SOFTWARE ENGINEERING

B.Tech. III Year I Sem. **Course Code: CS503PC**

UNIT- V

Risk management: Reactive vs Proactive Risk strategies, software risks, Risk identification, Risk projection, Risk refinement, RMMM, RMMM Plan.
Quality Management: Quality concepts, Software quality assurance, Software Reviews, Formal technical reviews, Statistical Software quality Assurance, Software reliability, The ISO 9000 quality standards.

TEXT BOOK:

Software engineering A practitioner's Approach, Roger S Pressman, sixth edition McGraw Hill International Edition. (Page Nos. 726 to 768)

RISK MANAGEMENT:

- Risk analysis and management are actions that help a software team to understand and **manage uncertainty**. **Managers, software engineers, and other stakeholders** participate in risk analysis and management.
- A risk is a potential problem—**it might happen, it might not**. But, regardless of the outcome, it's a really **good idea to identify it, assess** its probability of occurrence, estimate its impact, and establish a contingency plan.
- First step is "**risk identification**" Next, each risk is **analyzed**. Once this information is established, risks are **ranked**, by probability and impact. Finally, a **plan** is developed to manage those risks.
- A **risk mitigation, monitoring, and management (RMMM) plan** or a set of **risk information sheets** is produced. The risks that are analyzed and managed should be derived from thorough study of the people, the product, the process, and the project.

REACTIVE VS PROACTIVE RISK STRATEGIES:

The majority of software teams rely solely on reactive risk strategies. At best, a reactive strategy monitors the project for likely risks. More commonly, the software team does nothing about risks until something goes wrong. Then, the team flies into action in an attempt to correct the problem rapidly. This is often called a **fire-fighting mode**. When this fails, "crisis management" takes over and the project is in real jeopardy.

A more intelligent strategy for risk management is to be proactive. A proactive strategy begins long before technical work is initiated. Potential risks are identified, their probability and impact are assessed, and they are ranked by importance. Then, the software team establishes a plan for managing risk. The primary objective is to avoid risk, but because not all risks can be avoided, the team works to develop a contingency plan that will enable it to respond in a **controlled and effective** manner.

SOFTWARE RISKS:

Risk always involves two characteristics:

(1) **uncertainty**—the risk may or may not happen; that is, there are no 100 percent probable risks.

(2) **loss**—if the risk becomes a reality, unwanted consequences or losses will occur.

When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk. To accomplish this, different categories of risks are considered.

- **Project risks** threaten the project plan. That is, if project risks become real, it is likely that the project schedule will slip and that **costs** will increase. Project risks identify potential **budgetary**, **schedule**, personnel (**staffing** and organization), resource, **stakeholder**, and requirements problems and their impact on a software project. The **project complexity, size, and the degree of structural uncertainty** were also defined as project (and estimation) **risk factors**.
- **Technical risks** threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, **implementation may become difficult or impossible**. Technical risks identify potential d**esign, implementation, interface, verification, and maintenance** problems. In addition, specification ambiguity, technical uncertainty, technical obsolescence, and "leading-edge" technology are also **risk factors**.
- **Business risks** threaten the viability of the software to be built and often jeopardize the project or the product. Candidates for the **top five business risks** are (1) building an excellent product or system that no one really wants (**market risk**), (2) building a product that **no longer fits** into the overall business strategy for the company

(**strategic risk**), (3) building a product that the sales force **doesn't understand how to sell** (**sales risk**), (4) **losing the support of senior management** due to a change in focus or a change in people (**management risk**), and (5) **losing budgetary** or personnel commitment (**budget risks**).

- **Known risks** are those that can be **uncovered after careful evaluation** of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment).
- **Predictable risks** are **extrapolated from past project experience** (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced).
- **Unpredictable risks** can and do occur, but they are extremely **difficult to identify in advance**.

Seven Principles of Risk Management :

The Software Engineering Institute (SEI) (www.sei.cmu.edu) identifies seven principles that "provide a framework to accomplish effective risk management." They are:

1. **Maintain a global perspective**—view software risks within the context of a system in which it is a component and the business problem that it is intended to solve.
2. **Take a forward-looking view**—think about the risks that may arise in the future (e.g., due to changes in the software); establish contingency plans so that future events are manageable.
3. **Encourage open communication**—Encourage all stakeholders and users to suggest risks at any time.
4. **Integrate**—a consideration of risk must be integrated into the software process.
5. **Emphasize a continuous process**—the team must be vigilant throughout the software process, modifying identified risks as more information is known and adding new ones as better insight is achieved.
6. **Develop a shared product vision**—if all stakeholders share the same vision of the software, it is likely that better risk identification and assessment will occur.
7. **Encourage teamwork**—the talents, skills, and knowledge of all stakeholders should be pooled when risk management activities are conducted.

RISK IDENTIFICATION:

Risk identification is a **systematic attempt** to specify threats to the project plan (estimates, schedule, resource loading, etc.). By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.

There are two distinct types of risks are generic risks and product-specific risks.

- **Generic risks** are a potential threat to every software project.
- **Product-specific risks** can be identified only by those with a clear understanding of the **technology, the people, and the environment** that is specific to the software that is to be built. To identify product-specific risks, the project plan and the software statement of scope are examined, and an answer to the following **question** is developed: **"What special characteristics of this product may threaten our project plan?"**

The checklist can be used for risk identification and focuses on some subset of known and predictable risks in the following generic subcategories:

• **Product size**—risks associated with the overall size of the software to be built or modified.

•**Business impact**—risks associated with constraints imposed by management or the marketplace.

