Chapter

6

Best Current Practice of SRE

Mary Donnelly, Bill Everett, John Musa, and Geoff Wilson
AT&T Bell Laboratories

6.1 Introduction

This chapter describes the best current practice (BCP) for doing software reliability engineering (SRE) as adopted by AT&T. It represents an integrated consensus of some 70 software project managers and engineers based on practices employed by a large number of projects within AT&T. Consequently, to maintain its integrity, we have not incorporated material from other sources.

The practice of SRE provides the software engineer or manager the means to predict, estimate, and measure the rate of failure occurrences in software (including firmware). Such measures are understandable to your customer. Using SRE in the context of software engineering, you can:

- Analyze, manage, and improve the reliability of your software products.
- Balance customer needs for competitive price, timely delivery, and a reliable product.
- Determine when the software is good enough to release to your customer, minimizing the risks of releasing software with serious problems.
- Avoid excessive time to market due to overtesting.

Although portions of SRE are based on sophisticated statistical concepts, available software tools as discussed in App. A allow the average software developer or engineer to apply SRE easily, without the need for statistical background.

The practice of SRE may be summarized in six steps:

1. Quantify product usage by specifying how frequently customers will use various features and how frequently various environmental conditions that influence processing will occur.
2. Define *quality* quantitatively with your customers by defining failures and failure severities and by specifying the balance among the key quality objectives of reliability, delivery date, and cost to maximize customer satisfaction.
3. Employ product usage data and quality objectives to guide design and implementation of your product and to manage resources to maximize productivity (i.e., customer satisfaction per unit cost).
4. Measure reliability of reused software and acquired software components delivered to you by suppliers, as an acceptance requirement.
5. Track reliability during test and use this information to guide product release.
6. Monitor reliability in field operation and use results to guide new feature introduction, as well as product and process improvement.

This chapter will help assess the benefits and costs of doing SRE in your organization or project. It will also aid you in visualizing how to practice SRE and how to get started. It provides information on resources and examples of successful applications.

6.2 Benefits and Approaches to SRE

SRE predicts, models, estimates, measures, and manages the reliability of software-based products. It extends from product conception through delivery and use by the customer. Again, *reliability* is the probability that a hardware or software system or component does its required functions without failure for a specified period of time under specified operating conditions.

Software reliability varies with how and in what environment the software is used. Characterizing customer operating conditions as precisely as possible is an important part of SRE. This is done with a *profile* or set of alternative uses and their occurrence probabilities under different environmental conditions, as discussed in Chap. 5. Early in the software development life cycle, you deal with *functions* or tasks needed by the user in different environments and determine a *functional profile.* Later, when you know how tasks will be implemented by the system as *operations,* you translate the functional profile to an *operational profile.*

6.2.1 Importance and benefits

Most surveys of users of software-based systems show that reliability ranks first on the list of customer satisfiers. Eighty percent of AT&T's largest customers considered reliability to be *the most important quality attribute* to them. With greater use of software in products, the proportion of total failures that are software related is increasing. Some projects report that the number of customer-reported software failures of their products exceeds the number of hardware failures. To be successful in the marketplace a product must meet the quality expectations of its customers. To meet this end, you must be able to measure quality from your *customer's* perspective.

Fault-based measures commonly used in the software development industry have served well as developer-oriented measures of quality. They have helped to improve the quality of development processes in the past and will continue to play an important role. However, users of software products experience failures, not faults. Your customers are not concerned as much with how many faults there are in your software product as they are with how often the product will fail and the impact such failures will have on the job they must do. Evaluating reliability of software from the customer's perspective requires *failure-based measures.*

SRE will help your project

- *Satisfy customer needs more precisely.* Having precise reliability requirements focuses development on meeting your customers' reliability needs. Reliability requirements enable system testers to concretely verify that the finished product meets customers' needs before it is released.
- *Deliver earlier.* Delivering the exact reliability needed by the customer avoids wasting time for unneeded extra testing.
- *Increase productivity.* By using the functional and operational profiles to focus resources on the *high-usage* functions or operations and by developing and testing for *exactly* the reliability needed, productivity is improved.
- *Plan project resources better.* Before testing begins, SRE supports prediction of the amount of system test resources needed, avoiding unnecessary waste and disruption due to unpleasant surprises.

6.2.2 An SRE success story

AT&T's International DEFINITY® PBX started a new quality program that included doing SRE along with other proven quality methodologies [Abra92]. The SRE part of the program included

- Defining an *operational profile* based on customer modeling
- Generating test cases automatically based on frequency of use reflected in the *operational profile*
- Delivering software in increments to system test with quality factor assessments (*reliability* being one factor)
- Employing clean-room development techniques together with feature testing based on the *operational profile*
- Testing to reliability objectives

The quality improvement from the previous major release was dramatic:

- A *factor-of-10 reduction* in customer-reported problems
- A *factor-of-10 reduction* in program maintenance costs
- A *factor-of-2 reduction* in the system test interval
- A *30 percent reduction* in new product introduction interval

In addition, no serious service-affecting outages were reported in the first two years of operation. There was significant improvement in customer satisfaction. The reliability improvement and an aggressive sales plan have pushed sales to 10 times those for the previous version. Items contributing to these successes were as follows:

- Using *reliability* as a release criterion prevented excessive customer-reported problems and associated maintenance activity.
- Using *operational-profile-driven testing* increased test efficiency: 20 percent of the operations represented 95 percent of the use; 20 percent of the faults caused 95 percent of the failures; testing the 20 percent high-usage operations first speeded reliability improvement.

6.2.3 SRE costs

The principal cost in applying SRE is determining the operational profile. The effort depends on the number of operations defined and the precision of their occurrence probabilities. You must use engineering judgment to control the amount of work. In our experience, effort has ranged from one staff week to one staff year, depending on the size and the complexity of the project. The average effort (for a project with 10 developers, approximately 100,000 source lines and a development interval of about 18 months) is about 2 staff months. Once you develop the operational profile for a product, you need only update it for subsequent releases, a much smaller task. Another cost is associated with

processing and analyzing failure data during reliability growth testing, which requires about 4 hours per week. The total SRE effort during a test is thus usually less than 0.3 staff month. There is a training cost in starting to apply SRE. At least one project representative needs a course in SRE. Also, engaging an SRE consultant to transfer SRE knowledge to the project during initial implementation (jump-start consulting) is cost-effective. The cost (about one staff month of consultant's time) is offset by preventing costly mistakes in applying SRE.

Most SRE activities involve using SRE measures to better perform tasks that projects normally do anyway. SRE does not contribute more cost to these tasks. As most projects have multiple releases, and SRE cost drops sharply after initial application, the per-release cost of SRE is about one staff month. The very largest projects (over 1000 software developers and multiple large operational profiles) do not exceed an effort of one person full time. Thus, for most projects, SRE costs considerably less than 1 percent of total project cost.

6.2.4 SRE activities

Figure 6.1 illustrates SRE activities across the phases of the software product life cycle. The phases, of course, need not follow a neat sequence, and neither do the SRE activities. There is considerable overlap and iteration. Each SRE activity is therefore simply placed in the phase in which most of the effort occurs. Sections 6.3 to 6.6 each discuss the activities of one phase in detail.

6.2.5 Implementing SRE incrementally

Most projects start by implementing just some of the SRE activities. As they learn to apply and profit from these activities, they add additional ones. A typical implementation sequence is shown in Table 6.1, with the benefits from each step noted. The sequence applies to both new and existing projects. This incremental approach keeps the rate of change in your development process to a manageable level.

The order of implementing the sets of activities in Table 6.1 is the recommended one. However, there is considerable *flexibility* in choosing the sets of activities to implement, and in modifying the sequence to meet your project's needs. If there are external factors that hamper the implementation of a particular set, implement another set of activities in the sequence and return to the first one when the difficulties are resolved. For example, although monitoring field reliability is important and should be implemented early, the difficulties associated with obtaining field data may require it to be implemented later. It's a good practice to have an expert check the SRE implementation plan to

make sure you haven't inadvertently left out a prerequisite activity. The section references in the "Details" column of Table 6.1 indicate where to read more about the activities.

6.2.6 Implementing SRE on existing projects

There is no essential difference between new and existing projects in applying SRE for the first time. After it has been applied to one release, however, you will need much less effort for succeeding releases. Most of the activities noted in Fig. 6.1 will require only small updates after they have been completed one time.

