge in an automobile might incorrectly report that there is fuel when the tank is actually empty. Depending on system architecture and the specific failure, these silent failures may directly impact users (e.g., a server is down, but the operations team doesn't know it) or they may not immediately impact users but put the system into a simplex or vulnerable state (e.g., spare tire is flat, but the driver doesn't know it).

If automatic failure detection, isolation, and recovery mechanisms are not themselves highly reliable, then excess downtime will be accrued while human beings manually detect, diagnose, and recover unsuccessful automatic failures, as shown in Figure 1.4. If the system does not automatically detect the initial failure, then the failure will probably escalate or cascade to produce more service impact until it is either automatically detected by the system or is manually detected by a human operator. Note that highly available systems typically have several tiers of automatic failure detection, isolation, and recovery mechanisms to increase the overall likelihood of successful automatic recovery. If the system's automatic failure isolation indicts the wrong recoverable module, then the automatic recovery action will not clear the failure, and thus will require manual failure diagnosis by a human operator. If the automatic recovery action fails, then intervention by a human operator will typically be required to clear the failed recovery and restore service. Assuming that automatic failure detection, isolation, and recovery is at least moderately effective, human operators who do detect a failure will often monitor the status of automatic robustness operations before taking any manual action to avoid disrupting automatic failure detection, isolation, and recovery actions that could be progressing (perhaps slowly). Figure 1.4 illustrates the possible interplay between automatic robustness mechanisms and manual recovery.

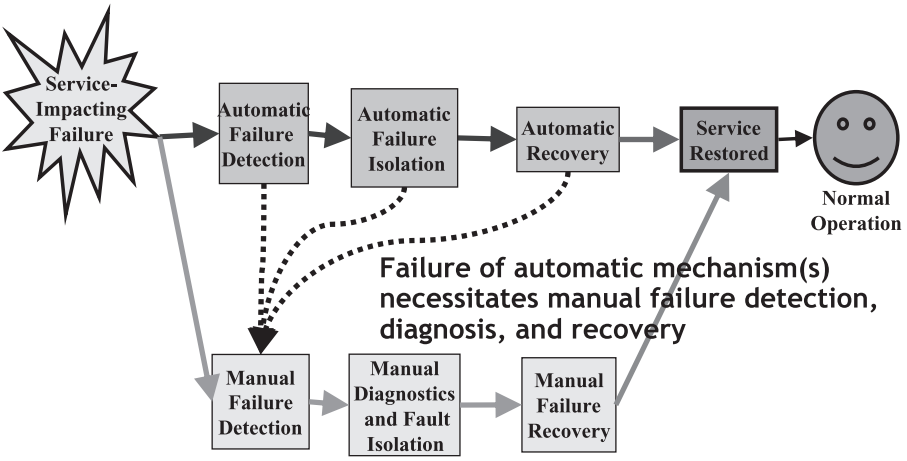

Figure 1.4 Automatic and Manual Recovery

In the real world, robust systems typically follow an elaborate automatic recovery process. There are an infinite number of possible failures that can confront a system. Beyond hardware and software failures, other systems or users can send malformed or malicious requests, human operators can make mistakes, and supporting systems or infrastructure can fail. Myriad failure detectors are integrated throughout the system, such as parity detectors in hardware and timeouts in software. When these mechanisms detect a failure,

then alarm correlation software should isolate the true failure. A recovery strategy is then executed, such as switching service to a redundant element, or restarting a software module. Unfortunately, no single automatic recovery strategy is always effective, and thus a secondary—often more severe—recovery action may be required. For example, if restarting a failed process does not restore service, then it may be necessary to restart all application software on a processor or perhaps restart the processor or entire computer, or take other recovery steps. Thus, the system monitors progress of the automatic recovery, and if the system does not recover promptly, then a secondary recovery mechanism may be activated. There are usually human operators who are responsible for monitoring and maintaining systems, and if the responsible human operators deem that automatic recovery is not progressing acceptably, then they can initiate a manual recovery action. As a practical matter, not all automatic recovery actions succeed either because automatic failure detection, isolation, or recovery didn't work properly, or because the human operator didn't want to wait long enough for automatic mechanisms to complete.

1.6 QUANTIFYING AVAILABILITY

The period when service is available is called *uptime*; the period when service is unavailable is called *downtime*. While most personal and consumer electronics devices support only a single user at a time, most enterprise and commercial systems support many users simultaneously. To account for variations in failure impact to end users for multiuser systems, one can prorate service downtime by capacity lost. For example, a 10-minute outage that impacts half of a system's users or capacity is logically equivalent to a total (100%) capacity loss outage of 5 minutes. Prorating of capacity loss is a practical way to capture the impact of smaller events that affect only a portion of a system's users or subscribers.

Service downtime is the sum of outage duration prorated by capacity lost for all failures in a particular time period. Mathematically, this is:

$$Downtime = \sum_{Failures} OutageDuration * PortionOfCapacityLost$$

Consider each of the input factors separately.