• **Stakeholder characteristics**—risks associated with the sophistication of the stakeholders and the developer's ability to communicate with stakeholders in a timely manner.

• **Process definition**—risks associated with the degree to which the software process has been defined and is followed by the development organization.

• **Development environment**—risks associated with the availability and quality of the tools to be used to build the product.

• **Technology to be built**—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.

• **Staff size and experience**—risks associated with the overall technical and project experience of the software engineers who will do the work.

Questions relevant to each of the topics can be answered for each software project. The answers to these questions allow you to estimate the impact of risk.

Assessing Overall Project Risk :

The questions are ordered by their relative importance to the success of a project.

1. Have top software and customer managers formally committed to support the project?
2. Are end users enthusiastically committed to the project and the system/ product to be built?
3. Are requirements fully understood by the software engineering team and its customers?
4. Have customers been involved fully in the definition of requirements?
5. Do end users have realistic expectations?
6. Is the project scope stable?
7. Does the software engineering team have the right mix of skills?
8. Are project requirements stable?
9. Does the project team have experience with the technology to be implemented?
10. Is the number of people on the project team adequate to do the job?
11. Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

If any one of these questions is answered negatively, mitigation, monitoring, and management steps should be instituted without fail. **The degree to which the project is at risk is directly proportional to the number of negative responses to these questions.**

Risk Components and Drivers:

The project manager identify the risk drivers that affect software risk components—performance, cost, support, and schedule. The risk components are defined in the following manner:

- *Performance risk*—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.
- *Cost risk*—the degree of uncertainty that the project budget will be maintained.
- *Support risk*—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.
- *Schedule risk*—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

FIGURE 28.1 Impact assessment. Source: [Boeh89]

Category		Performance	Support	Cost	Schedule
Catastrophic	1	Failure to meet the requirement would result in mission failure		Failure results in increased costs and schedule delays with expected values in excess of $500K	
	2	Significant degradation to nonachievement of technical performance	Nonresponsive or unsupportable software	Significant financial shortages, budget overrun likely	Unachievable IOC
Critical	1	Failure to meet the requirement would degrade system performance to a point where mission success is questionable		Failure results in operational delays and/or increased costs with expected value of $100K to $500K	
	2	Some reduction in technical performance	Minor delays in software modifications	Some shortage of financial resources, possible overruns	Possible slippage in IOC
Marginal	1	Failure to meet the requirement would result in degradation of secondary mission		Costs, impacts, and/or recoverable schedule slips with expected value of $1K to $100K	
	2	Minimal to small reduction in technical performance	Responsive software support	Sufficient financial resources	Realistic, achievable schedule
Negligible	1	Failure to meet the requirement would create inconvenience or nonoperational impact		Error results in minor cost and/or schedule impact with expected value of less than $1K	
	2	No reduction in technical performance	Easily supportable software	Possible budget underrun	Early achievable IOC

Note: (1) The potential consequence of undetected software errors or faults.
(2) The potential consequence if the desired outcome is not achieved.

The impact of each **risk driver** on the risk component is divided into one of four impact categories—**negligible, marginal, critical, or catastrophic**. Referring to Figure 28.1, a characterization of the potential consequences of **errors (rows labeled 1)** or a failure to achieve a desired **outcome (rows labeled 2)** are described. The impact category is chosen based on the characterization that best fits the description in the table.

RISK PROJECTION: (RISK ESTIMATION)

Risk projection, also called *risk estimation*, attempts to rate each risk in two ways—

(1) the likelihood or probability that the risk is real and

(2) the consequences of the problems associated with the risk.

Four risk projection steps:

1. Establish a scale that reflects the perceived likelihood of a risk.
2. Delineate the consequences of the risk.
3. Estimate the impact of the risk on the project and the product.
4. Assess the overall accuracy of the risk projection so that there will be no misunderstandings.

The intent of these steps is to consider risks in a manner that **leads to prioritization**. By prioritizing risks, you can allocate resources where they will have the most impact.

A risk table provides you with a simple technique for risk projection. A sample risk table is illustrated in Figure 28.2.

FIGURE 28.2 Sample risk table prior to sorting

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
Σ				

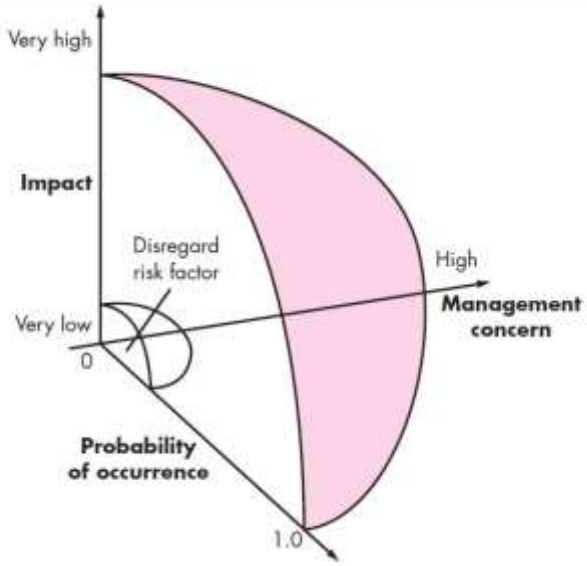
Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

You begin by listing all risks in the first column of the table. Each risk is categorized in the second column (e.g., PS implies a project size risk, BU implies a business risk). The probability of occurrence of each risk is entered in the next column of the table. The probability value for each risk can be estimated by team members individually. One way to accomplish this is to poll individual team members in round-robin fashion until their collective assessment of risk probability begins to converge.

