Chapter 2

SOFTWARE PROJECTS MANAGEMENT

2.1

Metrics in Process and Project Domain

Measurement is less common in the software engineering world. Metrics should be collected so that process and product indicators can be ascertained.

Process indicators

- It enables software engineering organization to get the efficacy of an existing process.
- They enable managers and practitioners to assess what works and what doesn't.
- Process metrics are collected across all projects and over long periods of time.

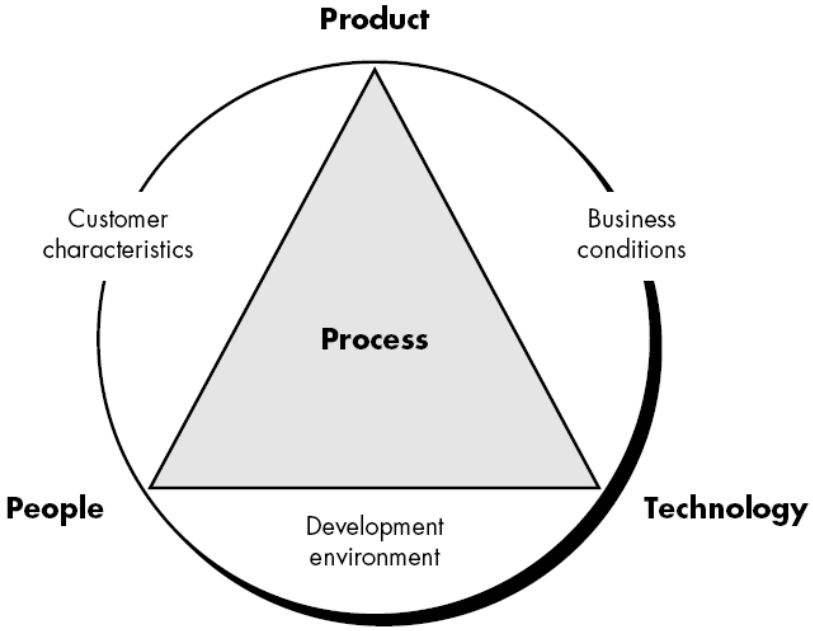
Project indicators

Enable a software project manager to

- assess the status of an ongoing project,
- track potential risks,
- uncover problem areas before they go "critical,"
- adjust work flow or tasks, and
- evaluate the project team's ability to control quality of software work products.