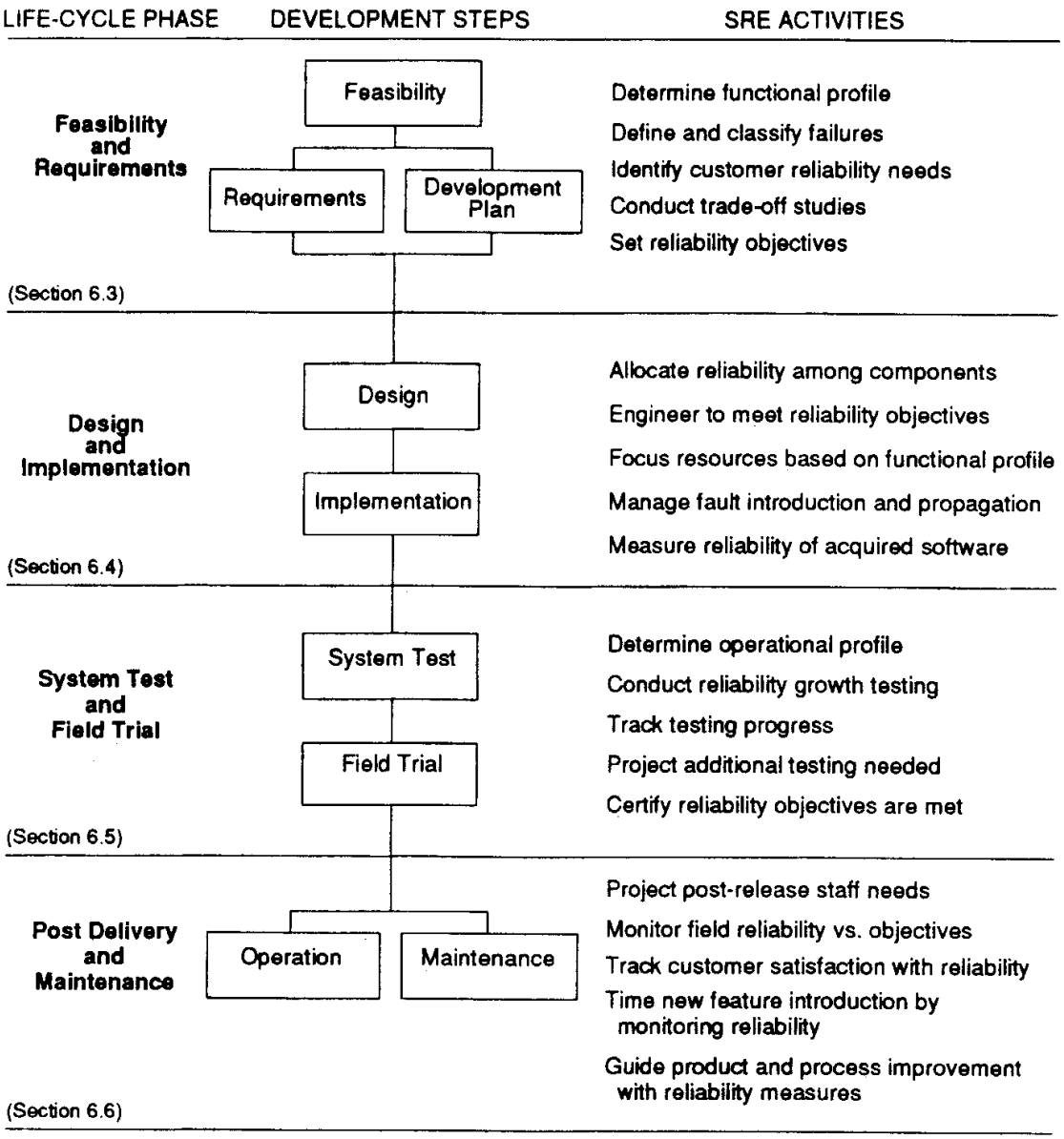

Figure 6.1 SRE activities in the software product life cycle.

TABLE 6.1 Typical Incremental Introduction of SRE

Set of activities	Benefits	Details
1. Conduct reliability growth testing, track testing progress	Know exactly what reliability your customer would experience at different points in time if you released the software at those points	6.5
2. Determine functional and operational profiles	Speed up time to market by saving test time, reduce test cost	6.3, 6.5
3. Define and classify failures, identify customer reliability needs, set reliability objectives, certify reliability objectives are met, project additional testing needed	Release software at a time that meets customer reliability needs but is as early and inexpensive as possible	6.3, 6.5
4. Monitor field reliability versus objectives, track customer satisfaction with reliability	Maximize likelihood of pleasing customer with reliability	6.6
5. Time new feature introduction by monitoring reliability	Ensure that software *continues* to meet customer reliability needs in the field	6.6
6. Focus resources based on functional profile	Speed up time to market by guiding development priorities, reduce development cost	6.4
7. Allocate reliability among components	Reduce development time and cost by striking better balance among components	6.4
8. Measure reliability of acquired software	Reduce risks to reliability, schedule, cost from unknown software	6.4
9. Engineer to meet reliability objectives	Reduce development time and cost with better design	6.4
10. Guide product and process improvement with reliability measures	Maximize cost-effectiveness of product and process improvements selected	6.6
11. Manage fault introduction and propagation	Maximize cost-effectiveness of reliability improvement	6.4
12. Project postrelease staff needs	Reduce postrelease costs with better planning	6.6
13. Conduct trade-off studies	Increase market share by providing better match to customer needs	6.3

6.2.7 Implementing SRE on short-cycle projects

Small projects or releases or those with short development cycles may require a modified set of SRE activities to keep costs low or activity durations short. A reasonable approach is to select a reduced group of sets of activities from Table 6.1, following approximately the order listed. Leave out sets of activities whose benefits for your project do not justify their costs or time durations.

Determining the functional and/or operational profiles is often the SRE activity that takes the most cost and time. Reduction in cost and time can be obtained by limiting the number of elements in the profile [Musa93] and by accepting less precision.

One difficulty with small projects is the likelihood of only a small sample of failures occurring during reliability growth testing. You can still track testing progress, but not as precisely as you might wish.

6.3 SRE During the Feasibility and Requirements Phase

Although the makeup of this phase may vary across projects, this section describes the SRE activities, outlined in Fig. 6.1, that are conducted during the feasibility and requirements stages. The feasibility stage involves product concept development. The requirements stage involves the development of detailed product requirements and a plan to develop the product.

6.3.1 Feasibility stage

The output of the feasibility stage is either a feasibility report, product prospectus, or a business plan. This defines a certain market and explores the potential of the product to meet that market. It also assesses the capability of the organization to produce the product for the market in the time frame needed, at a price and level of reliability desired by the customer relative to their competitors. For government contract work the output consists of a response to a request for proposal (RFP), which contains many of the items described above.

6.3.1.1 Determine functional profile. To determine the *functional profile,* you must first establish the set of functions for the product. Functions are characterized in terms of *both* the tasks performed and the environmental factors that influence processing. For example, functions of a telephone PBX system would include the types of telephone calls placed and received by subclasses of users (e.g., managers, secretaries, and engineers). In addition, they would also contain information on the environment, such as the types of telephone sets configured on

the system and call features (for example, call forwarding) configured on the sets by each class of user.

In the case of both functional and operational profiles, you can account for *criticality* of functions or operations by weighting them.

Requirements are sometimes developed in great detail, so that in effect they incorporate some of the system architecture and actually specify the operations performed by the system. In that case, you can determine an operational profile directly in this stage, rather than after design and implementation.

Quality function deployment (QFD) is a useful method of working with your customers to identify functions that satisfy their needs. Having them ask, "How often do I visualize that need occurring?" and "How costly is it to me when that need is not satisfied?" helps in defining a functional profile and in defining classes of failures. Having a customer answer these questions also helps them rank their needs in completing the QFD process. For more detailed information on the functional profile see Chap. 5.

6.3.1.2 Define and classify failures. You should define *failure* from your customer's perspective. Start with similar products with the same customer base as your new product and determine the customer's failure experiences with them. Distinguish software failures from hardware and procedural failures.