Next, the impact of each risk is assessed. Once the first four columns of the risk table have been completed, the table is **sorted by probability and by impact**. High-probability, high-impact risks percolate to the top of the table, and low-probability risks drop to the bottom. **This accomplishes first-order risk prioritization**.

The *cutoff line* (drawn horizontally at some point in the table) implies that only risks that lie above the line will be given further attention. Risks that fall below the line are reevaluated to accomplish **second-order prioritization**. Referring to Figure 28.3, A risk factor that has a **high impact but a very low probability of occurrence should not absorb** a significant amount of management time. However, high-impact risks with moderate to high probability and low-impact risks with high probability should be carried forward into the risk analysis steps that follow. (e.g., a probability of 0.7 to 0.99 implies a highly probable risk).

FIGURE 28.3
Risk and management concern



Assessing Risk Impact :

Three factors - its nature, its scope, and its timing.

The nature of the risk indicates the problems that are likely if it occurs. For example, a poorly defined external interface to customer hardware (**a technical risk**) will stop early design and testing and will likely **lead to system integration problems** late in a project.

The scope of a risk combines the **severity** (just how serious is it?) with its overall distribution (how much of the project will be affected or how many stakeholders are harmed?).

Finally, **the timing of a risk** considers when and for **how long the impact will be felt**.

The risk analysis approach proposed by the U.S. Air Force, the following steps to determine the overall consequences of a risk:

(1) determine the average probability of occurrence value for each risk component;

(2) using Figure 28.1, determine the impact for each component based on the criteria shown, and

(3) complete the risk table and analyze the results as described in the preceding sections.

The overall risk exposure RE is determined using the following relationship :

$$RE = P \times C$$

where P is the probability of occurrence for a risk, and C is the cost to the project should the risk occur.

For example,

Risk probability. 80 percent (likely).

Risk cost. Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is $14.00, the overall cost (impact) to develop 18 components would be 18 x 100 x14 = $25,200.

Risk exposure. RE = 0.80 x 25,200 ~ $20,200.

Risk exposure can be **computed for each risk** in the risk table. The total risk exposure for all risks can provide a means for **adjusting the final cost estimate for a proj**ect. It can also be used to predict the probable **increase in staff resources** required at various points during the project schedule.

RISK REFINEMENT:

Risk Refinement represents the risk in *condition-transition-consequence* (CTC) format. That is, the risk is stated in the following form:

Given that <condition> then there is concern that (possibly) <consequence>.

Ex. **Given that** all reusable software components must conform to specific design standards and that some do not conform, **then there is concern that (possibly)** only 70 percent of the planned reusable modules may actually be integrated into the as-built system, resulting in the need to custom engineer the remaining 30 percent of components.

This general condition can be refined in the following manner:

Sub condition 1. Certain reusable components were developed by a third party with no knowledge of internal design standards.

Sub condition 2. The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

Sub condition 3. Certain reusable components have been implemented in a language that is not supported on the target environment.

The consequences associated with these refined subconditions remain the same (i.e., 30 percent of software components must be custom engineered), but the refinement helps to isolate the underlying risks and might lead to easier analysis and response.

RMMM: (RISK MITIGATION, MONITORING, AND MANAGEMENT)

The Goal of Risk analysis is to assist the project team in developing a strategy for dealing with risk. An effective strategy must consider three issues: **risk avoidance, risk monitoring, and risk management**.

RISK MITIGATION :

If a software team adopts a proactive approach to risk, **avoidance** is always the best strategy. This is achieved by developing a plan for ***risk mitigation***.

For example, assume that **high staff turnover** is noted as a project risk. Based on past history and management intuition, the likelihood of high turnover is estimated to be 0.70 (70 percent) and the impact is projected as critical. That is, **high staff turnover will have a critical impact on project cost and schedule**. To **mitigate** this risk, you would develop a strategy for **reducing turnover**.

Among the possible steps to be taken are:

- Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market).
- Mitigate those causes before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity.
- Organize project teams so that information about each development activity is widely dispersed.
- Define work product standards and establish mechanisms to be sure that all models and documents are developed in a timely manner.
- Conduct peer reviews of all work.
- Assign a backup staff member for every critical technologist.

RISK MONITORING:

The **project manager monitors** factors that may provide an indication of whether the **risk is becoming more or less** likely. In the case of high staff turnover, the general attitude of team members based on project pressures, the degree to which the team has jelled, interpersonal relationships among team members, potential problems with compensation and benefits, and the availability of jobs within the company and outside it are all monitored.

In addition to monitoring these factors, a project manager should monitor the effectiveness of risk mitigation steps.

RISK MANAGEMENT:

Risk management and contingency planning assumes that mitigation efforts have **failed** and that the risk has become a reality. For example, the project is well under way and a **number of people announce that they will be leaving**. If the **mitigation strategy has been followed**, **backup is available**, information is documented, and knowledge has been

dispersed across the team. In addition, we can temporarily refocus resources (and readjust the project schedule) to those functions that are fully staffed, enabling newcomers who must be added to the team to "get up to speed." Those individuals who are leaving are asked to stop all work and spend their last weeks in "**knowledge transfer mode**." This might include video-based knowledge capture, the development of "commentary documents" and/or meeting with other team members who will remain on the project.

THE RMMM PLAN:

FIGURE 28.4
Risk information sheet.
Source: [Wil97].

Risk information sheet

Risk ID: P02-4-32	Date: 5/9/09	Prob: 80%	Impact: high

Description:
Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Refinement/context:
Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards.
Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.