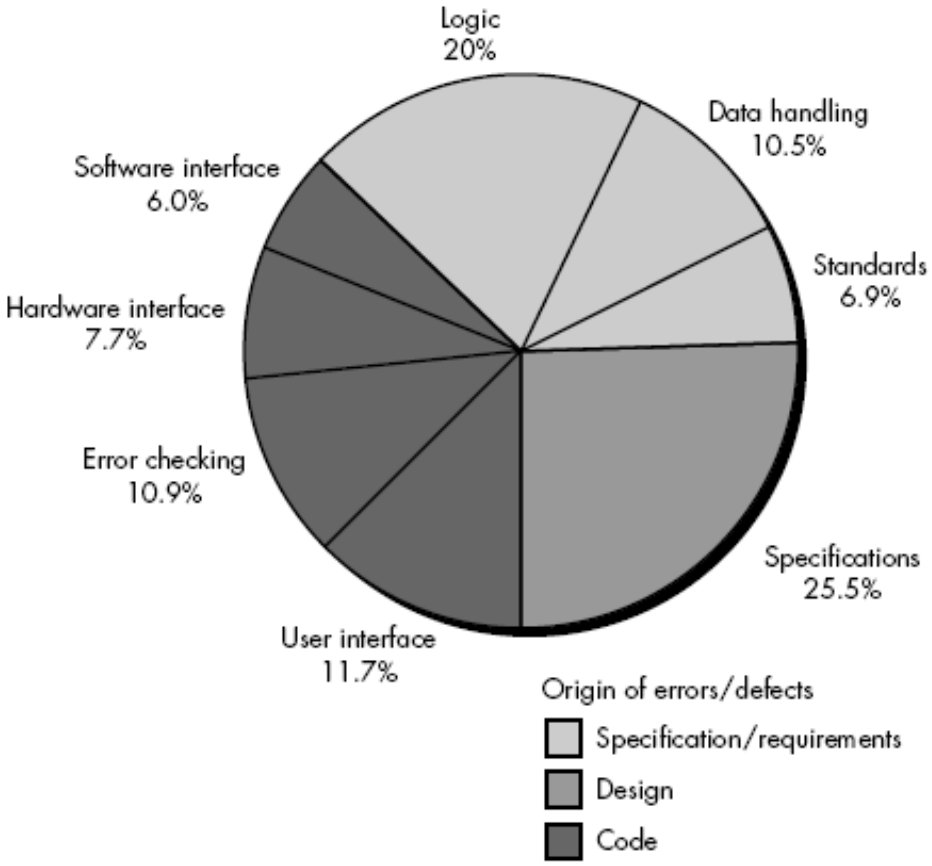
Process Metric

- Process is only one of a number of "controllable factors in **improving software quality and organizational performance.**
- Process **sits at the center** of a triangle connecting three factors that have an influence on software quality.
- We measure the efficacy of a software process indirectly.



- **It shows them how to define processes and how to measure their quality** and productivity.
- Private process data can serve as an important driver as the individual software engineer works to improve.
- Some **process metrics are private to the software project team but public** to all team members.
- Project level defect rates (absolutely not attributed to an individual), **effort, calendar times, and related** data are collected can improve organizational **proces**s performance.
- Software process metrics can provide significant benefit as an organization works to improve its overall level.
- Process Metrics can be useful for failure analysis as

    1. All **errors and defects are categorized** by origin (e.g., flaw in specification, flaw in logic, nonconformance to standards).
    2. The **cost to correct** each error and defect is recorded.
    3. The number **of errors and defects** in each category is **counted** and ranked in descending order.
    4. **The overall cost of errors and defects in each category is computed**.
    5. Resultant data are analyzed to uncover the categories that result in highest cost to the organization.
    6. **Plans are developed to modify the process** with the intent of eliminating (or reducing the frequency of) the class of errors and defects that is most costly.



Project Metric

- Project metrics and the indicators derived from them are used to **adapt project work flow and technical activities.**
- The application of **project metrics** on most software projects **occurs during estimation**. **Metrics collected from past projects are used as a basis from which effort and time estimates** are made for current software work.
- As a project proceeds, measures of **effort and calendar time expended are compared to original estimates**
- The project manager **uses data to monitor** and control progress.
- Project metrics begin to have significance Production rates represented in **terms of pages of documentation, review hours, function points, and delivered source lines are measured**.
- The project metrics are used to **minimize the development schedule** by making the adjustments necessary to avoid delays
- **Project metrics are used to assess product quality** on an ongoing basis and, when necessary, modify the technical approach to improve quality.

- **Project metric is leads to a reduction in overall project cost.**

Software project metrics suggests that every project should measure:

• Inputs—measures of the resources required to do the work.
• Outputs—measures of the deliverables or work products created during the software engineering process.
• Results—measures that indicate the effectiveness of the deliverables

Software Measurements

- Measurements in the physical world can be categorized in two ways:
  - ✓ direct measures (e.g., the length of a bolt) and
  - ✓ indirect measures (e.g., the "quality").

- Software metrics can be categorized similarly.
  - Direct measures of the software engineering process include
    cost and effort applied.
    include lines of code (LOC) produced,
    execution speed,
    memory size, and
    defects reported over some set period of time.
  - Indirect measures of the product include
    functionality,
    quality,
    complexity,
    efficiency,
    reliability,
    maintainability,

Size-Oriented Metrics

- Size-oriented measures the size of the software that has been produced. If a software organization maintains simple records, a table of size-oriented measures,
- Such as the one shown in Figure for project alpha: 12,100 lines of code were developed with 24 person-months of effort at a cost of $168,000. 365 pages of documentation were developed, 134 errors were recorded and 29 defects
- It should be noted that the effort and cost recorded in the table represent all software engineering activities (analysis, design, code, and test), not just coding.

