

UNIT – V

Risk management:

First, risk concerns future happenings. Today and yesterday are beyond active concern, as we are already reaping what was previously sowed by our past actions. The question is, can we, therefore, by changing our actions today, create an opportunity for a different and hopefully better situation for ourselves tomorrow. This means, second, that risk involves change, such as in changes of mind, opinion, actions, or places . . . [Third,] risk involves choice, and the uncertainty that choice itself entails. Thus paradoxically, risk, like death and taxes, is one of the few certainties of life.

When you consider risk in the context of software engineering, Charrette's three conceptual underpinnings are always in evidence. The future is your concern—what risks might cause the software project to go awry? Change is your concern—how will changes in customer requirements, development technologies, target environments, and all other entities connected to the project affect timeliness and overall success? Last, you must grapple with choices— what methods and tools should you use, how many people should be involved, how much emphasis on quality is “enough”?

Peter Drucker [Dru75] once said, “While it is futile to try to eliminate risk, and questionable to try to minimize it, it is essential that the risks taken be the right risks.” Before you can identify the “right risks” to be taken during a software project, it is important to identify all risks that are obvious to both managers and practitioners.

Reactive vs. Proactive Risk strategies:

Reactive risk strategies have been laughingly called the “Indiana Jones school of risk management” [Tho92]. 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” [Cha92] takes over and the project is in real jeopardy.

A considerably more intelligent strategy for risk management is to be pro- active. 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:

Although there has been considerable debate about the proper definition for software risk, there is general agreement that risk always involves two characteristics: uncertainty—the risk may or may not happen; that is, there are no 100 percent probable risks¹—and loss—if the risk becomes a reality, unwanted consequences or losses will occur [Hig95]. 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.


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 you thought it would be.

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).

It is extremely important to note that simple risk categorization won't always work. Some risks are simply unpredictable in advance.

Another general categorization of risks has been proposed by Charette [Cha89]. 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.



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:

- 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.
- 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.
- Encourage open communication**—if someone states a potential risk, don't discount it. If a risk is proposed

- in an informal manner, consider it. Encourage all stakeholders and users to suggest risks at any time.
- Integrate**—a consideration of risk must be integrated into the software process.
- 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.
- 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.
- Encourage teamwork**—the talents, skills, and knowledge of all stakeholders should be pooled when risk management activities are conducted.

Risk identification:

1. Assessing overall project Risk

- **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.

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 you 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” [AFC88] are listed along with their probability of occurrence. Drivers for performance, support, cost, and schedule are discussed in answer to later questions.

A number of comprehensive checklists for software project risk are available on the Web (e.g., [Baa07], [NAS07], [Wor04]). You can use these checklists to gain insight into generic risks for software projects. In addition to the use of check- lists, risk patterns [Mil04] have been proposed as a systematic approach to risk identification.

Assessing Overall Project Risk:

The following questions have been derived from risk data obtained by surveying experienced software project managers in different parts of the world [Kei98]. 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.

2. Risk components and drivers:

The U.S. Air Force [AFC88] has published 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—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. Referring to **below Figure of Impact assessment** [Boe89], 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 projection, also called risk estimation, attempts to rate each risk in two ways—

- (1) The likelihood or probability that the risk is real and will occur and
- (2) The consequences of the problems associated with the risk, should it occur. You work along with other managers and technical staff to perform 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. No software team has the resources to address every possible risk with the same degree of rigor. By prioritizing risks, you can allocate resources where they will have the most impact.

**Impact
assessment**
Source: [Boe89].

Components		Performance	Support	Cost	Schedule
Category					
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.