Mitigation/monitoring:
1. Contact third party to determine conformance with design standards.
2. Press for interface standards completion; consider component structure when deciding on interface protocol.
3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.

Management/contingency plan/trigger:
RE computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly.
Trigger: Mitigation steps unproductive as of 7/1/09.

Current status:
5/12/09: Mitigation steps initiated.

Originator: D. Gagne	Assigned: B. Laster

The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan. Each risk is documented individually using a ***risk information sheet*** **(RIS)**. In most cases, the RIS is maintained using a database system so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily. The format of the RIS is illustrated in Figure 28.4.

Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence. Risk monitoring is a project tracking activity with three primary objectives:

(1) to assess whether predicted risks do, in fact, occur.

(2) to ensure that risk aversion steps defined for the risk are being properly applied.

(3) to collect information that can be used for future risk analysis.

QUALITY MANAGEMENT:

A single goal of software engineering is to **produce high-quality software**. ***Software quality assurance*** **(SQA)** is an umbrella activity that is applied throughout the software process.

SQA encompasses

(1) a quality management approach,

(2) effective software engineering technology (methods and tools),

(3) formal technical reviews that are applied throughout the software process,

(4) a multi-tiered testing strategy,

(5) control of software documentation and the changes made to it,

(6) a procedure to ensure compliance with software development standards, and

(7) measurement and reporting mechanisms.

QUALITY CONCEPTS:

(1) ***Variation control*** is the heart of quality control. A manufacturer wants to minimize the variation among the products that are produced.

From one project to another, we want to **minimize the difference** between the predicted resources needed to complete a project and the actual resources used, including **staff, equipment, and calendar time**. We want to **minimize the number of defects** that are released to the field and variance in the **number of bugs is also minimized** from one release to another. (Our customers will likely be upset if the third release of a product has ten times as many defects as the previous release.)

(2) Quality:

Quality is defined as "a characteristic or attribute of something."— cyclomatic complexity, cohesion, number of function points, lines of code, and many others. Two kinds of quality may be encountered: quality of design and quality of conformance.

- ***Quality of design*** refers to the **characteristics** that designers specify for an item. The grade of materials, tolerances, and performance specifications all contribute to the quality of design. As higher-grade materials are used, tighter tolerances and greater levels of performance are specified, the design quality of a product increases, if the product is manufactured according to specifications. **In software development, quality of design encompasses requirements, specifications, and the design of the system**.
- ***Quality of conformance*** is **the degree** to which the design specifications are followed during manufacturing. Again, the greater the degree of conformance, the higher is the level of quality of conformance. In Software Engineering, Quality of conformance is an issue **focused primarily on implementation**.

User satisfaction = compliant product + good quality + delivery within budget and schedule

DeMarco states: "**A product's quality is a function of how much it changes the world for the better.**" This view of quality contends that if a software product provides substantial benefit to its end-users, they may be willing to tolerate occasional reliability or performance problems.

(3) Quality Control:

Variation control may be equated to quality control. ***Quality control*** **involves the series of inspections, reviews, and tests** used throughout the software process to ensure each work product meets the requirements placed upon it. Quality control includes a **feedback loop** to the process that created the work product. The **combination of measurement and feedback** allows us to tune the process when the work products created fail to meet their specifications. This approach views quality control as part of the manufacturing process.

Quality control activities may be fully automated, entirely manual, or a combination of automated tools and human interaction. A key concept of quality control is that all work products have defined, measurable specifications to which we may compare the output of each process. The **feedback loop is essential to minimize the defects produced**.

(4) Quality Assurance:

Quality assurance **consists of the auditing and reporting functions of management**. The goal of quality assurance is to provide management with the data necessary to be **informed about product quality**, thereby gaining insight and confidence that product quality is meeting its goals. If the data provided through quality assurance identify problems, it is **management's responsibility to address the problems** and apply the necessary resources to **resolve quality issues**.

(5) Cost of Quality:

The *cost of quality* includes all costs incurred in performing quality-related activities.

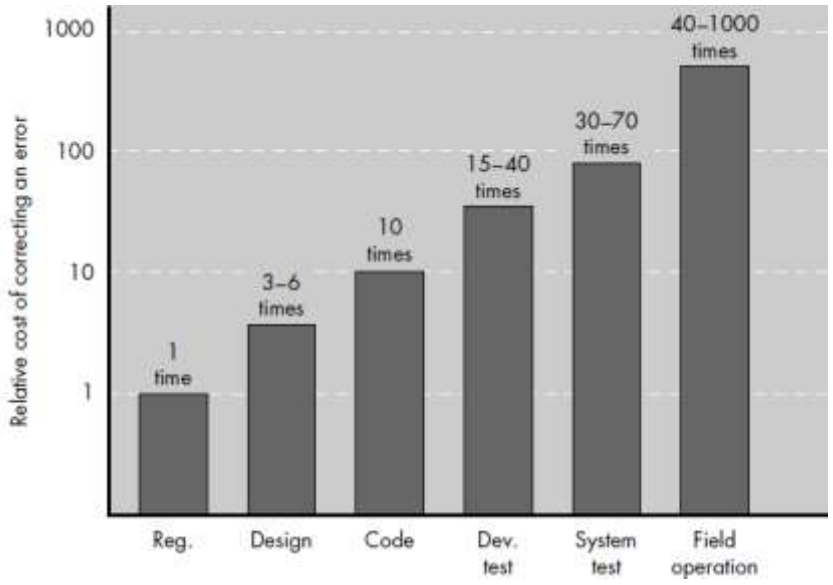
Once we have quality costs on a dollar basis, we have the necessary data to evaluate where the opportunities lie to improve our processes.