A good practice is to group identified software failures into a few severity classes (say, three or four). This grouping should be done from the viewpoint of the effect of failures on the customer's ability to conduct business. The effect can be measured in terms of cost, service degradation, or safety. Table 6.2 gives an example based on service degradation for a telephone switching system.

There is a trade-off here. More classes require more effort later in the life cycle in collecting and analyzing data for each class, whereas fewer classes give a coarser resolution of the effect on the customer.

6.3.1.3 Identify customer reliability needs. You identify customer reliability needs at a high level in this stage. A small team conducts the assessment while defining other aspects of the product (e.g., feature set content, capacity, and performance capabilities). A recommended team consists of a system engineer, a system architect, a marketing (product planning) person, a customer representative (marketing may serve as a customer surrogate), a reliability analyst, and a system tester. The marketing person is a likely candidate to lead this team.

Establish who (other suppliers) and what (alternative products) your competitors are and assess their reliability capabilities. A good way of defining the level of reliability required is to relate the product to an existing product or set of products having the same customer base. For

TABLE 6.2 Severity Classification Based on Service Degradation

Severity classification	Definition	Example
Catastrophic	Entire system failed, no functionality left	No one can get dial tone
Severe	A high-priority customer feature is not working	You can make local but not long-distance calls
Significant	Customers must change how they use the system	You can't use your calling card in a pay phone but must read number to operator
Minor	Problem is not noticeable by customers	Some maintenance functions are not currently performed

example, relating a new product to an existing switching system product, a reliability assessment might state "product X should have a reliability comparable to switching system Y used in the customer central office."

As a minimum, determine an approximate acceptable failure intensity for each severity class. Typical acceptable values of failure intensity can range from one failure per 100 hours (low-severity failures for routine business applications) through one failure per billion hours (air traffic control or nuclear power plant shutdown). This approach was used within AT&T to establish the reliability of a large network system which included 337 thousand new-end changed source lines (KNCSL) of new software reused from previous products and of a PBX system with over 1100 thousand lines of code (KLOC).

6.3.2 Requirements stage

The requirements stage involves preparing a detailed requirements specification for the product and laying out a development plan. The *requirements specification* expands the needs and high-level features defined during the feasibility stage. It includes attributes such as product reliability, availability, performance, and capacity. A requirements specification must have a section defining the reliability requirements of the product. The *development plan* outlines the resources, costs, and schedules needed to develop the product. You determine resources, costs, and schedules from the product requirements specification, *including* the reliability specifications. The development plan should include adequate resources and schedule for reliability-related activities. These activities include reliability training of staff (Sec. 6.7.1) and particular reliability activities conducted during each phase of the product life cycle (Secs. 6.3 through 6.6).

The reliability assessment started in the feasibility stage is refined in this stage in forming your reliability requirements. You establish specific quantified reliability objectives. These are based on the functional profile and failure definitions you have determined and that you continue to refine. For example, for a PBX product used in a telemarketing application, you might specify that a software failure that could cause a system outage should occur no more than once every three months. When such an outage occurs, it should be recoverable in less than 30 minutes.

In this stage, the system engineer would generally serve as team leader of the reliability assessment team formed in the feasibility stage. You should also seriously consider adding a field support person to the team.

6.3.2.1 Conduct trade-off studies. A major cost in developing your product is associated with testing in general and with reliability growth testing in particular. For some product developments, testing accounts for over half of the development cost. Trade-off studies help you in setting objectives for reliability, cost, and delivery date. Investigate the balance among these attributes and functionality with respect to reliability growth testing and current software engineering technology.

Reliability and functionality. The number of functions is correlated with the number of lines of code developed, total faults introduced, and hence failure intensity. Thus, increasing functionality generally decreases reliability. Increased levels of reliability generally equate to increased levels of testing, which means increased time and cost.

Reliability, cost, and delivery date. Increasing the failure intensity objective (and hence reducing reliability) reduces reliability growth testing time and cost but increases the cost associated with field failures. Cost associated with field failures includes cost to the *supplier* in terms of repair, field support service, and potential lost future business. In addition, there is cost to the *customer.* For simplicity, we assume that the field failure cost increases linearly with the number of failures, although this assumption is probably conservative.

You can plot the total life-cycle cost (defined here to be the sum of the cost of doing reliability testing and field failure cost) versus the failure intensity objective (left-hand plot of Fig. 6.2). There is an optimum value of failure intensity objective for which the life-cycle cost is minimum. As field failure cost will vary with the failure severity class, individual trade-off studies can be done for each. Likewise, you can plot the time to do reliability growth testing versus the failure intensity objective (right-hand plot of Fig. 6.2) to determine which delivery date would correspond to a specified objective. The SRE tools discussed in App. A will produce plots like these.

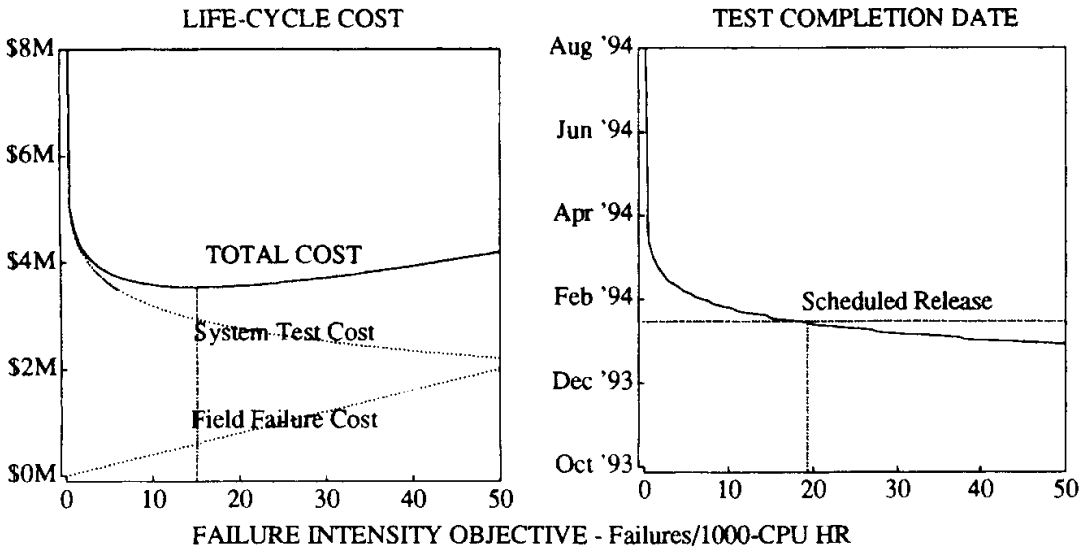

Figure 6.2 Cost and release date trade-offs.

Modeling to support trade-off studies. Currently, parameters of some reliability growth models can be predicted from product and development process characteristics. Product characteristics include such items as newly developed lines of code and lines of reused code. Process characteristics depend on the development environment and development organization. Determine them for the same or a similar situation. They include such quantities as requirements volatility, thoroughness of design documentation, and programmer experience level. Most process characteristics are incorporated in models through their effect on fault density.

Chapter 7 of [Musa87] provides details on conducting trade-off studies using one such model. Examples of their application and details on the model employed in such studies are also provided. Since these studies are performed early in the product life cycle, be aware that there can be substantial errors in absolute results, although relative results are usually valid.

6.3.2.2 Set reliability objectives. Separate reliability objectives are established for each failure category. Usually, system reliability objectives are set first (chap. 4 in [Ever90] and secs. 3.7.1 and 4.7.1 in [SAE90]) and then allocated between hardware and software. Factors that influence setting software reliability objectives include:

- Explicitly stated reliability requirements from a request for proposal or standards document
- Reliability performance of and customer satisfaction with previous releases or similar products

- Capabilities of competition (including both direct competitors and indirect ones such as the manual methods the product is intended to displace)
- Trade-offs with other characteristics such as performance, delivery date, and cost
- Warranty considerations
- Potential of reusing highly reliable components from other software systems
- Technology capabilities and constraints (e.g., use of fault-tolerant techniques)

For example, one project set reliability objectives on a particular category of failures based on customer experiences with existing similar products. Another project set reliability objectives for its new products by relating customer acceptance of previous products and failure rates observed during their system test.

