Unit-5

RISK MANAGEMENT

What is it?

Risk analysis and management are a series of steps that help a software team to understand and manage uncertainty. Many problems can plague a software project. 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 should the problem actually occur.

Who does it?

Everyone involved in the software process—managers, software engineers, and customers—participate in risk analysis and management.

REACTIVE VS. PROACTIVE RISK STRATEGIES

Reactive risk strategies have been laughingly called the "Indiana Jones School of risk management". In the movies that carried his name, Indiana Jones, when faced with overwhelming difficulty, would invariably say, "Don't worry, I'll think of something!" Never worrying about problems until they happened, Indy would react in some heroic way.

Sadly, the average software project manager is not Indiana Jones and the members of the software project team are not his trusty sidekicks. Yet, the majority of software teams rely solely on reactive risk strategies.

At best, a reactive strategy monitors the project for likely risks. Resources are set aside to deal with them, should they become actual problems. 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 considerably 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. Throughout the remainder of this chapter, we discuss a proactive strategy for risk management.

SOFTWARE RISKS

Although there has been considerable debate about the proper definition for software risk, there is general agreement that risk always involves two characteristics:

a set of risk information sheets is produced. How do I ensure that I've done it right? The risks that are analyzed and managed should be derived from thorough study of the people, the product, the process, and the project. The RMMM should be revisited as the project proceeds to ensure that risks are kept up to date. Contingency plans for risk management should be realistic.

• Uncertainty—the risk may or may not happen; that is, there are no 100% probable risks.

• 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 project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, customer, and requirements problems and their impact on a software project.

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 design, implementation, interface, verification, and maintenance problems.

In addition, specification ambiguity, technical uncertainty, technical obsolescence, and "leading-edge" technology are also risk factors. Technical risks occur because the problem is harder to solve than we thought it would be. Business risks threaten the viability of the software to be built.

Business risks often jeopardize the project or the product. Candidates for the top five business risks are

(1) building a 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,

(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).

It is extremely important to note that simple categorization won't always work. Some risks are simply unpredictable in advance. Another general categorization of risks has been proposed by Charette.

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 are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance.

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 for each of the categories that have been presented in the above risks: 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 project at hand.

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?" One method for identifying risks is to create a risk item checklist.

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.

• **Customer characteristics**—risks associated with the sophistication of the customer and the developer's ability to communicate with the customer 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.

The risk item checklist can be organized in different ways. Questions relevant to each of the topics can be answered for each software project. The answers to these questions allow the planner to estimate the impact of risk.

A different risk item checklist format simply lists characteristics that are relevant to each generic subcategory. Finally, a set of “risk components and drivers" are listed along with their probability

Although generic risks are important to consider, usually the product-specific risks cause the most headaches. Be certain to spend the time to identify as many product-specific risks as possible.

Drivers for performance, support, cost, and schedule are discussed in answer to later questions. A number of comprehensive checklists for software project risk have been proposed in the literature. These provide useful insight into generic risks for software projects and should be used whenever risk analysis and management is instituted.

However, a relatively short list of questions can be used to provide a preliminary indication of whether a project is “at risk.”

Assessing Overall Project Risk The following questions have derived from risk data obtained by surveying experienced software project managers in different part of the world. 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 their customers?

4. Have customers been involved fully in the definition of requirements?

5. Do end-users have realistic expectations?

6. Is 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 U.S. Air Force has written a pamphlet that contains excellent guidelines for software risk identification and abatement. The Air Force approach requires that the project manager identify the risk drivers that affect software risk components are performance, cost, support, and schedule. In the context of this discussion, 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.

The impact of each risk driver on the risk component is divided into one of four impact categories—negligible, marginal, critical, or catastrophic.

Components / 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

RISK PROJECTION

Risk projection, also called risk estimation, attempts to rate each risk in two ways—the likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur. The project planner, along with other managers and technical staff, performs four risk projection activities:

(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, and

(4) note the overall accuracy of the risk projection so that there will be no misunderstandings.

Developing a Risk Table

A risk table provides a project manager with a simple technique for risk projection. A sample risk table is illustrated in the following Figure.