- *Quality costs* may be divided into costs associated with prevention, appraisal, and failure.
- *Prevention costs* include quality planning, formal technical reviews, test equipment, training.
- *Appraisal costs* include activities of in-process and inter process inspection, equipment calibration and maintenance, testing.
- *Failure costs* may be subdivided into internal failure costs and external failure costs.

- *Internal failure costs* are incurred when we detect a defect in our product prior to shipment. Internal failure costs include rework, repair, failure mode analysis.
- *External failure costs* are associated with defects found after the product has been shipped to the customer. Examples of external failure costs are complaint resolution, product return and replacement, help line support, warranty work.

As expected, the relative costs to find and repair a defect increase dramatically as we go from prevention to detection to internal failure to external failure costs.

FIGURE 8.1
Relative cost of correcting an error



A total of 7053 hours was spent inspecting 200,000 lines of code with the result that 3112 potential defects were prevented. Assuming a programmer cost of $40.00 per hour, the total cost of preventing 3112 defects was $282,120, or roughly $91.00 per defect.

Compare these numbers to the cost of defect removal once the product has been shipped to the customer. Suppose that there had been no inspections, but that programmers had been extra careful and only one defect per 1000 lines of code. That would mean that 200 defects.

SOFTWARE QUALITY ASSURANCE:(SQA)

Software quality **is defined as Conformance** to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

The definition serves to emphasize three important points:

1. Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.

2. Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.

3. A set of **implicit requirements** often goes unmentioned (e.g., the desire for ease of use and good maintainability). If software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspect.

SQA Activities:

Software quality assurance is composed of a variety of tasks associated with two different constituencies—

Software engineers who do technical work and Software engineers address quality by applying solid technical methods and measures, conducting formal technical reviews, and performing well-planned software testing.

SQA group that has responsibility for quality assurance planning, oversight, record keeping, analysis, and reporting. SQA group is to assist the software team in achieving a high quality end product.

The activities are performed by an independent SQA group that:

- **Prepares an SQA plan for a project.** The plan is developed during project planning and is reviewed by all interested parties. The plan identifies
  - evaluations to be performed
  - audits and reviews to be performed
  - standards that are applicable to the project
  - procedures for error reporting and tracking
  - documents to be produced by the SQA group
  - amount of feedback provided to the software project team
- **Participates in the development of the project's software process description.**

  The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g., ISO-9001), and other parts of the software project plan.
- **Reviews software engineering activities to verify compliance with the defined software process.**
- **Audits designated software work products to verify compliance with those defined as part of the software process.**
- **Ensures that deviations in software work and work products are documented and handled according to a documented procedure.**
- **Records any noncompliance and reports to senior management.**

SOFTWARE REVIEWS:

Reviews are applied at various points during software development and **serve to uncover errors and defects** that can then be removed. Software reviews "purify" the software engineering activities that we have called *analysis, design,* and *coding.*

A review is a way of using the diversity of a group of people to:

1. Point out needed improvements in the product of a **single person or team**;
2. Confirm those parts of a product in which improvement is either not desired or not needed;

An **informal meeting** around the coffee machine is a form of review, if technical problems are discussed. A **formal presentation** of software design to an audience of customers, management, and technical staff is also a form of review.

F*ormal technical review*, sometimes called a *walkthrough* or an *inspection.* A formal technical review is the most effective filter from a quality assurance standpoint.

Cost Impact of Software Defects:

The primary objective of formal technical reviews is to find errors during the process so that they do not become defects/faults after release of the software. The obvious benefit of formal technical reviews is the early discovery of errors so that they do not propagate to the next step in the software process.

A number of industry studies indicate that design activities introduce between 50 and 65 percent of all errors during the software process. However, formal review techniques have been shown to be up to 75 percent effective in uncovering design flaws.

By detecting and removing a large percentage of these errors, the review process substantially reduces the cost of subsequent steps in the development and support phases. Assume that an error uncovered **during design will cost 1.0** monetary unit to correct. Relative to this cost, the same error uncovered just **before testing** commences will **cost 6.5 units**; **during testing, 15 units**; and **after release, between 60 and 100 units**.

Defect Amplification and Removal:

A defect amplification model can be used to illustrate the generation and detection of errors during the preliminary design, detail design, and coding steps of the software engineering process. The model is illustrated schematically in Figure 8.2. A box represents a **software development step**. During the step, errors may be inadvertently generated. Review may fail to uncover newly generated errors and errors from previous steps, resulting in some number of errors that are passed through.