- **Portion of Capacity Lost** captures the portion of system capacity that was impacted by the service outage. Some events, such as power failure, will render a system completely unavailable, and thus 100% of system capacity is unavailable. Other failures may impact only a portion of a system's capacity, such as if users are distributed across several process or hardware instances and one of those instances fails. For example, if user data is distributed across 5 hard disks, each user's data is confined to a single hard disk, and we assume that users are uniformly distributed across the five hard disks, then the failure of a single hard disk will

nominally impact 20% of the system's users. Note that some systems can experience partial functionality outages in addition to partial capacity outages. For example, a voice-mail system might both record messages from callers and play back those recordings to subscribers; if one of those functions fails (perhaps the system can play back previously recorded messages but not record new messages), then that might be considered a 50% loss of functionality and 50% could be used for Portion of Capacity Lost.

- **Outage Duration** is duration of service unavailability for the failure event. Outage duration is normally measured in seconds or minutes and lasts from start of service disruption until service is restored. This duration generally includes the time to detect, isolate, and recover service from the failure.

Section 1.1 explained that availability can also be expressed as:

$$Availability = \frac{Uptime}{Uptime + Downtime}$$

Rather than explicitly calculating uptime, one can simplify the calculation to:

$$Availability = \frac{TotalSystemMinutes - DownMinutes}{TotalSystemMinutes}$$

- **TotalSystemMinutes** represents the number of in-service systems multiplied by the number of minutes in the reporting period. For example, the month of April has 30 days or 43,200 minutes (= 30 days times 24 hours per day times 60 minutes per hour). If 50 systems are in-service, then the *TotalSystemMinutes* for April is 2,160,000 (= 50 systems times 43,200 minutes per system).
- **DownMinutes** is the cumulative, prorated service downtime accrued by the in-service systems in the reporting period. For example, assume that in one 30-day month, 50 deployed systems experience three outages:
  1. 10-minute outage impacting 25% of a single system's users. For example, assume that users are uniformly distributed across four identical frontend processes on each system and one of those processes failed.
  2. 20-minute outage impacting one-third of a single system's primary functionality for all users. For example, consider a social networking site that supports two broad functions: users can their own post content, and users can search and read content posted by others and themselves. If a failure prevents users from posting new content but does not impact the ability to search and read previously posted content, then that failure could be considered a 50% functionality loss event.

3. 30-minute outage impacting all of a system's users. For example, imagine that a janitor plugs a vacuum cleaner into the same electrical circuit providing electrical power to the server hosting the service, and when the vacuum cleaner is turned on the circuit breaker trips, thus causing power outage to the server and a total service outage.

Thus,

$$Downtime = 10 * 25\% + 20 * 50\% + 30 * 100\% = 42.5$$

$$Availability = \frac{2{,}160{,}000 - 42.5}{2{,}160{,}000} = \frac{2159957.5}{2{,}160{,}000} = 0.99998 = 99.998\%$$

Note that some service impairments may be so brief or minor that they will not be classified as outages. For example, few people would classify a momentary disruption of residential AC power that caused lights to flicker as a power outage.

Since there are 525,960 minutes per average year (365.25 days per average year * 24 hours per day * 60 minutes per hour), annualized down-minutes are often expressed as "availability" via the following formula:

$$Availability = \frac{525{,}960 - AnnualizedDownMinutesPerSystem}{525{,}960}$$

Five 9's (99.999%) availability works out to be 5.26 prorated down-minutes per system per year. *Four 9's* (99.99%) availability is about an hour of downtime per system per year (52.6 down-minutes); *three 9's* is about nine hours of annualized downtime.

1.7 OUTAGE ATTRIBUTABILITY

Service outages generally have a single primary cause and may have additional contributory causes that prolong outage duration or increase outage extent. The various causes may be attributable to system or equipment suppliers, system integrator, the enterprise operating the system, others, or a combination. By clearly defining responsibility for actual outage causes, suppliers and enterprises operating the systems can proactively manage their respective outage responsibilities to minimize the overall risk of any outage. TL 9000 factors outage attributability into product or supplier-attributable, enterprise-attributable, and external-attributable outages, and this taxonomy is applicable to a wide range of systems.

1. **Product-attributable or supplier-attributable outage.** Some outages are primarily attributable to the design or failure of the system's software or hardware itself. The telecommunications industry defines product-attributable outages [TL9000] as follows:

An outage primarily triggered by
a) the system design, hardware, software, components or other parts of the system,
b) scheduled outage necessitated by the design of the system,
c) support activities performed or prescribed by an organization [system supplier] *including documentation, training, engineering, ordering, installation, maintenance, technical assistance, software or hardware change actions, etc.,*
d) procedural error caused by the organization [system supplier]*,*
e) the system failing to provide the necessary information to conduct a conclusive root cause determination, or
f) one or more of the above.