As another example, a large-scale switching system has established objectives on software asserts, audits, peripheral processor interrupts, and system interrupts, based on the number of telephone calls originated. The project combines these measures to compute a stability index that also has a particular objective associated with it. The stability index objective is based on measures from previous releases of the switching system software and from customer satisfaction with these releases. See, for example, [Berg89] and [DiMa91].

Effect on architecture. Reliability requirements may strongly influence the architecture adopted for a product. In turn, changes to architecture may dramatically affect reliability. During this phase of the life cycle and continuing into the design and implementation phase, the architecture evolves through successive iterations. There may be successive reviews in this iterative process referred to as *discovery* reviews. The reliability assessment conducted during each of the stages in this phase provides valuable information for these reviews. The reliability analyst should participate in each of these reviews.

Availability. Availability is an important consideration in setting reliability objectives. It depends on how much time elapses after a failure before service can be restored. For example, for one product, the category of failures for which reliability objectives were set was those failures that required a reboot of the system to restore service. The particular reliability objectives were satisfactory only if the reboot time was under 15 minutes. Consequently, the project included an objective on the reboot time as part of the reliability objective.

6.4 SRE During Design and Implementation Phase

Although the structure of this phase may vary across projects, this section describes the SRE activities, outlined in Fig. 6.1, that are conducted during the design and implementation stages.

6.4.1 Design stage

The design stage involves translating a requirements specification into a design of the software product. During the implementation stage, the design is used to implement the code for the software programs.

The *system architecture,* expressing the system concept in terms of hardware and software components and the interfaces among them and with the external environment, is completed during this stage. As described in Sec. 6.3, the architecture evolves through successive iterations. A reliability analysis can assess whether the successive iterations of the architecture will satisfy reliability and availability requirements, as will be discussed in subsequent sections.

6.4.1.1 Allocate reliability among components. While defining the architecture, consider alternative ways of dividing the system into components while achieving the overall reliability objective. Various factors should be considered when you divide the system into components (see [Musa87], pp. 85–88). These factors include the physical nature of the system, the nature of previously collected data (for example, what sets of similar elements of known reliabilities exist), the need to track a particular component for project management purposes, and the amount of effort required for data collection. [Fran93] and [Pant91] describe component analysis experiences.

To determine the reliabilities required for the components, first make a trial allocation of the reliabilities. Next, calculate the system reliability using the trial allocation. Adjust the allocation so that the system reliability requirement is met, along with approximate equality of development time, difficulty, and risk for the components and minimum total system cost.

6.4.1.2 Engineer to meet reliability objectives. There are several methods for engineering the product to meet reliability objectives.

Plan recovery strategies. Many software failures result from transient environmental conditions such that if execution of the software is simply retried, the failure will not repeat. However, before execution is retried, some attempt should be made to repair possibly damaged data. Also, execution needs to be restarted from some known (possibly previ-

ously "checkpointed") place of execution. Finally, mechanisms should be in place to determine when a failure has occurred and to prevent further execution so as to contain damage to program data. [Lee90] discusses techniques of failure detection, damage confinement, and failure recovery.

Use redundant software elements. Redundant software elements can increase reliability only if they are not exactly the same copies. One approach is to have different groups develop them independently [Aviz85]. However, you probably can't achieve completely independent development in practice. Hence, the possible improvement is limited. Therefore, developers have used redundant software only in ultrareliable systems (typically, systems with failure intensity objectives less than 10^{-6} failure/CPU hr). Chapter 14 will discuss fault-tolerant software reliability engineering techniques in detail.

Identify high-risk areas. Two techniques used in safety-critical and ultrareliable systems design that are beginning to find their way into software design are FMEA and fault tree analysis. FMEA (failure modes and effects analysis) is a bottom-up technique that starts by defining failure modes of the components and then assesses how they can cause systemwide failures. Fault tree analysis (see Chap. 15 for details) is a top-down technique that starts with failures at the system level and successively decomposes and relates these failures to failures at the subsystem and component levels. Although FMEA and fault tree analysis are not extensively used for software in general, these techniques are finding applications in safety-critical software (e.g., see [Leve86]).

6.4.1.3 Focus resources based on functional profile.

The functional profile can help you, as a developer, focus on what is really important from the customer's standpoint. The information it provides on the frequency of use and criticality of different functions can help weigh design alternatives. For example, a designer might opt for a simpler manual recovery design for a condition that occurs infrequently rather than for a more complex automated recovery design.

6.4.2 Implementation stage

6.4.2.1 Focus resources based on functional profile.

The functional profile can also help allocate effort during the implementation stage, based on the relative usage and criticality of different functions. For example, you might expend more inspection effort or do more thorough testing for high-usage than low-usage functions. Similarly, the functional profile can provide helpful guidance for ordering the time periods scheduled for developing the functions (highest use and criticality first).

6.4.2.2 Manage fault introduction and propagation. Since faults are the underlying cause of failures in software, controlling the number of faults introduced in each development step and the number of faults that propagate undetected to the next development step is important in managing product reliability. Although we discuss the topic in this phase, you must manage faults across *all phases* of the life cycle. Many development practices affect fault management. A few of the more important ones are as follows:

- *Practicing a development methodology.* Using a common approach in translating high-level design into code, and in documenting it, is particularly important for larger projects. It facilitates good communication between project team members, helping reduce introduction of faults into the software.
- *Constructing modular systems.* A modular system consists of well-defined, simple, and independent parts interacting through well-defined interfaces. Small, simple modules are easier for designers and programmers to build and hence less prone to faults being introduced through human error. Also, modular designs are more maintainable and hence decrease the chances that detected faults are incorrectly repaired. [Ever90] provides a list of criteria for decomposing systems into manageable sets of modules.
- *Employing reuse.* Reuse of software components that have been well tested for an operational profile that is close to that expected for your system reduces fault introduction. The alternative is to develop new components. They will almost certainly have more faults than the properly reused ones.
- *Doing unit and integration testing.* Testing plays a major role in preventing faults from propagating to a next development step. Unit tests verify modules' functions as specified in their low-level (or module) designs. Integration tests verify that modules interact as specified in the high-level design (or architecture). [ISO87] requires design verification through tests, design reviews, and other measures. [ANSI86] requires that software projects maintain test plans and recommends the contents of these plans.
- *Conducting inspections and reviews.* You can review or inspect requirements, design documents, software code, user manuals, user training materials, and test documents. Both reviews and inspections use a small team of people to compare the output of a development step with what was specified on input to that step.
- *Controlling change.* Many failures result from change in the intermediate items produced as part of the completed product. Such inter-

mediate items include the code of software components, design and requirement specifications, test plans, and user documentation. To reduce the occurrence of such failures, you need to manage the various versions of such items and how they go together to produce the completed product. This is called *version control.* You must also maintain an orderly procedure for submitting, tracking, and completing requested changes to items (referred to as *change control).* Reducing the rate of change of requirements generally increases reliability. Projects often refer to requested changes as *modification requests* or MRs. Version and change control are together referred to as *configuration management.*

6.4.2.3 Measure reliability of acquired software. If you use acquired software in your product (i.e., software that was not developed or tested in previous releases of your product), you will have to decide if the reliability of the software should be certified for your application. Acquired software may be a commercial off-the-shelf program, reused modules (increasingly common for productivity reasons), or software delivered by a contractor.

A concern is that acquired software may have been tested and used under a different set of conditions than is expected for your application. Therefore its reliability may be different in your application than in other applications. In such a case, the reliability should be certified using the operational profile for your application. Certify acquired software as early as possible. This allows you time to take action if the software displays a level of reliability that is less than what you require.

Reliability demonstration testing. *Certifying the reliability of existing software can be done by using reliability demonstration testing. Select test cases at random according to the operational profile. Do not fix the underlying faults that cause failures. The test tells you after each failure whether the software should be accepted or rejected or whether you should continue testing.*

tool can be useful to construct the chart shown in Fig. 6.3 and to failure times on it (see App. A). The action you take depends on the on in which the failure point is located. Note that you continue testafter the first three failures in the example, but stop testing and it the software after the fourth. The test is based on sequential ling theory.

SRE During the System Test ield Trial Phase

m test and field trial activities certify that the software require- of the product are met and that the product is ready for general

use by the customer. The system test stage is critical, since it is the last stage in the development process where corrective action can be taken to improve the reliability of the product before release to the first customer.