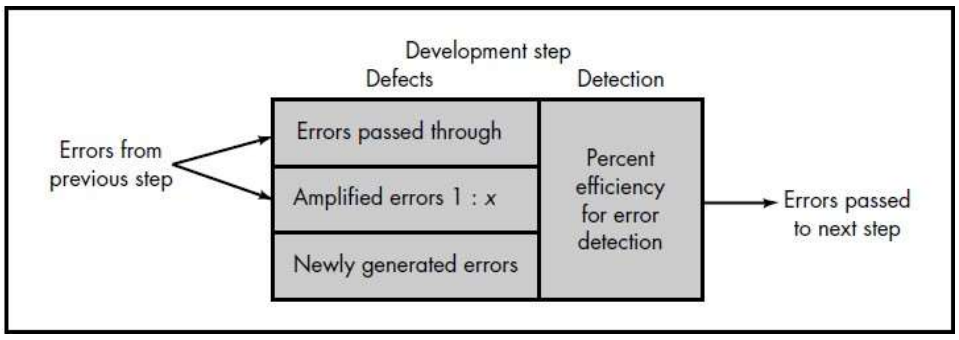
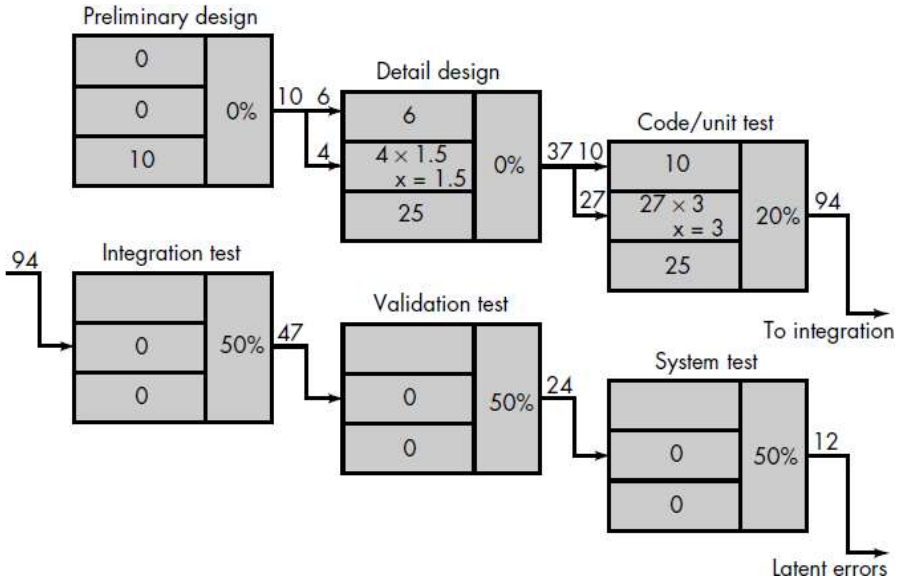


FIGURE 8.2
Defect amplification model

In some cases, errors passed through from previous steps are amplified (amplification factor, *x*) by current work. The box subdivisions represent each of these characteristics and the **percent of efficiency for detecting errors**, a function of the thoroughness of the review.

Figure 8.3 illustrates a hypothetical example of defect amplification for a software development process in which no reviews are conducted. Referring to the figure, each test

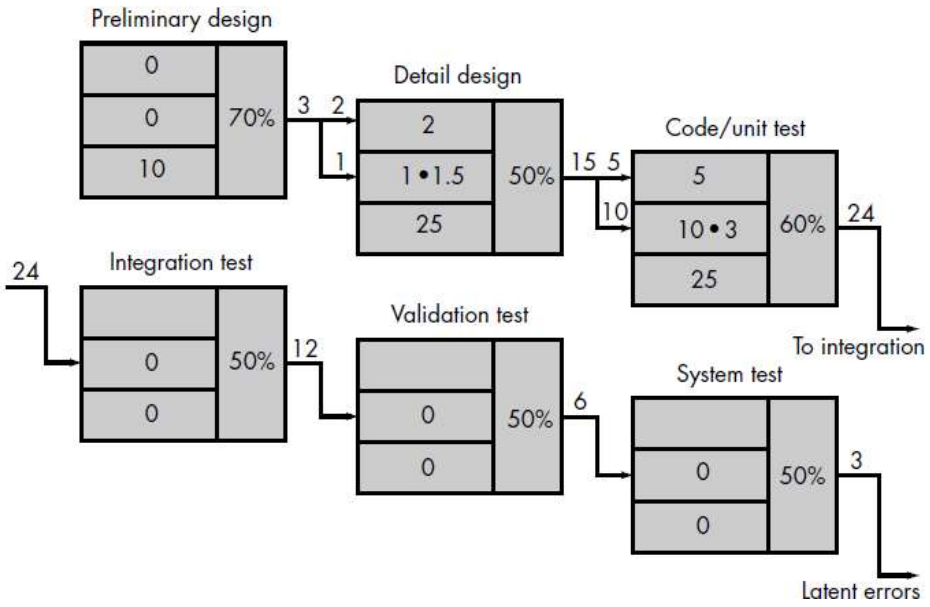
FIGURE 8.3
Defect amplification, no reviews



step is assumed to uncover and correct 50 percent of all incoming errors without introducing any new errors (an optimistic assumption). Ten preliminary design defects are amplified to 94 errors before testing commences. Twelve latent errors are released to the field.

Figure 8.4 considers the same conditions except that design and code reviews are conducted as part of each development step. In this case, ten initial preliminary design errors are amplified to 24 errors before testing commences.

FIGURE 8.4
Defect amplification, reviews conducted



Only three latent errors exist. Recalling the relative costs associated with the discovery and correction of errors, overall cost (with and without review for our hypothetical example) can be established.

The number of errors uncovered during each of the steps noted in Figures 8.3 and 8.4 is multiplied by the cost to remove an error (1.5 cost units for design, 6.5 cost units before test, 15 cost units during test, and 67 cost units after release). Using these data, the total cost for development and maintenance when reviews are conducted is 783 cost units. When no reviews are conducted, total cost is 2177 units—nearly three times more costly.

To conduct reviews, a software engineer must expend time and effort and the development organization must spend money. **Formal technical reviews (for design and other technical activities) provide a demonstrable cost benefit. They should be conducted.**

FORMAL TECHNICAL REVIEWS:

A formal technical review is a software quality assurance activity performed by software engineers (and others).