1. Developing a risk Table

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

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				
Σ				
Σ				

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

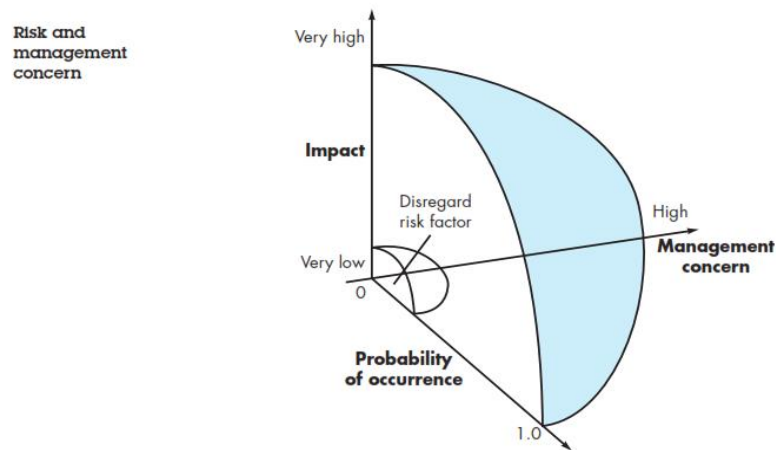
You begin by listing all risks (no matter how remote) in the first column of the table. This can be accomplished with the help of the risk item checklists. Each risk is categorized in the second column (e.g., PS implies a project size risk, BU implies a business risk). The probability of occurrence of 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. Each risk component is assessed using the characterization presented in above Figure of Impact assessment, and an impact category is determined. The categories for each of the four risk components—performance, support, cost, and schedule—are averaged 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.

You can study the resultant sorted table and define 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 reevaluated to accomplish second-order prioritization. Referring to *below Figure*, 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 should 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 cut-off.

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.

2. 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 stakeholders are harmed?). Finally, the timing of a risk considers when and for how long the impact will be felt. In most cases, you 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 [AFC88], you can apply 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 Impact assessment Figure, 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

[Hal98]:

$$RE = 5 P^3 C$$

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

For example, assume that the software team defines a project risk in the following manner:

Risk identification. 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.

Risk probability. Eighty percent (likely).

Risk impact. Sixty reusable software components were planned. If only 70 percent can be used, 18 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development). 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 the components would be $18 \times 100 \times 14 = \$25,200$.

Risk exposure. $RE = 5 (0.80)^3 \times 25,200 \sim \$20,200$

Risk exposure can be computed for each risk in the risk table, once an estimate of the cost of the risk is made. The total risk exposure for all risks (above the cut-off in the risk table) can provide a means for adjusting the final cost estimate for a project. It can also be used to predict the probable increase in staff resources required at various points during the project schedule.

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 [Glu94]. That is, the risk is stated in the following form:

Using the CTC format for the reuse risk noted in Assessing Risk Impact, you could 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:

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 sub conditions 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, RMMM Plan:

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, and 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 *r*. Based on past history and management

intuition, the likelihood l of high turnover is estimated to be 0.70 (70 percent, rather high) and the impact x is projected as critical. That is, high 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, and competitive job market).
- Mitigate those causes that are under your 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 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 (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 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. For example, a risk mitigation step noted here called for the definition of work product standards and mechanisms to be sure that work products are developed in a timely manner. This is one mechanism for ensuring continuity, should a critical individual leave the project. The project manager should monitor work products carefully to ensure that each can stand on its own and that each imparts information that would be necessary if a newcomer were forced to join the software team somewhere in the middle of the project.

Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality. Continuing the 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, you 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 or Wikis,” and/or meeting with other team members who will remain on the project.

It is important to note that risk mitigation, monitoring, and management (RMMM) steps incur additional project cost. For example, spending the time to back up every critical technologist costs money. Part of risk management, therefore, is to evaluate when the benefits accrued by the RMMM steps are outweighed by the costs associated with implementing them.

In essence, you perform a classic cost-benefit analysis. If risk aversion steps for high turnover will increase both project cost and duration by an estimated 15 percent, but the predominant cost factor is “backup,” management may decide not to implement this step. On the other hand, if the risk aversion steps are projected to increase costs by 5 percent and duration by only 3 percent, management will likely put all into place.