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	
•				
•				
•				

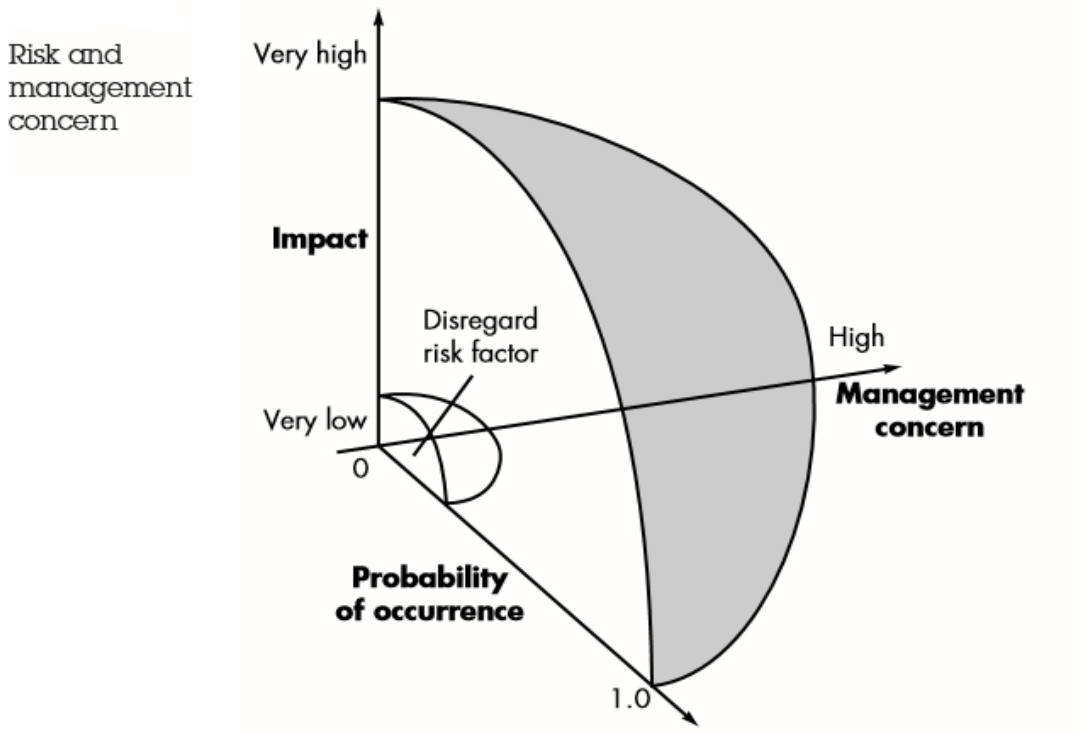
Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

A project team begins by listing all risks (no matter how remote) in the first column of the table. This can be accomplished with the help of the risk item checklists referenced in the above figure. 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. Individual team members are polled in round-robin fashion until their assessment of risk probability begins to converge.

Next, the impact of each risk is assessed. Each risk component is assessed using the characterization presented in impact table, and an impact category is determined. The categories for each of the four risk components—performance, support, cost, and schedule—are averaged3 to determine an overall impact value.

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 project manager studies the resultant sorted table and defines a cutoff line. 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 re-evaluated to accomplish second-order prioritization. The following Figure shows the risk impact and probability have a distinct influence on management concern.



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.

All risks that lie above the cutoff line must be managed. The column labeled RMMM contains a pointer into a Risk Mitigation, Monitoring and Management Plan or alternatively, a collection of risk information sheets developed for all risks that lie above the cutoff.

Risk probability can be determined by making individual estimates and then developing a single consensus value. Although that approach is workable, more sophisticated techniques for determining risk probability have been developed. Risk drivers can be assessed on a qualitative probability scale that has the following values: impossible, improbable, probable, and frequent.

Mathematical probability can then be associated with each qualitative value (e.g., a probability of 0.7 to 1.0 implies a highly probable risk).

Assessing Risk Impact

Three factors affect the consequences that are likely if a risk does occur: 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 preclude 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 customers are harmed?).

Finally, the timing of a risk considers when and for how long the impact will be felt. In most cases, a project manager might want the "bad news" to occur as soon as possible, but in some cases, the longer the delay, the better.