2. **Enterprise-attributable outage.** Some outages are primarily attributable to actions or inactions of the enterprise operating the equipment. The telecommunications industry defines this category (called *customer-attributable outage*, in which *customer* refers to the enterprise operating the equipment) as follows [TL9000]:

   An outage that is primarily attributable to the customer's [enterprise's] *equipment or support activities triggered by*
   a) customer [enterprise] *procedural errors,*
   b) office environment, for example power, grounding, temperature, humidity, or security problems, or
   c) one or more of the above.

3. **External-attributable outage.** Some outages are attributable to external events beyond the control of either the enterprise operating the system or the system supplier. The telecommunications industry defines this category as follows [TL9000]:

   Outages caused by natural disasters such as tornadoes or floods, and outages caused by third parties not associated with the customer or the organization such as commercial power failures, 3rd party contractors not working on behalf of the organization [system supplier] *or customer* [enterprise].

Real outages may have a primary cause and one or more contributing factors that caused the outage impact either to increase (e.g., via a failure cascade) or to prolong outage recovery, or both. For example, the primary cause of a flat tire may be an external-attributable road hazard like a nail, but the outage may be prolonged if the driver (*enterprise*, per the taxonomy above) is unable or unwilling to repair the tire himself and had not previously joined an automobile club that could quickly arrange for a repair technician to change the tire. Interestingly, outages are occasionally prolonged by enterprises for deliberate policy reasons. For example, if a small-capacity loss outage occurs on a system during a peak usage period and recovering service will require taking the entire system out of service briefly (e.g., to restart the system), then the enterprise may elect to defer the system recovery to an off-peak period to minimize service impact to other users. Conversely, if an outage occurs in an off-peak period, then under certain circumstances the enterprise may elect to

defer system recovery to normal business hours to avoid the incremental cost of paying maintenance staff overtime to perform the work in off-hours. While the downtime associated with an initial product-attributable failure and undelayed service recovery might reasonably be assigned to the supplier, additional service downtime accrued due to deliberately deferred service recovery should be assigned to the enterprise.

Obviously, system integrators and suppliers should focus on minimizing product-attributable outage causes, and enterprises should focus on both minimizing enterprise-attributable outages and mitigating the risk of external-attributable events. For example, enterprises can install uninterruptable power supplies to mitigate the risk of variations and disruption in external, commercial AC power. System integrators and equipment suppliers often provide recommendations and guidance to enterprises to minimize risk of enterprise- and external-attributable outages, such as offering training for enterprise staff to minimize risk of human error.

1.8 HARDWARE RELIABILITY

System hardware is packaged in field-replaceable units (FRUs) that can be individually replaced. Replaceable parts on home appliances, automobiles, computer systems, and other products are FRUs. Hardware reliability addresses how often each of these FRUs will fail. The following sections give a basic review of hardware reliability, service life, and return rates, and discusses typical system considerations related to hardware reliability.

1.8.1 Hardware Reliability Background

Hardware reliability is much better understood than software or system reliability for several reasons. First, hardware fails for physical reasons, and persistent (versus transient) hardware failures can be thoroughly analyzed to determine the precise failure mode and the likely root cause(s). Second, since early electronic hardware (e.g., vacuum tubes) was often prone to high failure rates, engineers have been working to understand and improve hardware reliability since at least World War II and the physics of hardware failure are now well understood. Third, actual hardware reliability of deployed elements is generally easier to measure in the field than software reliability because hardware failures generally require physical replacement of failed FRUs and/or rework of failed connections to repair, rather than simply restarting, a system, or reseating an FRU that reboots some or all of a system's software.

Hardware failure rates generally follow the so-called "bathtub curve" illustrated in Figure 1.5. The X-axis shows operational time and the Y-axis shows failure intensity or rate. Some FRUs will quickly fail to operate because of weak components, solder joints, or manufacturing quality factors. FRUs that

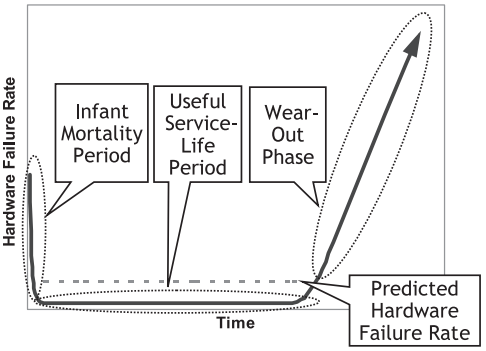

Figure 1.5 "Bathtub" Hardware Failure Rate Curve

fail to operate the first time they are powered on in the field are often called "dead on arrival" (DOA). Over the first days and weeks of operation, some poorly manufactured parts and assemblies are likely to fail; the time when weak FRUs fall out is called the "infant mortality period". The rate of early life failures declines quickly as the weak hardware fails and the failure rate stabilizes; the period of stable hardware failure rate is called the "useful service-life period." As the useful lifetime of the hardware expires, the hardware enters the wear-out phase and the rate of hardware failures increases due to wear-out factors. At some point in the wear-out phase, all hardware elements will have failed, and thus the curve ends.