For a large project, 30 or 40 risks may be identified. If between three and seven risk management steps are identified for each, risk management may become a project in itself. For this reason, you should adapt the Pareto 80–20 rule to software risk. Experience indicates that 80 percent of the overall project risk (i.e., 80 percent of the potential for project failure) can be accounted for by only 20 percent of the identified risks. The work performed during earlier risk analysis steps will help you to determine which of the risks reside in that 20 percent (e.g., risks that lead to the highest risk exposure). For this reason, some of the risks identified, assessed, and projected may not make it into the RMMM plan—they don’t fall into the critical 20 percent (the risks with highest project priority).

Risk is not limited to the software project itself. Risks can occur after the software has been successfully developed and delivered to the customer. These risks are typically associated with the consequences of software failure in the field.

Software safety and hazard analysis (e.g., [Dun02], [Her00], [Lev95]) are software quality assurance activities (Chapter 21) that focus 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.

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) [Wil97]. 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.

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; and (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].

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	

Fig: RISK Information Sheet

Software quality assurance:

The software engineering approach described in this book works toward a single goal: to produce on-time, high-quality software. Yet many readers will be challenged by the question: “What is software quality?”

Philip Crosby [Cro79], in his landmark book on quality, provides a wry answer to this question:

The problem of quality management is not what people don't know about it. The problem is what they think they do know . . .

In this regard, quality has much in common with sex. Everybody is for it. (Under certain conditions, of course.) Everyone feels they understand it. (Even though they wouldn't want to

explain it.) Everyone thinks execution is only a matter of following natural inclinations. (After all, we do get along somehow.) And, of course, most people feel that problems in these areas are caused by other people. (If only they would take the time to do things right.)

Some software developers continue to believe that software quality is something you begin to worry about after code has been generated. Nothing could be further from the truth! Software quality assurance (often called quality management) is an umbrella activity that is applied throughout the software process.

Software quality assurance (SQA) encompasses:

- (1) An SQA process,
- (2) Specific quality assurance and quality control tasks (including technical reviews and a multi-tiered testing strategy),
- (3) Effective software engineering practice (methods and tools),
- (4) Control of all software work products and the changes made to them,
- (5) A procedure to ensure compliance with software development standards (when applicable), and
- (6) Measurement and reporting mechanisms.

Here, we focus on the management issues and the process-specific activities that enable a software organization to ensure that it does “the right things at the right time in the right way.”

Elements of SQA:

Software quality assurance encompasses a broad range of concerns and activities that focus on the management of software quality. These can be summarized in the following manner:

Standards. The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.

Reviews and audits. Technical reviews are a quality control activity performed by software engineers for software engineers (Chapter 20). Their intent is to uncover

errors. Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work. For example, an audit of the review process might be conducted to ensure that reviews are being performed in a manner that will lead to the highest likelihood of uncovering errors.

Testing. Software testing (Chapters 22 through 26) is a quality control function that has one primary goal—to find errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted so that it has the highest likelihood of achieving its primary goal.

Error/defect collection and analysis. The only way to improve is to measure how you're doing. SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.

Change management. Change is one of the most disruptive aspects of any software project. If it is not properly managed, change can lead to confusion, and confusion almost always leads to poor quality. SQA ensures that adequate change management practices (Chapter 29) have been instituted.

Education. Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement (Chapter 37) and is a key proponent and sponsor of educational programs.

Vendor management. Three categories of software are acquired from external software vendors—shrink-wrapped packages (e.g., Microsoft Office), a tailored shell [Hor03] that provides a basic skeletal structure that is custom tailored to the needs of a purchaser, and contracted software that is custom designed and constructed from specifications provided by the customer organization. The job of the SQA organization is to ensure that high-quality software results by suggesting specific quality practices that the vendor should follow (when possible), and incorporating quality mandates as part of any contract with an external vendor.

Security management. With the increase in cybercrime and new government regulations regarding privacy, every software organization should institute policies that protect data at all levels, establish firewall protection for WebApps, and ensure that software has not been tampered with internally. SQA ensures that appropriate process and technology are used to achieve software security.

Safety. Because software is almost always a pivotal component of human-rated systems (e.g., automotive or aircraft applications), the impact of hidden defects can be catastrophic. SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.