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

- set of simple size-oriented metrics can be developed for each project:
- Errors per KLOC (thousand lines of code).
- Defects per KLOC.
- Cost per LOC.
- Page of documentation per KLOC.
- Errors per person-month.
- LOC per person-month.
- $ per page of documentation.

- Size-oriented metrics are not universally accepted as the best way to measure the process of software development

Function-Oriented Metrics

- Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value.
- Since **'functionality' cannot be measured directly**, it must be derived indirectly using other direct measures.
- Its measurements of function points, as

Number of user inputs. Each user input that provides distinct application oriented data to the software is counted. Inputs should be distinguished from inquiries, which are counted separately.

Number of user outputs. Each user output that provides application-oriented information to the user is counted. In this context output refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.

Number of user inquiries. An inquiry is defined as an **on-line input that results in the generation of some immediate software response** in the form of
an on-line output. Each distinct inquiry is counted.

Number of files. Each logical master file (i.e., a logical grouping of data that may be one part of a large database or a separate file) is counted.

Number of external interfaces. All machine readable interfaces (e.g., data files on storage media) that are used to transmit information to another.

Metrics for Software Quality

The goal of software engineering is to produce a high-quality system, application, to achieve this goal, software engineers must apply effective methods. Although many quality measures can be collected, the **primary thrust at the project level is to measure errors and defects.** Error data can also be used to compute the *defect removal efficiency* (DRE) for each process framework.

Measuring Quality
Although there are many measures of software quality,

Correctness.
Correctness is the degree to which the software performs its required function. The most **common measure for correctness is defects per KLOC,** where a defect is defined as a verified lack of conformance to requirements.

Maintainability.
Maintainability **is the ease with which a program can be corrected if an error is encountered**, adapted if its environment changes, or enhanced if the customer desires a change in requirements. There is no way to measure maintainability directly; therefore, we must use indirect measures.

Integrity. Software integrity has become increasingly important in the age of hackers and firewalls. This attribute measures a system's **ability to withstand attacks (both accidental and intentional) to its security**. Attacks can be made on all three components of software

Usability.
If a program is not **user-friendly**, it is often doomed to failure, even if the functions that it performs are valuable. Usability is an attempt to quantify user-friendliness and can be measured in terms of four characteristics:
(1) the physical and or intellectual skill required to learn the system,
(2) the **time required to become moderately efficient in the use of the system**,
(3) the net increase in productivity (over the approach that the system replaces) measured when the system is used by someone who is moderately efficient, and
(4) a subjective assessment (sometimes obtained through a questionnaire) of user attitudes toward the system.

Integrating Metrics Within the Software Process

- It's important to measure the process of software engineering and the product(software) that it produces.
- **If we do not measure, there no real way of determining whether we are improving**.
- And if we are not improving, we are lost. Hence, metrics is used to establish a process baseline from which improvements can be assessed.
- The collection of quality metrics enables an organization to "tune" its software process to remove the "vital few" causes of defects that have the greatest impact on software development.
- Metrics includes technical guide for improvement
    - **user requirements are to change**
    - To find **components in this system are most error prone**
    - **No of Testing** should be planned for each component
    - **No of errors expecting during testing**
- The metrics baseline **consists of data collected from past software development projects**.