1.8.2 Hardware Reliability Prediction

Hardware failure rate prediction methodologies estimate the failure rate in the useful-life period. MIL-HDBK-217F [MIL217F] and Telcordia's SR-332 [SR332] standards for hardware reliability prediction are common in the industry. Prediction methodologies consider various factors, including the parts used in the design and assumptions about operational characteristics like ambient temperature. Assumptions and prediction models tend to be conservative, so observed hardware failure rates during useful-life periods are often much lower than standard prediction methodologies calculate. As customers expect hardware failure rates to be less than predicted failure rates throughout the useful service life, hardware suppliers have historically been motivated to give conservative hardware failure rate predictions to minimize the risk of exceeding official "predicted" hardware failure rates. Since different prediction methodologies and different prediction assumptions can give significantly different failure rates, it is useful to calibrate and validate a supplier's predictions against historical data to understand how pessimistic or realistic their predictions are likely to be. Hardware failure rates are highly dependent on temperature; the higher the temperature, the higher the hardware failure rate

and perhaps the shorter the useful service life. Thus, if the assumed ambient temperatures are not consistent with the actual ambient temperatures of deployed systems, then the predicted hardware failure rates can be different from actual observed values.

Hardware failure rates are commonly expressed as "failures in 1 billion hours" (FITs). Alternately, hardware failure rates may be expressed as "mean time between failures" (MTBF). MTBF and FIT rates are mathematically related as

$$MTBF_{Hours} = \frac{1,000,000,000}{FITs}$$

1.8.3 Hardware Service Life

Although MTBF and service life are often expressed in the same unit (hours or years), they are completely different concepts. Hardware service life is the period that hardware should operate before the hardware failure rate rises above the predicted hardware failure rate as wear-out failures increase. A predicted hardware failure rate estimates the rate of failure during the useful service life, rather than during the wear-our or infant mortality periods. While the hardware service life of most electronic devices is significantly longer than the expected useful life of the system those devices are in, some devices with mechanical parts may have service lives that are shorter than the expected useful life of the system. As hard disk drives have moving parts with lubricated bearings, they generally have a hardware service lifetime of less than 5 years. For example, a hard disk drive manufacturer may quote a mean time between failures (MTBF) of 1 million hours, which is mathematically equivalent to a predicted failure rate of 10^{-6} failures per hour (10^{-6} is the mathematical reciprocal of 10^{6} MTBF). The designed hardware service lifetime might be 5 years (43,430 hours) and the predicted hardware failure rate during that useful lifetime might be 10^{-6} hardware failures per unit per hour. An MTBF of 1 million hours is equivalent to an MTBF of 114 years, but with a designed hardware service life of 5 years, very few units will survive to even 10 years. Experience and common sense assures us that the moving parts in a typical hard disk drive will wear out after *far less* than a century of actual service. Figure 1.6 graphically illustrates this example for a hypothetical hard disk drive.

Service life creates a reliability risk if customers expect to keep the system in service beyond the designed hardware service life of any individual FRUs because of the increasing hardware failure rate that occurs after the FRU enters its wear-out phase. Having some FRUs with a shorter lifetime in a complex system is a very common phenomenon. For example, consumers expect to replace the tires and battery on their car at least once before the car wears out, and they accept that light bulbs, appliances, carpets, roofs, and so

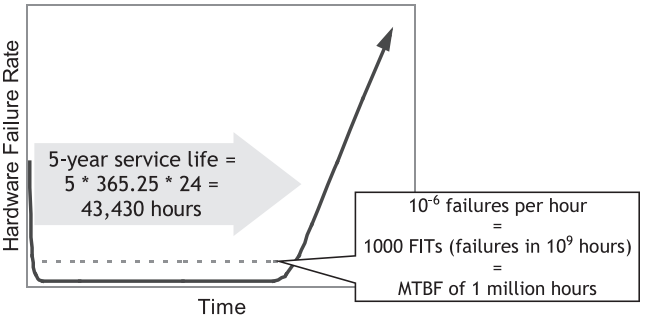

Figure 1.6 MTBF and Service Life of Sample Hard Disk Drive

on will require replacement long before their house reaches the end of its useful life. Thus, system suppliers should carefully design to assure that hardware elements that are likely to fail before the system reaches the end of its designed hardware service life can be easily replaced. Just as light bulbs, batteries, and tires can be replaced relatively easily, failed fans, hard disks, batteries, and so on should be easily replaceable in the field so that customers are not forced to endure prolonged service disruption or exceptional expense to replace worn-out hardware components. Like automobile manufacturers recommending maintenance schedules to replace filters, tires, batteries, and other components that are not designed to last the entire useful service life of the automobile, both hardware and system suppliers should clearly communicate the service-life expectation of any FRUs that are expected to wear out before the system itself reaches the end of its useful service life.