The objectives of the FTR are

(1) to **uncover errors** in function, logic, or implementation for any representation of the software;

(2) to verify that the software under review **meets its requirements**;

(3) to ensure that the software has been represented according to **predefined standards**

(4) to achieve software that is **developed in a uniform manner**; and

(5) to make projects **more manageable**.

In addition, the FTR serves as a training ground, enabling junior engineers to observe different approaches to software analysis, design, and implementation. The FTR also serves to promote backup and continuity.

The FTR is actually a class of reviews that includes walkthroughs, inspections, round-robin reviews and other small group technical assessments of software. **Each FTR is conducted as a meeting and will be successful only if it is properly planned, controlled, and attended.**

The Review Meeting:

Every review meeting should abide by the following constraints:

- Between **three and five people** (typically) should be involved in the review.
- **Advance preparation** should occur but should require no more than two hours of work for each person.
- The duration of the review meeting should be **less than two hours**.

For example, rather than attempting to review an entire design, walkthroughs are conducted for each component or small group of components. By narrowing focus, the FTR has a higher likelihood of uncovering errors.

- The **Software developer** who has developed the work product—the **producer**—informs the project leader that the work product is complete and that a review is required.
- The **project leader** contacts a **review leader**, who evaluates the product for readiness, generates copies of product materials, and distributes them to two or three reviewers for advance preparation.
- Each reviewer is expected to **spend between one and two hours** reviewing the product, making notes, and becoming familiar with the work.
- The review leader also reviews the product and establishes an **agenda** for the review meeting, which is typically scheduled for the next day.
- The review meeting is attended by the review leader, all reviewers, and the producer.
- One of the reviewers takes on the **role of the *recorder*;** that is, the individual who records (in writing) all important issues raised during the review.
- The FTR begins with an introduction of the agenda and a brief introduction by the producer. The producer then proceeds to "walk through" the work product, explaining the material, while reviewers raise issues based on their advance preparation. When valid problems or **errors are discovered, the recorder notes each**.
- At the end of the review, all attendees of the FTR must decide whether to

  (1) accept the product **without further modification**,

  (2) **reject the product** due to severe errors (once corrected, another review must be performed)

  (3) **accept the product provisionally** (minor errors have been encountered and must be corrected, but no additional review will be required).
- The decision made, all FTR attendees complete a sign-off, indicating their participation in the review and their concurrence with the review team's findings.

Review Reporting and Record Keeping:

During the FTR, a reviewer (the recorder) actively records all issues that have been raised. These are summarized at the end of the review meeting and a review issues list is produced**. In addition, a formal technical review summary report is completed**.

A ***review summary report*** **answers three questions**:

1. What was reviewed? **2.** Who reviewed it?

3. What were the findings and conclusions?

The review summary report is a **single page form** (with possible attachments). It becomes part of the project historical record and may be distributed to the project leader and other interested parties.

The **review issues list** serves two purposes:

(1) to **identify problem areas** within the product and

(2) to **serve as an action item checklist** that guides the producer as corrections are made. An issues list is normally attached to the summary report.

It is important to establish a follow-up procedure to ensure that items on the issues list have been properly corrected.

Review Guidelines:

Guidelines for the conduct of formal technical reviews must be established in advance, distributed to all reviewers, agreed upon, and then followed. The following represents a minimum set of guidelines for formal technical reviews:

1. Review the product, not the producer.

2. Set an agenda and maintain it.

3. Limit debate and rebuttal.

4. Enunciate problem areas, but don't attempt to solve every problem noted.

5. Take written notes.

6. Limit the number of participants and insist upon advance preparation.

7. Develop a checklist for each product that is likely to be reviewed

Checklists should be developed for analysis, design, code, and even test documents.

8. Allocate resources and schedule time for FTRs.

9. Conduct meaningful training for all reviewers.

10. Review your early reviews.

STATISTICAL SOFTWARE QUALITY ASSURANCE:

Statistical quality assurance reflects a growing trend throughout industry to become **more quantitative about quality**. For software, statistical quality assurance implies the following steps:

1. Information about software defects is collected and categorized.

2. An attempt is made to trace each **defect to its underlying cause** (e.g., non conformance to specifications, design error, violation of standards, poor communication with the customer).

3. Using the **Pareto principle** (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the "vital few").

4. Once the **vital few causes have been identified**, move to correct the problems that have caused the defects.

Although hundreds of different errors are uncovered, all can be tracked to one (or more) of the following causes:

- incomplete or erroneous specifications (IES)
- misinterpretation of customer communication (MCC)
- intentional deviation from specifications (IDS)
- violation of programming standards (VPS)
- error in data representation (EDR)
- inconsistent component interface (ICI)

- error in design logic (EDL)
- incomplete or erroneous testing (IET)
- inaccurate or incomplete documentation (IID)
- error in programming language translation of design (PLT)
- ambiguous or inconsistent human/computer interface (HCI)
- miscellaneous (MIS)

Once the vital few causes are determined as shown in table 8.1, the software engineering organization can begin corrective action. For example,

TABLE 8.1 DATA COLLECTION FOR STATISTICAL SQA

	Total		Serious		Moderate		Minor	
Error	**No.**	**%**	**No.**	**%**	**No.**	**%**	**No.**	**%**
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Totals	942	100%	128	100%	379	100%	435	100%

- **To correct MCC**, the software developer might implement facilitated application specification techniques to improve the quality of customer communication and specifications.
- **To improve EDR**, the developer might acquire CASE tools for data modeling and perform more stringent data design reviews.

SOFTWARE RELIABILITY:

Software reliability is defined in statistical terms as "the **probability of failure-free operation of a computer program** in a specified environment for a specified time". Software reliability can be measured, directed and estimated using historical and developmental data.