Returning once more to the risk analysis approach proposed by the U.S. Air Force, the following steps are recommended to determine the overall consequences of a risk:

1. Determine the average probability of occurrence value for each risk component.

2. Using impact table, determine the impact for each component based on the criteria shown.

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 x C

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

Risk Assessment

At this point in the risk management process, we have established a set of triplets of the form:

[ri, li, xi]

where ri is risk,

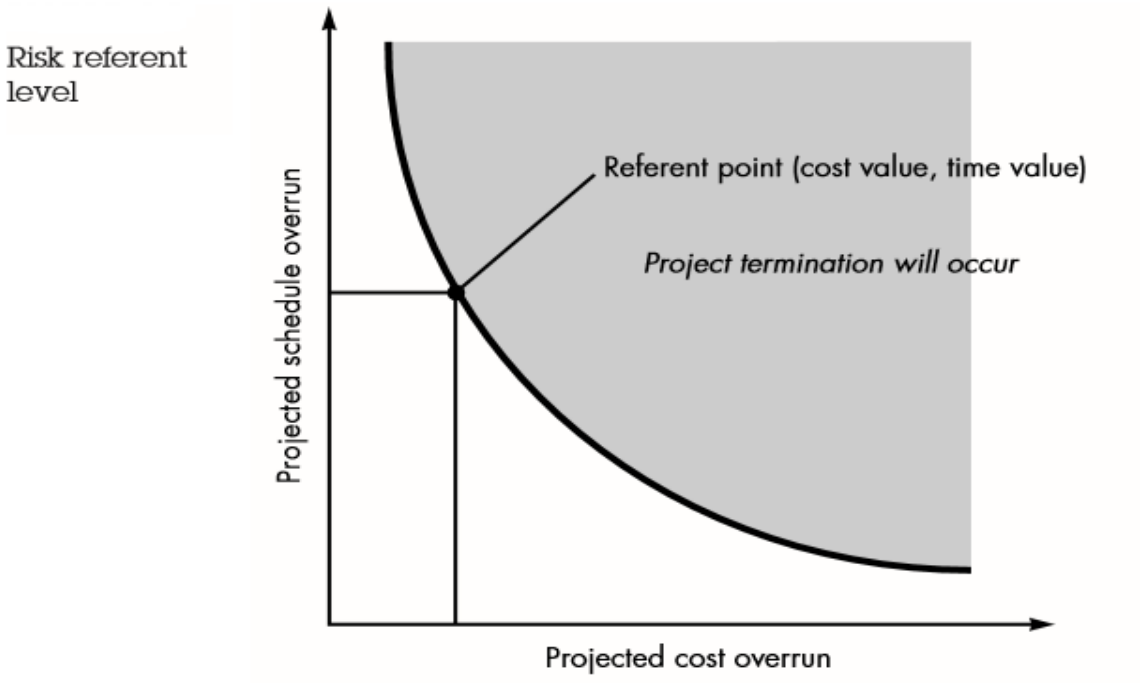
li is the likelihood (probability) of the risk,

xi is the impact of the risk.

During risk assessment, we further examine the accuracy of the estimates that were made during risk projection, attempt to rank the risks that have been uncovered, and begin thinking about ways to control and/or avert risks that are likely to occur.

For assessment to be useful, a risk referent level must be defined. For most software projects, the risk components discussed earlier—performance, cost, support, and schedule—also represent risk referent levels. That is, there is a level for performance degradation, cost overrun, support difficulty, or schedule slippage (or any combination of the four) that will cause the project to be terminated.

If a combination of risks create problems that cause one or more of these referent levels to be exceeded, work will stop. In the context of software risk analysis, a risk referent level has a single point, called the referent point or break point, at which the decision to proceed with the project or terminate it (problems are just too great) are equally weighted. The following Figure represents this situation graphically.



In reality, the referent level can rarely be represented as a smooth line on a graph. In most cases it is a region in which there are areas of uncertainty; that is, attempting to predict a management decision based on the combination of referent values is often impossible. Therefore, during risk assessment, we perform the following steps:

1. Define the risk referent levels for the project.

2. Attempt to develop a relationship between each (ri, li, xi) and each of the referent levels.

3. Predict the set of referent points that define a region of termination, bounded by a curve or areas of uncertainty.

4. Try to predict how compound combinations of risks will affect a referent level.

RISK REFINEMENT

During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about the project and the risk, it may be possible to refine the risk into a set of more detailed risks, each somewhat easier to mitigate, monitor, and manage. One way to do this is to represent 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>.

Using the CTC format for the reuse risk we can write:

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:

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.