1.8.4 Hardware Return Rates

While some system hardware elements like batteries and fans are considered consumable and are discarded on failure, most hardware will be repaired following failure. For repairable hardware, the rate at which hardware elements are returned for repair is a common measure of hardware reliability, and hardware return rates are a reasonable proxy for the hardware failures that customers experience for nonconsumable hardware units. Hardware return rate measures are generally expressed as *annualized return rates*, and may be further subdivided into *time windows*, such as TL 9000's metrics:

- Early Return Index (ERI) measures hardware returns in the first 6 months after shipment.
- Yearly Return Rate (YRR) measures hardware returns 7 to 18 months after shipment.
- Long-Term Return Rate (LTR) measures hardware returns more than 18 months after shipment.

Hardware reliability predictions can be converted to equivalent annualized return rate predictions as follows:

$$\mathrm{Re}\,turnRate_{Annualized} = \frac{1}{MTBF_{Annual}}$$

$$\mathrm{Re}\,turnRate_{Annualized} = \frac{24 * 365.25}{MTBF_{Hourly}} = \frac{8,766}{MTBF_{Hourly}}$$

$$\mathrm{Re}\,turnRate_{Annualized} = \frac{24 * 365.25 * FITs}{1,000,000,000} = \frac{FITs}{114,077}$$

As explained in Section 1.8.2, "Hardware Reliability Prediction," hardware suppliers often use fairly conservative assumptions so that the best estimate of actual return rates will generally be below prediction.

The hardware return rate metric is complicated by several factors:

1. **Not all hardware returns are attributed to hardware failures.** Inevitably, some returned hardware FRUs will be tested at the repair center and found to function properly with *no trouble found* (abbreviated *NTF*; sometimes called *no fault found—NFF*). No-trouble-found hardware is generally attributed to a variety of causes including software failures misdiagnosed as hardware failures, and diagnostic failures in which functional hardware was incorrectly deemed to have failed and was thus returned. In addition, unused materials may also be returned to the supplier because too much material was ordered, or because the order was changed or canceled or for other reasons. Hardware may also be returned in response to product recall or change notices to have recommended corrective changes applied. The percentage of confirmed hardware failures as a portion of total returns often varies significantly based on the nature of the FRU. High NTF rates are a concern because although the hardware reliability does not appear to be a problem, one or more root causes are forcing customers to spend effort and cost to manually remove and return hardware that had not failed.
2. **Uncertainty in the actual number of FRUs in service to normalize returns against.** It is very easy for hardware suppliers to count how many FRUs are shipped from their factory, but it is not very easy to know how many FRUs are actually in service. There is always a delay between the time an FRU is shipped from a factory to the time that the FRU is installed in a system and powered on. Some individual FRUs will be promptly delivered to customer sites, installed and powered on; other FRUs will endure logistics or staging delays; some FRUs will be stocked as spares, and may not be installed for years; some FRUs may never even be sold. The uncertainty of actual number of FRUs in service is often highest in the early months of production

as the logistics and delivery process is being refined and spares pools are created.

3. **Uncertainty of in-service time of individual returned FRUs.** Because complex systems are generally built from many different FRUs and systems cannot go into service until all required FRUs are available on-site, installed, configured, and brought into service, variable logistic and business delays add uncertainty between the time a FRU was shipped by the hardware supplier and the time the FRU begins normal, continuous operation. In addition, some FRUs will be held as spares, and these may spend months or longer in factory packaging without ever being powered on. Thus, it may be hard to reliably determine how many hours of service a particular FRU sustained before it was returned. Just as automobiles include odometers to maintain a persistent count of the distance driven, the best practice is for each FRU to record wear-out-related parameters like power-on hours and maximum operating temperature in persistent-on-FRU storage as part of an "odometer."
4. **Not all failures are returned to the equipment manufacturer's authorized repair center.** Just as automobile owners are not required to take their cars to factory authorized service centers for repairs, system owners may send their failed FRUs to competitive repair centers, and thus hardware suppliers may have no visibility or knowledge of those failures. With inexpensive or short service life FRUs, customers may simply discard the failed unit rather than returning it for repair.

Despite these complicating factors, hardware return rates are an excellent measure of actual hardware reliability. Hardware suppliers should be able to provide return rate data for each FRU, and the confirmed hardware failure rate should be well below predicted hardware failure rate. If the overall return rate is above prediction or the no trouble found rate is substantial, then there is significant risk that system owners may experience higher apparent hardware failure rates than predicted. After all, a perceived hardware failure (even if it is later deemed to be "no trouble found" by the repair center) causes maintenance engineers to take the time to replace a FRU and the expense of returning that FRU, restocking a spare, and so on. From the system owner's perspective, a confirmed hardware failure and a no trouble found failure have similar cost, but the no trouble found pack may offer an incremental intangible cost of uncertainty surrounding the true root cause of the failure and thus elevated risk of failure recurrence.