To illustrate, program X is estimated to have a **reliability of 0.96** over eight elapsed processing hours. In other words, if program X were to be executed 100 times and require eight hours of elapsed processing time (execution time), it is likely to operate correctly (without failure) **96 times out of 100**.

Failure is non-conformance to software requirements. **One failure** can be corrected **within seconds** while **another** requires weeks or even **months** to correct. Complicating the issue even further, the **correction of one failure** may in fact result in the **introduction of other errors** that ultimately result in other failures.

Measures of Reliability and Availability:

All software failures can be traced to design or implementation problems, a simple measure of reliability is *meantime- between-failure* (MTBF), where

MTBF = MTTF + MTTR

MTTF - mean time to failure

MTTR - mean time to repair

Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as

Availability = [MTTF/(MTTF + MTTR)] x 100%

Software Safety:

Software safety is a software quality assurance activity that focuses on the **identification and assessment of potential hazards** that may affect software negatively and cause an entire system to fail.

If hazards can be identified early in the software engineering process, software design features can be specified that will either eliminate or control potential hazards.

The hazards associated with a computer-based cruise control for an automobile might be

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

Once these system-level hazards are identified, analysis techniques are used to assign severity and probability of occurrence. To be effective, software must be analyzed in the context of the entire system.

For example, a user input error may be magnified by a software fault to produce control data that improperly positions a mechanical device, will cause a disastrous failure. **Analysis techniques such as *fault tree analysis, real-time logic*, or *petri net models*** can be used to predict the chain of events that can cause hazards and the probability of occurrence.

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

Software reliability uses statistical analysis to determine the likelihood that a software failure will occur. However, the occurrence of a failure does not necessarily result in a hazard or mishap**. Software safety** examines the ways in which failures result in conditions that can lead to a mishap.

THE ISO 9000 QUALITY STANDARDS:

Quality assurance systems are created to **help organizations ensure their products and services satisfy customer expectations** by meeting their specifications. A quality assurance system may be defined as the organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.

These systems cover a wide variety of activities **encompassing a product's entire life cycle** including planning, controlling, measuring, testing and reporting, and improving quality levels throughout the development and manufacturing process.

The ISO 9000 standards have been adopted by many countries. After adopting the standards, a country typically permits **only ISO registered companies to supply goods** and services to government agencies and public utilities.

Telecommunication equipment and medical devices are examples of product categories that must be supplied by ISO registered companies. Private companies such as automobile and computer manufacturers frequently require their **suppliers to be ISO registered**.

To become registered to one of the quality assurance system models contained in **ISO 9000**, a company's quality system and operations are **scrutinized by third party auditors** for compliance to the standard and for effective operation. Upon successful registration, a company is issued a **certificate from a registration body** represented by the auditors.

The ISO Approach to Quality Assurance Systems:

The ISO 9000 quality assurance models treat an enterprise as a network of interconnected processes. For a quality system to be ISO compliant, these processes must address the areas identified in the standard and must be documented and practiced as described.

ISO 9000 describes the elements of a quality assurance system in general terms. These elements include the organizational structure, procedures, processes, and resources needed to implement quality planning, quality control, quality assurance, and quality improvement. However, ISO 9000 does not describe how an organization should implement these quality system elements. Consequently, the challenge lies in designing and implementing a quality assurance system that meets the standard and fits the company's products, services, and culture.

The ISO 9001 Standard:

ISO 9001 is the quality assurance standard that applies to software engineering. The standard contains **20 requirements** that must be present for an effective quality assurance system. Because the ISO 9001 standard is applicable to all engineering disciplines,

a special set of ISO guidelines (ISO 9000-3) have been developed to help and interpret the standard for use in the software process.

The requirements delineated by ISO 9001 address topics such as management responsibility, quality system, contract review, design control, document and data control, product identification and traceability, process control, inspection and testing, corrective and preventive action, control of quality records, internal quality audits, training, servicing, and statistical techniques.

THE SQA PLAN :

The *SQA Plan* provides a **road map** for instituting software quality assurance. The plan serves as a **template for SQA activities** that are instituted for each software project.

A standard for SQA plans has been **recommended by the IEEE**. Initial sections describe the purpose and scope of the document and indicate those software process activities that are covered by quality assurance.

All documents noted in the *SQA Plan* are listed and all applicable standards are noted. The management section of the plan describes SQA's place in the organizational structure,

The **documentation section** describes each of the work products produced, as part of the software process. These include

- project documents (e.g., project plan)
- models (e.g., ERDs, class hierarchies)
- technical documents(e.g.,specifications, test plans)
- user documents (e.g., help files)

The standards, practices, and conventions section lists all applicable standards and practices that are **applied** during the software process (e.g., **document standards, coding standards, and review guidelines**). In addition, all project, process, and product metrics that are to be collected as part of software engineering work are listed.

The reviews and audits section of the plan identifies the **reviews and audits** to be conducted by the software engineering team, the **SQA group**, and the customer. It provides an overview of the approach for each review and audit.

Software Test Plan and Procedure defines test record-keeping requirements. **Problem reporting and corrective action defines procedures for** reporting, tracking, and resolving errors and defects, and identifies the organizational responsibilities for these activities.

The *SQA Plan also* identifies the tools and methods that support SQA activities and tasks; references software configuration management procedures for controlling change; defines a contract management approach; establishes methods for assembling, safeguarding, and maintaining all records; identifies training required to meet the needs of the plan; and defines methods for identifying, assessing, monitoring, and controlling risk.