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

RISK MITIGATION, MONITORING, AND MANAGEMENT (RMMM)

All of the risk analysis activities presented to this point have a single goal—to assist the project team in developing a strategy for dealing with risk. An effective strategy must consider three issues:

• risk avoidance

• risk monitoring

• risk management and contingency planning

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, r1. Based on past history and management intuition, the likelihood, l1, of high turnover is estimated to be 0.70 (70 percent, rather high) and the impact, x1,is projected at level 2. That is, high turnover will have a critical impact on project cost and schedule.

To mitigate this risk, project management must 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 that are under our control before the project starts.

• Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.

• Organize project teams so that information about each development activity is widely dispersed.

• Define documentation standards and establish mechanisms to be sure that documents are developed in a timely manner.

• Conduct peer reviews of all work (so that more than one person is "up to speed”).

• Assign a backup staff member for every critical technologist.

As the project proceeds, risk monitoring activities commence. 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 following factors can be monitored:

- 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.
- The availability of jobs within the company and outside it.

THE RMMM PLAN

A risk management strategy can be included in the software project plan or the risk management steps can be organized into a separate Risk Mitigation, Monitoring and Management Plan.

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. Some software teams do not develop a formal RMMM document.

Rather, 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 the following Figure.

Risk information sheet			
Risk ID: P02-4-32	Date: 5/9/02	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 7/1/02			
Current status: 5/12/02: Mitigation steps initiated.			
Originator: D. Gagne		Assigned: B. Laster	

Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence. As we have already discussed, risk mitigation is a problem avoidance activity.

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.

In many cases, the problems that occur during a project can be traced to more than one risk. Another job of risk monitoring is to attempt to allocate origin (what risk(s) caused which problems throughout the project).

UNIT-5

QUALITY MANAGEMENT

❖ QUALITY CONCEPTS

It has been said that no two snowflakes are alike. Certainly when we watch snow falling it is hard to imagine that snowflakes differ at all, let alone that each flake possesses a unique structure. In order to observe differences between snowflakes, we must examine the specimens closely, perhaps using a magnifying glass.

In fact, the closer we look, the more differences we are able to observe. This phenomenon, variation between samples, applies to all products of human as well as natural creation. For example, if two "identical" circuit boards are examined closely enough, we may observe that the copper pathways on the boards differ slightly in geometry, placement, and thickness.

In addition, the location and diameter of the holes drilled in the boards varies as well. All engineered and manufactured parts exhibit variation. The variation between samples may not be obvious without the aid of precise equipment to measure the geometry, electrical characteristics, or other attributes of the parts. However, with sufficiently sensitive instruments, we will likely come to the conclusion that no two samples of any item are exactly alike.

Variation control is the heart of quality control. A manufacturer wants to minimize the variation among the products that are produced, even when doing something relatively simple like duplicating diskettes.

➢ Quality

The American Heritage Dictionary defines quality as "a characteristic or attribute of something." As an attribute of an item, quality refers to measurable characteristics— things we are able to compare to known standards such as length, color, electrical properties, and malleability. However, software, largely an intellectual entity, is more challenging to characterize than physical objects.

Nevertheless, measures of a program's characteristics do exist. These properties include cyclomatic complexity, cohesion, number of function points, lines of code, and many others.

When we examine an item based on its measurable characteristics, 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 product increases, if the product is manufactured according to specifications.

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 development, quality of design encompasses requirements, specifications, and the design of the system.

Quality of conformance is an issue focused primarily on implementation. If the implementation follows the design and the resulting system meets its requirements and performance goals, conformance quality is high. But are quality of design and quality of conformance the only issues that software engineers must consider? Robert Glass argues that a more "intuitive" relationship is in order:

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

➢ Quality Control

Variation control may be equated to quality control. But how do we achieve 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.

➢ 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. Of course, 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.

➢ Cost of Quality

The cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities. Cost of quality studies are conducted to provide a base line for the current cost of quality, identify opportunities for reducing the cost of quality, and provide a normalized basis of comparison. The basis of normalization is almost always dollars.

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

changes in dollar-based terms. 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 to gain insight into product condition the “first time through” each process. Examples of appraisal costs include

- In-process and interprocess inspection
- Equipment calibration and maintenance
- Testing

Failure costs are those that would disappear if no defects appeared before shipping a product to customers. 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

❖ SOFTWARE REVIEWS

Software reviews are a "filter" for the software engineering process. That is, 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—any 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;
3. Achieve technical work of more uniform, or at least more predictable, quality than can be achieved without reviews, in order to make technical work more manageable.