1.8.5 Hardware Reliability Considerations

The primary considerations when selecting hardware are obviously functionality ones, such as:

- **Does the hardware deliver the required functionality with adequate performance?**
- **Does the hardware meet the physical design requirements?** (e.g., size, weight, form factor, power)
- **Does the hardware operate reliably in the target environment?** (e.g., temperature and humidity range, shock and vibration tolerance, presence of dust and corrosive gases, altitude, etc.)
- **Does the hardware meet the cost target?**

Hardware reliability is a secondary consideration for hardware options that meet the primary requirements. The fundamental business question when considering hardware reliability is

- **Will the hardware failure rate be acceptably low throughout the system's designed service life?**

Note that higher hardware failure rates may lead to higher operating expenses associated with increased hardware returns and sparing-related expenses. Acceptability of hardware failure rates is fundamentally driven by how often customers will tolerate executing hardware repairs or maintenance actions on the overall system. Just as consumers evaluate reliability of their automobile based on how often the car is in the shop rather than on the failure rate of individual subsystems or components, system owners are likely to consider how often they have to replace any board or FRU on a system rather than the failure rate of individual blades. Thus, there is likely to be some flexibility in failure rate requirements of individual FRUs so long as the overall hardware failure rate remains low enough that the customer doesn't have to repair hardware too often. Deep knowledge of customer expectations enables one to estimate the maximum acceptable system hardware repair rate, and manage individual FRU failure rates to be low enough to meet the overall system target.

End customers will expect hardware reliability to be below predicted failure rates and may include business remedies, such as contractual penalties with liquidated damages, if hardware returns exceed predicted values. System suppliers should assure that appropriate business arrangements are made with original equipment manufacturers so that risk of premature (i.e., warranty) or excessive hardware failures is covered by the hardware manufacturer who is best equipped to manage the risk.

1.9 SOFTWARE RELIABILITY

Although software doesn't physically break or fail in a persistent way that can be easily examined with an optical or electron microscope, it does fail or crash. Figure 1.7 shows an instance of the infamous *blue screen of death* software failure from Microsoft Windows®, which many readers will have personally

Figure 1.7 Blue Screen of Death

Source: Figure taken from Wikipedia, http://en.wikipedia.org/wiki/File:Windows_XP_BSOD.png, June 15, 2010.

experienced. In addition to the dramatic "blue screen," a critical software failure event may simply be called a *crash* or *system hang*. Practically speaking, software reliability is measured as the rate of critical software failures (e.g., crashes, hangs, blue screens), as in "application X crashes once a month unless we reboot the system."

Software reliability growth theory [Musa89] posits that there are a finite number of defects in a piece of software that can be exposed in a particular operational profile. While hardware wears out over time, software does not; new defects don't spontaneously arise over time in software as cracks and physical breakdowns arise in hardware. Thus, as residual software defects are discovered and removed, there are fewer defects left to be exposed in a particular operational profile. Fewer critical residual software defects should be encountered less frequently in normal operation, and thus should yield a lower software failure rate. This is generally demonstrated in maintenance or patch releases, which are often more stable and reliable than major releases of software because major releases inevitably introduce new defects along with new features, while maintenance or patch releases focus on removing software defects. Since new defects are seldom introduced when a software defect is fixed, each critical defect removed is essentially one less residual defect to cause a software failure in the field.

The next section defines operational profiles, describes software reliability growth theory, reviews several residual defect prediction techniques, and discusses how to evaluate prediction results.

1.9.1 Operational Profile

Operational profile characterizes how a system interacts with other elements in the enterprise's solution to deliver services to end users. A particular system can often be used in several different operational profiles, each of which may stress the system in slightly different ways, thus exposing somewhat different residual faults. For example, a four-door sedan automobile can be used both to carry passengers on paved roads and to haul bricks on dirt roads. Each of these profiles exposes the automobile to different shock, vibration, and mechanical stress, and thus the time to first failure of the automobile may be different in each of these two operational profiles. System testing should reflect the operation profile(s) that the deployed system will experience to assure that the vast majority of design and residual defects are discovered and corrected before field deployment.

Note that the operational profile of a deployed system can be changed by an enterprise after the system is deployed. For example, an enterprise (or perhaps even their end users) can change their policies and begin using names and strings containing Asian characters. While Western character sets such as ASCII can often be encoded as a single byte per character, Asian characters require two or more bytes per character, and thus the logic for manipulating and encoding Western and Asian character sets may be somewhat different. If Asian characters were not included in the system's test campaign, then there is a higher risk that residual defects will be encountered in field operations with Asian characters, such as importing and exporting system information via text configuration files. Hence, differences between *tested* operational profiles and *field* operational profiles present gaps in testing through which undiscovered software defects can escape to the field.