The field trial stage validates the specifications and system test reliability in a customer's environment. It is the first use of the product at a customer site. The end of this stage marks the general availability of the product to all customers.

This section describes the SRE activities outlined in Fig. 6.1 that are conducted during this phase.

6.5.1 Determine operational profile

Determining the operational profile is an important part of test planning, which generally occurs substantially ahead of the system test stage proper. The operational profile is a set of operations and their associated probability of occurrence. Operations are characterized by considering both tasks performed *and* environmental factors that influence processing. An operation may represent a command, or a command with its parameters or input variables set within certain ranges, executing in a particular environment. For example, separate operations might represent the same command executing in normal and overload traffic environments. Test cases must be based on the operations as they are implemented in the system, rather than on the functions conceived during system definition.

Test planners usually determine the operational profile, with extensive collaboration from system engineers and designers. It is desirable to involve test planners in the feasibility and requirements phase. The

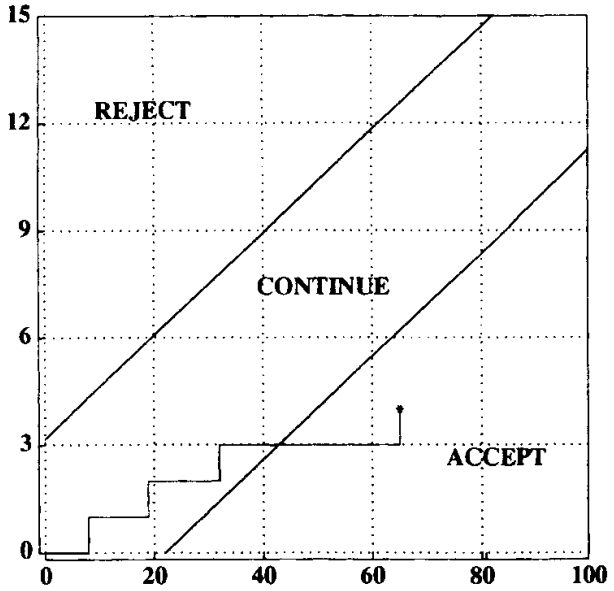

Figure 6.3 Reliability demonstration chart: failures versus execution time.

members of the team work together to specify the operational profile and to create test cases in conformance with the profile.

There are two main ways of deriving the operational profile used during testing. One is recording actual operation of a previous release or existing similar system. The other is estimating the operational profile, starting from the functional profile developed during the feasibility and requirements phase (Sec. 6.3). You often use some combination of both approaches.

Multiple operational profiles. It may be necessary to develop different operational profiles for different market segments with different applications. As an example, a point-of-sale system in a supermarket located in an urban area is likely to have a different usage profile than one in a suburban area. Also, you might fine-tune a special version of the system to meet a customer's high-reliability requirement for a particular set of functions. There are no technical limitations to the number of operational profiles you can define, but you are practically limited by the costs of developing and using them.

Ultrareliable operations. Catastrophic failures demand extra preventive effort. The frequency at which such failures occur is usually low (almost zero). The entire system need not be highly reliable; only the critical operations demand high reliability. For example, in a nuclear reactor power system, the alarm and shutdown operations must not fail. These operations (and only these operations) should have ultrareliable objectives. You may wish to establish a separate operational profile for them and test them separately.

For more detail on the determination of the operational profile, see Chap. 5.

6.5.2 System test stage

6.5.2.1 Conduct reliability growth testing. In reliability growth testing, system testers execute test cases in proportion to how often their corresponding operations occur in the field, as characterized by the operational profile. By mirroring customer use, reliability growth testing is likely to find the failures that cause the greatest customer dissatisfaction and will reflect the reliability the customer will experience.

You can select more than one test case per operation. First, select the operation randomly, in accord with its probability of occurrence. Note that frequently used operations tend to be selected first. Then select the test case randomly from those that belong to the operation, avoiding repetition. When failures occur, identify and remove the faults that are causing them. Repeated failures that occur during the period while failures are being resolved are not counted. When only the first occur-

rence of failures is counted, the failure intensity is based on the unknown faults remaining in the software. Failure intensity decreases with execution time (reliability growth).

The goal of reliability growth testing is to attain a level of confidence that the software product is being released with reliability that meets customers' needs. An example of many projects that have used SRE for this purpose is the case study described in [Ehrl90]. Reliability growth testing is usually the last step in system test. However, you need to allow enough execution time to demonstrate that the reliability objective is met.

Test automation. Investing some effort in automating the reliability growth testing process will often pay off. You can usually automate test selection and failure identification and recording. There are some failures you must identify manually, because you cannot specify their symptoms in advance. If automatically identified failures represent a constant proportion of all failures, you can rely on automatic identification and apply a correction factor.

Related types of testing. Certain special types of testing often meet the criteria for reliability growth testing. You may be able to combine failure data from these tests with reliability growth testing data to yield better reliability estimates.

- *Regression testing* ensures that old functions continue to work in the face of changes such as repair of problems and incorporation of new functions. Using the operational profile to drive regression testing is a particularly efficient way to accomplish this function.
- *Feature testing* verifies that the features/functions of a system are present and work as specified. It usually proceeds sequentially by functions, concentrating on one function and then another. Testers at present ordinarily don't use the operational profile to select or execute feature test cases. Thus, the rate at which failures occur does not reflect the rate at which the customer would see failures. Combining such data with reliability growth testing data is not advisable.

  However, with some effort, you can select test cases randomly in proportion to the frequencies specified by the operational profile, as noted at the start of this section. You can then use failure execution time data with existing software reliability models to estimate reliability. For more information see Chap. 5. Feature testing based on an operational profile is often more effective, because features are tested with user operations. This has been done in International DEFINITY [Abra92].

- *Performance and load testing* not only locates load/stress points at which the system fails to meet objectives, but also certifies that the software satisfies certain objectives regarding response time, throughput rates, start-up time, and capacity. Use the operational profile to drive performance and load testing so that it reflects customer usage. Since there is often a tendency to focus on peak-period usage, and systems behave differently under different load conditions, care must be taken to test at different levels if you wish to include the failure data in your reliability estimates.

Data collection. You will need to collect during system test the data shown in Table 6.3. The data are classified by the SRE activities you expect to perform. Include answers to the following questions in your planning:

1. Who will gather the data?
2. How will the data be collected?
3. What tools will be needed to support the data collection?
4. Who will analyze and interpret the data?

Measuring execution time. The use of SRE models during system test assumes that you can measure the execution time when failures occur. CPU time is a direct measure of execution time.

TABLE 6.3 Data to Collect in System Test

Type of data to be collected	SRE activity
1. Execution time of failures or another measure that can relate to execution time	Track testing progress, project additional testing needed, certify reliability objectives are met
2. Severity classification of failures	
3. Resource usage data • Tester time for failure identification • Developer time for failure resolution • Computer time for failure identification and resolution	Project additional calendar time needed for test
4. Number of faults found 5. Developed (new and modified) code in source instructions 6. Number of new faults spawned during failure resolution 7. Average instruction execution rate of processor 8. Total code in object instructions	Conduct trade-off studies (cost and schedule with reliability) for next generic or a new product with similar software and development characteristics

Several tools exist to measure the CPU time to a failure for a UNIX® application. UNIX® produces an entry in a process accounting file whenever a process is terminated, which can be extracted using the UNIX® "acctcom" administrative command. You can write a simple program to process the accounting file and record the execution time and calendar time when a failure occurred in the correct format for the software reliability tool.

Approximating execution time. CPU time is the preferred way of measuring execution time, but sometimes you cannot easily obtain it. In that case, you can estimate execution time by a variety of other means. You can use any measurement that is correlated with execution time. Examples are clock time, weighted clock time, number of commands executed [Ehrl90], telephone calls processed [Harr89], and interrupts processed. If the software fully utilizes its processor and is operating continuously through the time period, then execution time is equivalent to clock time. For weighted clock time, you can approximate execution time by the product of clock time and the system utilization.