A formal technical review is the most effective filter from a quality assurance standpoint. Conducted by software engineers (and others) for software engineers, the FTR is an effective means for improving software quality. The formal technical review, sometimes called a walkthrough or an inspection.

➢ Cost Impact of Software Defects

The IEEE Standard Dictionary of Electrical and Electronics Terms (IEEE Standard 1001992) defines a defect as "a product anomaly." The definition for fault in the hardware context can be found in IEEE Standard 610.12-1990:

(a) A defect in a hardware device or component; for example, a short circuit or broken wire.
(b) An incorrect step, process, or data definition in a computer program.

- Within the context of the software process, the terms defect and fault are synonymous. Both imply a quality problem that is discovered after the software has been released to end-users (or to another activity in the software process).

- The term error is used to depict a quality problem that is discovered by software engineers (or others) before the software is released to the end-user (or to another activity in the software process).

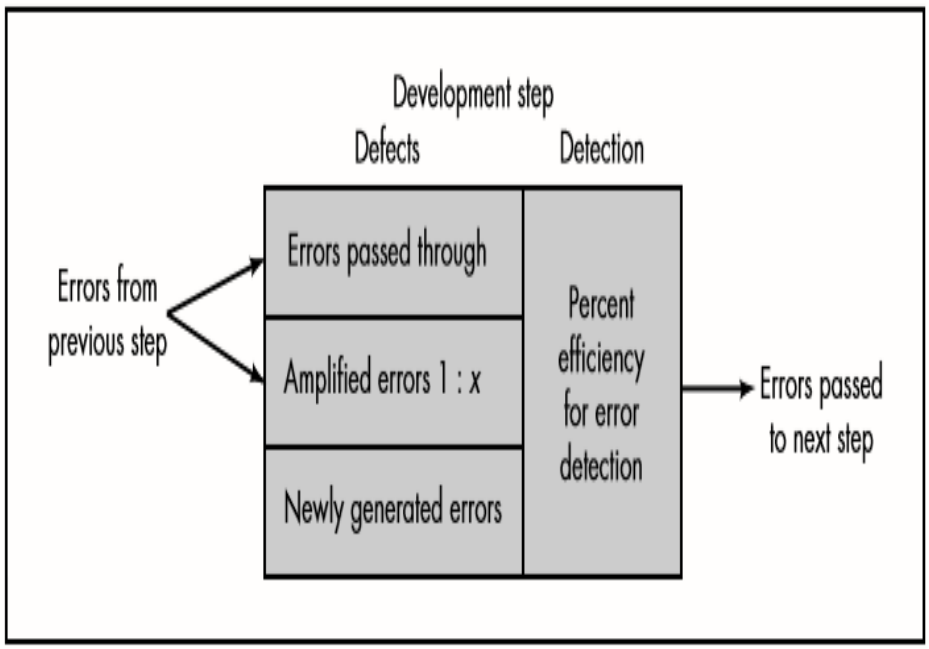
The primary objective of formal technical reviews is to find errors during the process so that they do not become defects after release of the software.

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.

➢ 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 following figure.

Defect amplification model



- In the above figure 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.

- 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.

  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 given bellow

(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;
(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 because a number of people become familiar with parts of the software that they may not have otherwise seen.

- **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

Regardless of the FTR format that is chosen, 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.

Given these constraints, it should be obvious that an FTR focuses on a specific (and small) part of the overall software.

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 focus of the FTR is on a work product (e.g., a portion of a requirements specification, a detailed component design, a source code listing for a component).

The individual 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 otherwise becoming familiar with the work.

Concurrently, 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), or

(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

(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. Unless this is done, it is possible that issues raised can "fall between the cracks." One approach is to assign the responsibility for follow up to the review leader.

Review Guidelines

Guidelines for the conduct of formal technical reviews must be established in advance, distributed to all reviewers, agreed upon, and then followed.

A review that is uncontrolled can often be worse that no review at all. The following represents a minimum set of guidelines for formal technical reviews:

1. **Review the product, not the producer.**

- An FTR involves people and egos. Conducted properly, the FTR should leave all participants with a warm feeling of accomplishment. Conducted improperly, the FTR can take on the aura of an inquisition.
- Errors should be pointed out gently; the tone of the meeting should be loose and constructive; the intent should not be to embarrass or belittle.