1.9.2 Software Reliability Growth Theory

When testing a system, residual defects are methodically exposed, so they can then be debugged and corrected. Discovery of residual defects generally progresses as shown in Figure 1.8. The X-axis shows the cumulative time system testers spend executing tests against the target system; the Y-axis shows cumulative number of defects discovered. Defect discovery often starts slowly as the test team progresses through a "*learning phase*" during which they become familiar with both the system and the test tools. As testers gain experience, their productivity increases and they enter a "linear phase" in which they discover defects fairly regularly as new test cases are executed and defects are exposed and recorded. This consistent defect discovery rate is shown as the

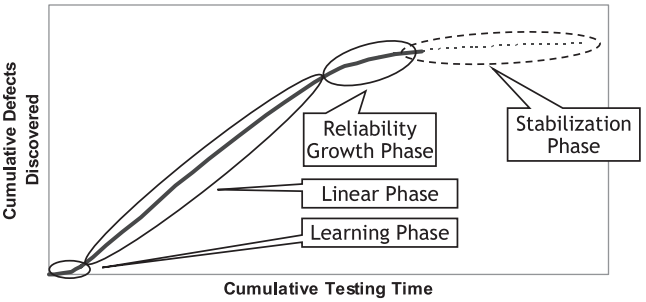

Figure 1.8 Canonical Software Reliability Growth

linear phase of defect discovery. Defects discovered throughout the test interval will generally be promptly fixed and new or patched software loads will be delivered to system test. These patched loads should enable system test to cover system functionality that may have been previously blocked by software defects. Thus, as more defects are discovered and removed, system test can verify more and more system functionality. Inevitably, a fraction of the defect corrections will themselves be defective, so some rework of defect correction (sometimes called "fix on fix") will be necessary. The rate of defective fixes should be very small, and thus the total number of residual defects should not increase significantly after software development is complete and formal system testing begins. After sufficient testing, debugging, and defect correction, the number of residual defects shrinks to the point that it becomes harder and harder for testers to discover previously unknown residual defects, and thus the previously linear discovery rate begins to flatten out. This transition from linear defect discovery rate to decreasing defect discovery rate characterizes the "*reliability growth phase.*" As the number of discovered defects asymptotically approaches the finite number of defects originally accessible to the piece of software in the particular operational profile, it takes more and more test effort to discover residual defects. At some point, the number of residual defects is so small that the system will operate with an acceptably low software failure rate, and the software enters the "*stabilization phase.*" As testing costs money and adds time to the schedule, a key business question is deciding when sufficient testing has been executed to assure that the system will be acceptably reliable.

Note that the X-axis in Figure 1.8 is labeled "*cumulative testing time*" rather than calendar time. This is because the independent variable that drives defect discovery during testing is actual test effort, rather than calendar time. The distinction between "testing time" and "calendar time" is important because test effort is rarely uniform across a 7-day week, and effort per week is rarely uniform across the entire test period. For example, test staff will take vacations and will have other work commitments like training and supporting other projects or releases that reduce their test time. System software failures

may block some test cases and hardware failures may render test cells unavailable for testing. Also, testers may be more likely to work overtime toward the end of the test interval than they are in the start of the test interval. Normalizing defect discovery against cumulative testing time should factor real world variations out of the data, and thus produce a more accurate curve that is easier to interpret. A practical benefit of plotting cumulative test time on the X-axis is that it makes planning easier because one can directly estimate the test effort required between any two points on the curve.

1.9.3 Estimating Residual Defects

The number of residual defects can be estimated using a number of software quality methodologies, including [Demarco86], [Humphrey89], [Lyu96], or [Musa89]. Software reliability growth modeling [Musa89] has several advantages over some traditional software quality methodologies, including:

1. **It does not require deep knowledge of process performance of specification, design, and coding activities.** Importantly, this means that one can consider the behavior of software modules that were developed by others (e.g., third parties, open source, or reused modules) for which no process performance data is available.
2. **It provides an intuitive visualization that is easy for all members of a project team to understand.**
3. **Required input data is easy to acquire.**

Figure 1.9 shows a sample software reliability growth curve with a vertical line showing "system release" after hundreds, thousands, or more hours of cumulative testing time. The slope of the software reliability growth curve shows the rate at which critical software defects are discovered, often expressed as number of hours or days of testing estimated to discover the next previously

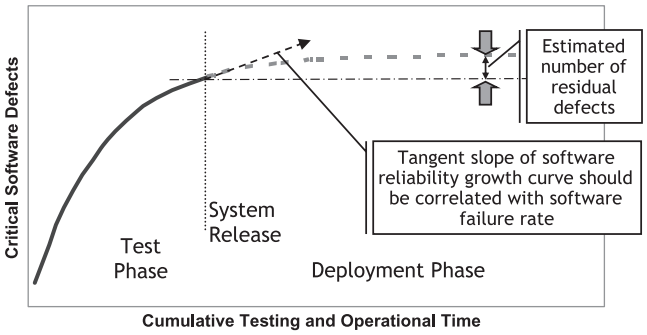

Figure 1.9 Estimating Residual Defects from Software Reliability Growth Model

unknown defect. As system testing is designed to mirror the operational profile of target customers, system testing should be a form of highly accelerated stress testing of the system because testers are deliberately trying to stress the system. Thus, the longer it is estimated to take dedicated system testers to expose a previously unknown residual defect, the longer it should take an enterprise or end user to encounter a previously unknown residual defect. The gap between the number of defects discovered at the time of system release and the asymptotic number of defects estimated by the software reliability growth curve estimates the number of residual defects.