6.5.2.2 Track testing progress and certify that reliability objectives are met. During reliability growth testing, failure data is collected and a software reliability tool (App. A) is used to track testing progress and project additional testing needed. Based on progress, management can make any necessary adjustments in resources and schedule as system testing continues. If you estimate reliability on a per-component basis, you can identify components with reliability problems for particular attention. When the current failure intensity reaches the failure intensity objective, you can certify that the reliability objective is met. You may need to do this for each severity class.

Adjusting for incremental delivery to system test. Software reliability models assume a stable program executing in a constant environment. However, software may change because of requirements changes or integration of parts during development. There are three approaches to handling software evolution during system test to get good reliability estimates:

1. Ignore change, and let the model adapt for itself. This is the best solution when the software is changing slowly in a continuous fashion and there is a final period of stability.
2. View the software change in terms of the addition or removal of independent components. This is best when you have a small number of changes, each an independent component.
3. View changes as occurring throughout the program. Here you adjust the failure times to what they would have been if the complete software had been present at the start of test. This is the best approach

when you have many changes of medium size, and it can be handled automatically by SRE Toolkit (see App. A and the Data and Tool Disk).

A detailed description of the three approaches to adjusting for incremental delivery of software during system test is given in [Musa87].

6.5.3 Field trial stage

When system testing is complete, software moves to the next stage, field trial. Field trial is sometimes referred to as *beta test* or *first office application* (FOA). It is advantageous if the field trial location has an operational profile that is close to the main-line usage of the product. You should have a field trial plan that includes failure recording procedures.

6.5.3.1 Certify that reliability objectives are met. During field trial, you collect failure data from the field site. You use field failure data and a software reliability tool to measure the reliability of the product in the field. Then you compare this with the reliability of the product measured at the end of system test.

Several factors can cause the field trial reliability to differ from the system test reliability:

- The definition of what the customer perceives as a failure is different from the definition used in testing the product.
- Inaccurate data collection during system test and/or field trial.
- The field and test operational profiles differ, the test environment not accurately reflecting field conditions.

Use of reliability growth modeling. If you resolve failures during the field trial and count only the first occurrence of each failure, you can apply a reliability growth model. The combined failure data from system test and from field trials can be concatenated, provided that failure definitions and the operational profile remain the same.

Example 6.1 Figure 6.4 is an example of the shape of the reliability growth model fitted to combined stability test (reliability growth test) and beta test failure data. The results in this example show that more failures were experienced during beta test than were expected at the end of system test [Ehrl90]. A discontinuity in the slope of the measured failure behavior is apparent. Based on stability test data, one additional failure had been expected during beta test. However, 16 additional failures occurred. Analysis indicated that the additional failures occurred because the operational profile used during stability test differed significantly from the beta test operational profile. The profile used in stability test did not include start-up operations such as database provisioning. Thus, behavior like that experienced can be a valuable warning sign that your testing is not realistic and needs to be modified.

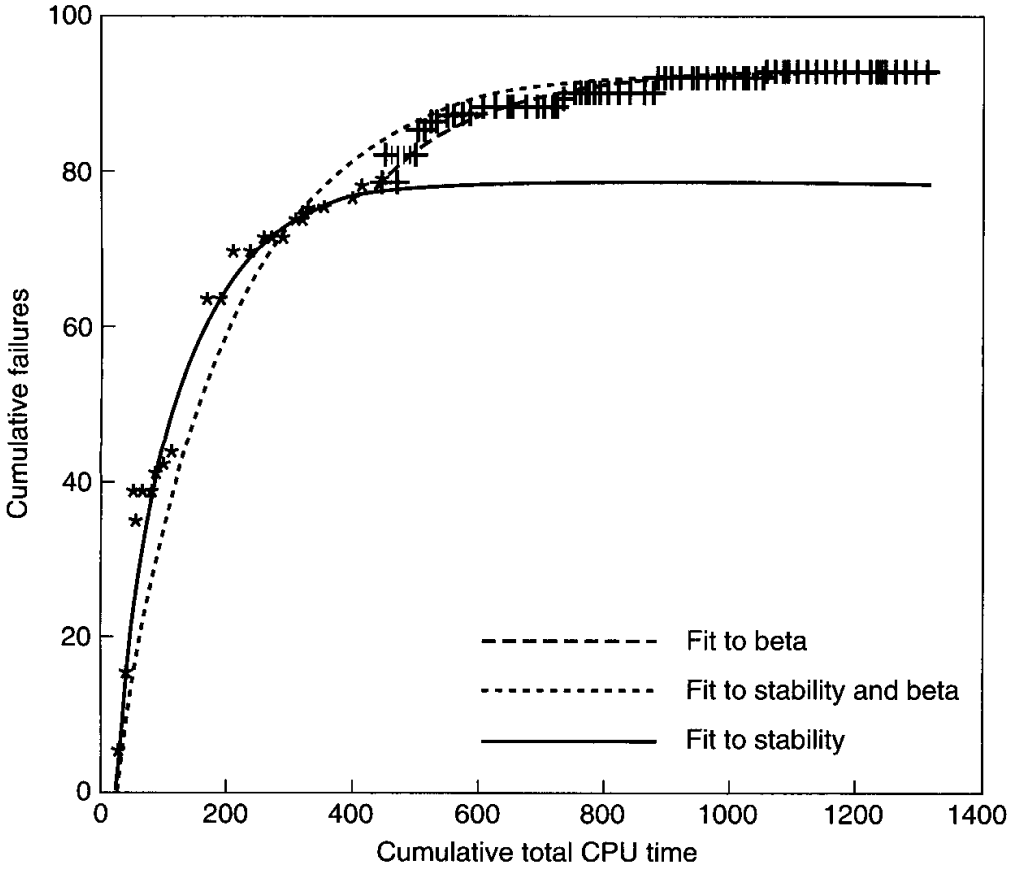

Figure 6.4 Measurements. System T stability (08/02/88–11/27/88) and beta test (12/01/88–03/21/89).

6.6 SRE During the Postdelivery and Maintenance Phase

During the postdelivery and maintenance phase, you deploy the product to your customers. Maintenance consists of removing the faults associated with failures reported by customers. This section describes the SRE activities that are conducted during this phase.

6.6.1 Project postrelease staff needs

You can use reliability models to project staff needs following the release of a software product. This includes:

1. The customer's operations staff to support service recovery following failures
2. The supplier's staff to handle customer-reported failures
3. The supplier's software development staff to locate and remove faults associated with customer-reported failures

When failures are not resolved during operations, constant reliability models (models with constant failure intensity) are used to project

items 1 and 2 for a given release. This is the situation between maintenance releases of delivered software products when no fixes or patches are introduced in the software. Less severe faults are generally not corrected between maintenance releases. When severe failures are resolved, use reliability growth models to project item 3.

6.6.2 Monitor field reliability versus objectives

If your organization has operations responsibility, you will find software reliability measurements useful for monitoring the reliability of the operating software. If not, you should attempt to get these data from your customer as an index of customer satisfaction. Failure intensity is approximately constant for a given release. However, there may be a period of reliability growth just after installation of a new release due to field fixes [Chri88].

Several projects at AT&T and elsewhere have reported on their experiences in analyzing failure data collected from field sites. The 5ESS U.S. telephone switch release 5E6 used software reliability to track product reliability after release to the field sites. An analysis of the data collected in the first month of operations showed that 5E6 met its projected reliability goals. Further analysis showed that 5E6 was better than the previous release after one year of operation. Field analysis identified one particularly troublesome failure, which received special attention during the development of the next release.

5ESS International collected failure intensity data from customer locations after release for four different time periods [DiMa91, Pant91]. They used this data to validate that the estimated failure intensity at the end of verification testing matched what was being observed at customer sites. The present failure intensity in the field varied about 25 percent around the corresponding failure intensity measured at the end of verification. However, the average failure intensity (total failures divided by the total execution hours) for each of the four time periods was 15 percent lower than the average failure intensity for the last three verification loads. Discrepancies were attributed to differences in the operational profiles used during verification testing and the customer's environment. The project used failure intensity and other metrics to assess the quality of the product and the development process [DiMa91].

If you observe differences in reliability in the customer's environment from what was predicted by system test, you should consider the same possible causes as were listed for the field trial stage in Sec. 6.5.3.1.