- The review leader should conduct the review meeting to ensure that the proper tone and attitude are maintained and should immediately halt a review that has gotten out of control.

2. **Set an agenda and maintain it.**

   One of the key maladies of meetings of all types is drift. An FTR must be kept on track and on schedule. The review leader is chartered with the responsibility for maintaining the meeting schedule and should not be afraid to nudge people when drift sets in.

3. **Limit debate and rebuttal.**

   When an issue is raised by a reviewer, there may not be universal agreement on its impact. Rather than spending time debating the question, the issue should be recorded for further discussion off-line.

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

   A review is not a problem-solving session. The solution of a problem can often be accomplished by the producer alone or with the help of only one other individual. Problem solving should be postponed until after the review meeting.

5. **Take written notes.**

   It is sometimes a good idea for the recorder to make notes on a wall board, so that wording and priorities can be assessed by other reviewers as information is recorded.

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

   Two heads are better than one, but 14 are not necessarily better than 4. Keep the number of people involved to the necessary minimum. However, all review team members must prepare in advance. Written comments should be solicited by the review leader (providing an indication that the reviewer has reviewed the material).

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

   A checklist helps the review leader to structure the FTR meeting and helps each reviewer to focus on important issues. Checklists should be developed for analysis, design, code, and even test documents.

8. **Allocate resources and schedule time for FTRs.**

   For reviews to be effective, they should be scheduled as a task during the software engineering process. In addition, time should be scheduled for the inevitable modifications that will occur as the result of an FTR.

9. **Conduct meaningful training for all reviewers.**

   To be effective all review participants should receive some formal training. The training should stress both process-related issues and the human psychological side of reviews. Freedman and Weinberg estimate a one-month learning curve for every 20 people who are to participate effectively in reviews.

10. **Review your early reviews.**

    Debriefing can be beneficial in uncovering problems with the review process itself. The very first product to be reviewed should be the review guidelines themselves.

❖ SOFTWARE RELIABILITY

There is no doubt that the reliability of a computer program is an important element of its overall quality. If a program repeatedly and frequently fails to perform, it matters little whether other software quality factors are acceptable.

- Software reliability, unlike many other quality factors, can be measured directed and estimated using historical and developmental data.

- **Software reliability is defined in statistical terms as "the probability of failure-free operation of a computer program in a specified environment for a specified time".**

- 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.
- Whenever software reliability is discussed, a pivotal question arises:
  What is meant by the term failure?
  In the context of any discussion of software quality and reliability, failure is nonconformance to software requirements.

- Yet, even within this definition, there are gradations. Failures can be only annoying or catastrophic.
- One failure can be corrected within seconds while another requires weeks or even months to correct.
- Complicating the issue even further, the correction of one failure may in fact result in the introduction of other errors that ultimately result in other failures.

➢ Measures of Reliability and Availability

- Early work in software reliability attempted to extrapolate the mathematics of hardware reliability theory to the prediction of software reliability.

- Most hardware-related reliability models are predicated on failure due to wear rather than failure due to design defects.

- In hardware, failures due to physical wear (e.g., the effects of temperature, corrosion, shock) are more likely than a design-related failure.

- Unfortunately, the opposite is true for software. In fact, all software failures can be traced to design or implementation problems; wear does not enter into the picture.

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

- **MTBF = MTTF + MTTR**

The acronyms MTTF and MTTR are mean-time-to-failure and mean-time-to-repair, respectively.

In addition to a reliability measure, we must develop a measure of availability.

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

Availability = [MTTF/(MTTF + MTTR)] X100%

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

➢ 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.

A modeling and analysis process is conducted as part of software safety. Initially, hazards are identified and categorized by criticality and risk.

For example, some of the hazards associated with a computer-based cruise control for an automobile might be

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

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

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 that each of the events will occur to create the chain.

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

Although software reliability and software safety are closely related to one another, it is important to understand the subtle difference between them.

Software reliability uses statistical analysis to determine the likelihood that a software failure will occur. However, the occurrence of a failure does not necessarily result in a hazard or mishap.

Software safety examines the ways in which failures result in conditions that can lead to a mishap. That is, failures are not considered in a vacuum, but are evaluated in the context of an entire computer-based system.