The physics of hardware failure enables one to reliably estimate the failure acceleration factor due to elevated temperatures, and so on, in highly accelerated stress testing of hardware. Unfortunately, the "physics" of software failures is far messier than the physics of hardware failure, and thus there is far more uncertainty in estimating the "acceleration" or calibration factor between the defect discovery rate in lab testing and the critical software failure rate in field operation. Nevertheless, assuming system testing is mirroring the operational profile(s) of field deployments, one expects a rough correlation between the defect discovery rate at system release and the critical failure rate in field operation.

1.9.4 Calibrating Residual Defect Models

While it theoretically takes infinite time to activate all residual defects and perfect debugging to determine which are truly *new* residual defects versus activation of previously known defects, project teams generally can estimate the minimum effective asymptotic number of critical defects after about a year of field deployment by summing both the number of unique field-found defects and the number of additional, previously unknown defects that were discovered when testing maintenance and patch loads on the release. Depending on operational policies of enterprises operating the system and their support agreements with the system supplier, some or all of those failures may be reported back to the system supplier for analysis and corrective action. Thus, while a system supplier may never know exactly how many residual critical software defects were in a piece of software released to the field, the supplier often knows the minimum number of residual critical defects that were present based on what defects were subsequently detected by customers or while developing and testing maintenance releases or patches. This minimum effective asymptotic number of estimated critical defects can be compared to the number of defects predicted by the software reliability growth or other defect prediction model used. Since system testing inherently differs from actual field deployment in a variety of ways, there may be somewhat more observed residual defects than were estimated from modeling of system test data. Nevertheless, system testers should constantly strive to make their test configurations better reflect actual operational profiles to improve the

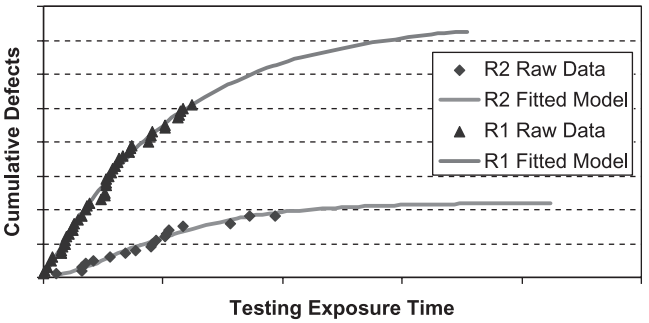

Figure 1.10 Comparing Software Reliability Growth of Different Software Releases

effectiveness of their testing, and thereby improve the expected prediction accuracy of residual defects based on system test data.

While it is often difficult to accurately and quantitatively extrapolate a single test or analysis result to likely field availability of a particular release, it is often insightful to compare and contrast test results from historic releases and current release against actual field performance of historic releases to triangulate a best estimate of reliability and availability. For example, Figure 1.10 overlays the software reliability growth analysis of two releases of a real system. Although R2 benefited from significantly more test time than R1, there appear to be far fewer critical defects and thus R2 should have a significantly lower software failure rate than R1. By combining this R2-versus-R1 comparison data with other R2 to R1 comparison data, one can estimate a range of how much lower the critical software failure rate of R2 is likely to be relative to R1. A software reliability growth curve, especially one that overlays previous release data like Figure 1.10, is often very useful to decision makers when evaluating software readiness for release.

1.10 PROBLEMS

1. Give three examples of fault activation that lead to service failure.
2. Give three examples of a failure cascade.
3. One 30-minute total outage and one 10-minute partial outage impacting half of a system's capacity occur across a population of 15 systems deployed in August; what is the service availability for the month?
4. What is the difference between MTBF and designed service life of hardware?
5. A particular FRU is predicted to have 10,000 FITs. What percentage of these FRUs is likely to fail per year during the unit's designed service life?
6. A system is built of five FRUs, each with 10,000 FITs. What is the annual predicted rate of hardware maintenance actions per system?

7. A particular FRU is predicted to have 10,000-hour MTBF. What percentage of these FRUs is likely to fail per year during the unit's designed service life?

1.11 FOR FURTHER STUDY

[TL9000] offers formal and rigorous guidance on quantitative measurement of service availability and hardware reliability. [Lyu96] and [Musa89] offer more information on software reliability and [O'Connor04] covers hardware reliability.