If reliability differs because the field and test operational profiles do not match, you need to find the source of the mismatch and take appropriate corrective action. The mismatch may be either in tasks or envi-

ronmental factors. As an example of the latter, simulators used in test to replace hardware components may not have faithfully reflected the operation of the replaced components. If the difference is substantial, you may wish to use an updated profile for the next release. Example 6.1 shows how differences in the operational profile between system test and beta test can affect measured reliability [Ehrl90].

Collecting failure data. To do software reliability estimation during operation, collect failure data that is tied to the execution time of the software, just as you did during reliability growth testing. Thus you need to determine total execution time across many customer sites. In field operations, more people will be involved, and you will need to carefully plan your data collection. For more information, see Chap. 11.

Some users may not report a given failure if they are not certain that a failure occurred or if they have focused on quick recovery rather than reporting. Thus, there is a need to motivate users to report failures and provide feedback so they will continue to do so.

For a true measure of field failure intensity, you need information about *all* failures. For other purposes, information about just the first occurrence of each failure (as provided by modification requests or MRs) will suffice. For example, MRs can be used to size the programming staff needed to remove faults. Collecting MRs is less expensive than manually collecting all failure occurrences. You may choose from several potential sources for obtaining failure data, but understand that each source has its advantages and limitations.

Using data from trouble tracking systems. To collect data on failures, many projects use a trouble-tracking system. In a typical operation, when the customer at a field site encounters a failure, they call in a trouble report. A "TR" (trouble report) number is assigned and information about the failure is entered into the system. The field support person assigned to the failure categorizes it (e.g., hardware, operator, software) and assigns a severity class.

The severity assigned depends on

- The extent to which the problem degrades system operation
- The lack of a work-around
- The customer's perception of the failure

If the failure is judged to be a new one, the field support person opens a modification request (MR). Although the same failure is likely to be reported from more than one site, resulting in multiple TRs, only one MR is (should be) opened. Thus, the number of MRs shows the *first occurrences* of failures, while TR numbers include repetitions of the same failures at different or even the same sites. An estimate of failure

intensity based on first occurrences of failures will yield an estimate of the failure intensity that will exist when the faults causing the reported failures have been removed.

One cannot assume there will be a TR for *every* occurrence of a failure in the field because some may not be reported. In practice, then, the number of TRs will fall *between* the number of first-occurrence failures and the total number of failures occurring in the field.

Talking directly to users. The operations staff at the field sites may maintain failure logs. Talking with the users and examining these logs can provide a true picture of the failures occurring. The discussion may clear up misunderstandings, for example, when the number of TRs is low, yet the customer voices dissatisfaction. This might occur if a project organization has been unresponsive to field problems in the past. To reduce the expense of this approach, sample just a few sites and adjust the results to estimate the total number of failures that occurred.

Automated failure data collection. The problem with TR and MR systems is that they were not designed to collect failure data. Some recent practitioners have reported that the practical extension of SRE to software systems' field usage requires instrumenting data collection so that the burden of data collection does not degrade the data. Under such an arrangement, field systems monitor their own health and then report data back to the development machine for analysis and interpretation.

6.6.3 Track customer satisfaction

Tracking field reliability in relation to objectives is necessary but not sufficient by itself. Select a sample of customer sites and survey their level of satisfaction with product reliability. You may be meeting your objectives but your customer may not be satisfied. Dissatisfaction may be due to inappropriate objectives being set or to other factors appearing in their use of the product. If there is dissatisfaction, you will want to follow up by modifying the objectives or by making any necessary field support service and product changes.

6.6.4 Time new feature introduction by monitoring reliability

Changes to add new functionality to a system will also likely add new defects, causing the failure intensity to rise. If the addition of new features is or can be separated from the removal of previous faults, a field operations manager may wish to use discretion in deciding when the new features are installed. The failure intensity of the system will exhibit a general stability about some value over the long term, but

with swings around this value when newly developed features are periodically incorporated. Failure intensity will increase just after you add new features. Periods in which fixes are installed to remove faults will exhibit decreasing failure intensity.

A field operations manager is often faced with conflicting demands. Some users want certain new features to be introduced as soon as possible. Other users, employing existing features, will insist that reliability be as high as possible. If these conflicts can be negotiated and a failure intensity objective established, the manager's job is then simplified. New features are introduced only when the failure intensity is below the objective.

The manager may use the amount of margin below the service objective as a guide to the size of the change to be permitted (see "Impact of Design Change," pp. 204–205, in [Musa87] for instruction on estimating this size). [Hami78] discusses an SRE application much like the one just described.

6.6.5 Guide product and process improvement with reliability measures

First, categorize field failures for analysis by their severities and frequencies experienced (if available). One might, for instance, analyze failures of severities 1 and 2 that exceed a certain frequency. Note that the frequency of a failure can vary from site to site. Once the selection of failures has been made, a root-cause analysis can begin on each underlying fault causing the failure to determine:

- Where and why the fault was introduced
- Why it escaped detection earlier in the development cycle
- What process change(s) are needed to reduce the probability that similar faults will be introduced in the future or at least increase the probability that such faults will be detected in the stage(s) where they are introduced

6.7 Getting Started with SRE

In the previous sections, you learned how and when to do SRE in your project. This section will help you:

1. Prepare your organization to do SRE.
2. Find information and support for doing SRE.
3. Do an SRE self-assessment.

6.7.1 Prepare your organization for SRE

6.7.1.1 SRE as a process. A key factor for a successful implementation of SRE is integrating SRE activities into an organization's software development process. First, you need to understand how SRE activities interrelate with development activities. These interrelationships are summarized in Fig. 6.1 and described in Secs. 6.3 through 6.6. Second, you need to understand *who* is to do *what* task or activity and *when* they are to do it. Table 6.4 summarizes the people who are involved in doing SRE and the life-cycle phase in which they are most likely involved. Large organizations may benefit in tailoring an SRE process for their organization.

6.7.1.2 Seven-step program. Two important ingredients for successfully introducing SRE into your organization are commitment by management and motivation of staff.

Table 6.5 is a sample seven-step implementation program that was followed with one organization.

The program first focuses on getting up-front management commitment by developing awareness of SRE and the benefits of doing it through a personal briefing. The level of commitment can be measured by the willingness of your management to expend resources and to review progress periodically in completing the steps in this program.

The program motivates your staff by providing adequate training and consulting support so that people understand and can do the jobs that are expected of them. Jump-start consulting ensures team members responsible for implementing SRE are properly supported in their efforts. It provides guidance, constructive critique, tutoring, and problem solving as required by project needs. Implementation might consist of one (or more) 4- to 6-month efforts with your team of one or more members who work part-time on the effort. It is important to designate one member of the team as a decision-making representative for your project.

Most efforts focus on first implementing SRE within system test to monitor and track reliability growth. On successfully completing this initial effort, you can add setting reliability objectives and defining an operational profile that better reflects customer usage of the product.

The program in Table 6.5 would be appropriate for organizations ranging from 50 to several thousand people. If your organization is smaller, parts of this program can be used to tailor a more appropriate program for you.

It is certainly possible to use outside experts instead of training your own personnel on a project if necessary. However, this is generally

TABLE 6.4 People Involved in Doing SRE

Job function	Software life-cycle phases			
	Feasibility and requirements	Design and implementation	System test and field trial	Postdelivery and maintenance
Product manager	X		X	
Project manager	X	X	X	
Development manager		X	X	
Reliability engineer	X	X	X	X
Systems analyst				
System engineer	X	X		
Software architect	X	X		
Software designer		X		
Programmer		X		
Test manager	X		X	
Quality assurance engineer		X	X	X
Test designer		X	X	
System tester	X		X	
Installation and operations manager	X			X
Users	X			X

TABLE 6.5 SRE Start-Up Program for Organizations

Step	Program element	Target audience	Purpose	When
1	SRE briefing	Executive management	SRE awareness; considerations for initial application.	0
2	Project manager overview course	Managers with project system engineering, development, or system test responsibility	SRE awareness for managers who will select potential projects for SRE implementation.	0 + 3 weeks
3	Two-hour follow-up meeting	Managers involved in step 2	Respond to questions and concerns, give advice. Identify one or more initial implementation efforts.	0 + 5 weeks
4	Project staff overview course	Staff of project using SRE	Prepare team members not needing in-depth SRE knowledge and skills.	0 + 8 weeks
5	SRE practitioner course	Staff directly involved in SRE implementation effort	Prepare team members needing in-depth SRE knowledge and skills.	0 + 8 weeks
6	SRE project planning course	Individual SRE implementation teams	Develop an action plan for initial implementation.	0 + 10 weeks
7	Jump-start consulting	Individual SRE implementation teams	Provide direct consulting to support SRE implementation effort.	After week 10

TABLE 6.6 Contacts for Support Resources

Items	Contact
AT&T SRE courses	AT&T Technical Education Center 1-800-TRAINER
AT&T SRE Toolkit	See enclosed Data and Tool Disk
IEEE SRE video (3 hrs)	IEEE Computer Society Press 800-272-6657, order number 1994AV
Software Reliability: Measurement, Prediction, Application by Musa, Okumoto, Iannino	McGraw-Hill 800-338-3987

undesirable because these experts will lack specific project knowledge; it will be much more difficult to integrate SRE with your development process; and you will not be preparing your people for the future.