Risk management. Although the analysis and mitigation of risk is the concern of software engineers, the SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

In addition to each of these concerns and activities, SQA works to ensure that software support activities (e.g., maintenance, help lines, documentation, and manuals) are conducted or produced with quality as a dominant concern.

SQA Tasks, Goals and Metrics:

Software quality assurance is composed of a variety of tasks associated with two different constituencies—the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, oversight, record keeping, analysis, and reporting.

Software engineers address quality (and perform quality control activities) by applying solid technical methods and measures, conducting technical reviews, and performing well-planned software testing.

SQA Tasks

The charter of the SQA group is to assist the software team in achieving a high-quality end product. The Software Engineering Institute recommends a set of SQA activities that address quality assurance planning, oversight, record keeping, analysis, and reporting. These activities are performed (or facilitated) by an independent SQA group that

Prepares an SQA plan for a project. The plan is developed as part of project planning and is reviewed by all stakeholders. Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan. The plan identifies evaluations to be performed, audits and reviews to be conducted, standards that are applicable to the project, procedures for error reporting and tracking, work products that are produced by the SQA group, and feedback that will be provided to the software team.

Participates in the development of the project's software process description. The software team selects a process for the work to be performed. 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. The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.

Audits designated software work products to verify compliance with those defined as part of the software process. The SQA group reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the results of its work to the project manager.

Ensures that deviations in software work and work products are documented and handled according to a documented procedure. Deviations may be encountered in the project plan, process description, applicable standards, or software engineering work products.

Records any noncompliance and reports to senior management.

Noncompliance items are tracked until they are resolved

Goals, Attributes, and Metrics:

The SQA activities described in the preceding section are performed to achieve a set of pragmatic goals:

Requirements quality. The correctness, completeness, and consistency of the requirements model will have a strong influence on the quality of all work products that follow. SQA must ensure that the software team has properly reviewed the requirements model to achieve a high level of quality.

Design quality. Every element of the design model should be assessed by the software team to ensure that it exhibits high quality and that the design itself conforms to requirements. SQA looks for attributes of the design that are indicators of quality.

Code quality. Source code and related work products (e.g., other descriptive information) must conform to local coding standards and exhibit characteristics that will facilitate maintainability. SQA should isolate those attributes that allow a reasonable analysis of the quality of code.

Quality control effectiveness. A software team should apply limited resources in a way that has the highest likelihood of achieving a high-quality result. SQA analyzes the allocation of resources for reviews and testing to assess whether they are being allocated in the most effective manner.

The below Figure (adapted from [Hya96]) identifies the attributes that are indicators for the existence of quality for each of the goals discussed. Metrics that can be used to indicate the relative strength of an attribute are also shown.

Software quality goals, attributes, and metrics

Source: Adapted from [Hya96].

Goal	Attribute	Metric
Requirement quality	Ambiguity	Number of ambiguous modifiers (e.g., many, large, human-friendly)
	Completeness	Number of TBA, TBD
	Understandability	Number of sections/subsections
	Volatility	Number of changes per requirement
		Time (by activity) when change is requested
	Traceability	Number of requirements not traceable to design/code
	Model clarity	Number of UML models
		Number of descriptive pages per model
Design quality		Number of UML errors
	Architectural integrity	Existence of architectural model
	Component completeness	Number of components that trace to architectural model
		Complexity of procedural design
	Interface complexity	Average number of pick to get to a typical function or content
Code quality		Layout appropriateness
	Patterns	Number of patterns used
	Complexity	Cyclomatic complexity
	Maintainability	Design factors (Chapter 8)
	Understandability	Percent internal comments
		Variable naming conventions
	Reusability	Percent reused components
	Documentation	Readability index
QC effectiveness	Resource allocation	Staff hour percentage per activity
	Completion rate	Actual vs. budgeted completion time
	Review effectiveness	See review metrics (Chapter 14)
	Testing effectiveness	Number of errors found and criticality
		Effort required to correct an error
		Origin of error