Experience indicates that the development of the functional and operational profiles are the activities where you are most likely to need expert consultation.

6.7.2 Find more information or support

Table 6.6 lists several contacts for finding more information or support. The particular contacts shown may become out of date between updates to this text, but a reasonable amount of diligence should lead the reader to the desired information.

6.7.3 Do an SRE self-assessment

Use the statements in the following subsections to assess the conformance of your project or process with the best current practice of SRE. For each statement below that is completely true, score the number of points indicated at the beginning of the statement. The activities in each phase correspond to those listed in Fig. 6.1 and discussed in Secs. 6.3 through 6.6. Note that the activities can overlap into other phases as well. For any statement that is partially true, score 1 point. If you can justify why a particular statement is not applicable or not cost-effective for your project, you may take full credit for that statement and attach a written explanation of your justification.

Conformance level is based on the total points received as follows:

Total score	Conformance level
60 to 64	Fully conforms
45 to 59	Mostly conforms
25 to 44	Partially conforms
0 to 24	Does not conform

6.7.3.1 Feasibility and requirements phase _____ points

In writing your requirements specification, you:

(3) Determine functional profile (alternatively, may determine operational profile directly)

(3) Define and classify failures

(3) Identify customer reliability needs

(2) Conduct trade-off studies to help set objectives for reliability, delivery date, functionality, and cost

(3) Set reliability objectives

6.7.3.2 Design and implementation phase _____ points

As you define the architecture and design, you:

(2) Allocate system reliability to components so that the overall reliability objective is met

(2) Engineer to meet reliability objectives

During implementation, you:

(2) Focus resources based on functional profile

(2) Manage fault introduction and propagation

(2) Measure reliability of acquired software

6.7.3.3 System test and field trial phase _____ points

During product validation, you:

(3) Determine operational profile

(3) Conduct reliability testing, consistent with the operational profile

(3) Track testing progress, acting on differences between expected and achieved reliability

(2) Project additional testing needed

(5) Certify that reliability objectives are met before release

6.7.3.4 Postdelivery and maintenance phase _____ points

Before product release, you:

(2) Project postrelease staff needs

After product release, you:

(3) Monitor field reliability versus objectives, acting on differences

(2) Track customer satisfaction with reliability

(2) Guide new feature introduction by monitoring reliability

(3) Guide product and process improvement with reliability measures

6.7.3.5 Organizational and project preparation _____ points

Your organization and project demonstrate adequate preparation for SRE through:

(2) Visible management commitment

(2) Training of at least one local expert in SRE application

(2) An agreed-upon set of reliability metrics

(2) A supported set of SRE tools

(2) An agreed-upon and monitored set of expected SRE benefits

(2) Planning reliability-related activities, and providing adequate resources and schedule time for them

TOTAL: _____ points

6.8 Summary

SRE presents a life-cycle approach to managing software reliability. It provides a software engineer or manager the means to estimate and measure the rate of failure occurrence in software. The main focus of SRE is on how the customer will use the product in their environment. Software product usage is part of the reliability definition.

During the *product feasibility and requirements* phase, the functions the product will perform for each user, the frequencies of use of these functions, and criticality are defined. These functions establish the functional profile. Failures are defined and categorized from the product-user perspective. The reliability objectives for different uses of the product are established based on trade-off studies between functionality, reliability, cost, and schedule. Tools are used to determine these trade-offs.

During the *design and implementation* phase, the developer allocates reliability objectives between the hardware and software components. The functional profile helps focus development resources according to frequency of customer usage and criticality. Operations are characterized by considering both the tasks to be performed and the environmental factors that influence processing. Since most software applications are built from acquired software (software not developed or tested in previous releases of the product), the reliability of this software has to be certified. This certification can be done using a reliability demonstration testing tool.

During the *system test and field trial* phase, the focus is on ensuring that the completed software meets the reliability objectives as specified in the requirements. The functional profile is refined into an operational profile that is used to select operations during system test in accordance with the occurrence probability. The philosophy of operational profile testing is to test the operations the customer will most likely use, and time is not spent on testing little-used and noncritical operations. Failures are reported and faults removed. Tools use the failure data to determine the current failure intensity and estimate release date. The system is released when the failure intensity objective (software reliability objective) has been achieved. This avoids excessive time due to overtesting. The reliability objective for the product can be determined by combining the hardware reliability objective and the software reliability objective.

The practice of SRE continues during *postdelivery and maintenance.* Field reliability is monitored against established product objectives and customer satisfaction. This information can be used to improve the reliability of future product releases and to improve the quality of the development process.

Problems

6.1 Studies to date show that up-front SRE investment during the development cycle results in earlier delivery, increased productivity, lower maintenance cost, and satisfied customers.

a. What SRE activities are important to achieve this payback?
b. What benefits are achieved?
c. In what phase of the life cycle do these activities occur?

6.2 A software reliability program requires that the development cycle of the product be definable and measurable. Reliability cannot be assumed if it cannot be defined and measured. A successful metrics program critically depends upon the quality of the data that is needed as input.

a. What type of data is needed to establish the functionality of the product?
b. What type of data is needed to establish a reliability objective?
c. What type of data is needed during reliability growth test to certify that the reliability objectives of the system are met?
d. What type of data should be collected during reliability growth test to provide historical data needed to establish a reliability objective for the next release or for a new product similar to this product?

6.3 The current airline reservation system has been in existence for 3 years. During this time three major problems have occurred:

- *System performance.* Transaction processing during the busy period has not been acceptable. The response time is adequate when the system is lightly

loaded. It becomes a problem around 10 A.M. and remains a problem to around 3 P.M. Response time is particularly bad from 11:30 to 1:30.

- *System availability.* The system is unavailable because the data base fails and must be restarted.
- *Cost of new functions.* The reservation system is a dynamic one and is frequently updated to include new services. Travel reservation personnel are saying that the costs of implementing new features are excessive.

a. A new release for the software is now needed. Your job as a system engineer/analyst is to establish a functional profile and to negotiate with the customer on:

- Performance requirements
- Reliability requirements
- Cost

(1) What is your first step?
(2) What information is needed to determine the functional profile?
(3) Establish the definition of failures for this system.
(4) Categorize these failures into severity classifications.
(5) How would you establish the functionality and failure intensity objective for the new software?
(6) How does decreasing the number of features increase reliability of the product?
(7) Where in the life cycle of the new release should inspections be scheduled?

b. The software is now in the design stage. Your job as reliability engineer is to ensure that the reliability requirements for the software are met.
(1) How can the total reliability (hardware and software) of the airline reservation system be determined?
(2) To update the transaction database, a large amount of acquired software will be used. How can you determine if the reliability of the acquired software meets the reliability requirements for the system?

c. The software has completed the implementation stage and is entering the system test stage. Your job as lead system tester is to ensure that the reliability objectives of the software will be met on release.
(1) What information is needed to determine the operational profile?
(2) Failure execution time may be hard to measure. What other measure could you use? What will be the unit of the failure intensity objective?
(3) What type of information is important to the test manager?
(4) How will you track the progress of system test and determine when the software reliability objective is met?
(5) Often the entire software is not available for system test. Can you use SRE tools with incremental development?

d. The software is now in operation in the field. Your job as reliability engineer is to determine if the field reliability of the software differs from the reliability that was measured at the end of system test. If the reliability is different, what are the main factors that could have contributed to the difference?