- Process improvement and/or cost and effort estimation, baseline data must have the following attributes:
    - **data must be** reasonably **accurate**
    - data should be collected for as many projects as possible;
    - measures must be consistent, for example, a line of code must be interpreted consistently across all projects for which data are collected;
    - applications should be similar to work that is to be estimated

Metrics for Small Organizations

- It is reasonable to suggest that software organizations measure and then use the resultant **metrics to help improve their local software process** and the quality
- and timeliness of the products they produce.
- Keep measurements simple, tailored them to each organization, and ensured that they produced valuable information
- **Keep it simple, customize to meet local needs**, and be sure it adds value.
- "Keep it simple" is a guideline that works reasonably well in many activities.
- We begin by focusing not on measurement but rather on results.
- A small organization might select the following set of easily collected measures:
  • **Time (hours or days) elapsed from the time a request is made** until
  Evaluation is complete
  • **Effort (person-hours) to perform** the evaluation
  • Time (hours or days) elapsed from completion of evaluation to assignment of change order to personnel, teal.
  • **Effort (person-hours) required to make the change**
  • **Time required (hours or days) to make the change**
  • **Errors uncovered during work to make change**, Exchange.
  • **Defects uncovered after change is released** to the customer base,

- Once these measures have been collected for a number of change requests, it is possible to compute the total elapsed time from change request to implementation.

2.2

Observations on Estimating

- Estimation of **resources, cost, and schedule** for a software engineering effort requires experience in it
- It's an access **to good historical information, requires for the development of software.**
- Although **estimating is as much art as it is science**, this important action need not be conducted
- **Useful techniques** for time and effort estimation do exist.
- Process and project metrics can provide historical perspective and powerful input for the generation of quantitative estimates.
- **Past experience** can aid immeasurably as estimates are developed and reviewed.
- Because estimation lays **a foundation for all other project planning actions, and project planning provides the road map for successful software engineering**,
- Estimation of resources, cost, and schedule for a software engineering effort requires experience, access to good historical information (metrics), and the courage to commit to quantitative predictions when qualitative information is all that exists.
- Problem decomposition, an important approach to estimating, becomes more difficult.
- The availability of historical information has a strong influence on estimation risk. By looking back, you can emulate things that worked and improve areas where problems arose.
- **When comprehensive software metrics are available for past projects, estimates can be made with greater assurance, schedules can be established to avoid past difficulties, and overall risk is reduced**.
- Estimation risk is measured by the degree of uncertainty in the quantitative estimates established for resources, cost, and schedule.
- **If project scope is poorly understood** or project requirements are subject to change
- As a planner, you and the customer should recognize **that variability in software requirements means instability in cost and schedule**.
- However, you should not become obsessive about estimation. Modern software engineering approaches take an iterative view of development.

- In such approaches, it is possible—although not always politically acceptable—to revisit the estimate and revise it when the customer makes changes to requirements

Project Planning Process

- The objective of software project planning is to provide a framework that enables the manager to make reasonable **estimates of resources**, cost, and schedule.
- In addition, estimates should attempt to define **best-case** and **worst-case** scenarios so that project outcomes can be bounded.
- **The plan must be adapted and updated as the project proceeds**.

Task Set for Project Planning

1. Establish **project scope**.
2. Determine **feasibility**.
3. **Analyze risks**
4. Define required resources.
    a. **Determine required human resources**.
    b**. Define reusable software resources**.
    c. **Identify environmental resources**.
5. Estimate **cost and effort**.
    a. **Decompose the problem**.
    b. Develop two or more **estimates using size, function points, process tasks, or use cases.**
    c. Reconcile the estimates.
6. Develop a project schedule
    a. Establish a meaningful task set.
    b. Define a task network.
    c. Use scheduling tools to develop **a time-line chart.**
    d. **Define schedule tracking mechanisms**.

Software Scope and Feasibility

- Software **scope describes the functions and features that are to be delivered to end users**;
- The **first activity in software project planning is the determination of software scope**.
- **Function and performance allocated to software during system engineering should be assessed** to establish a project scope.

- A statement of software scope must be bounded. Software scope describes the data **and control to be processed, function, performance, constraints, interfaces, and reliability**.
- Functions described in the statement of scope are evaluated and, in some cases, refined to provide more detail prior to the beginning of estimation.
- **As both cost and schedule estimates are functionally oriented, some degree of decomposition** is often useful.
- Performance considerations encompass processing and response time requirements. Constraints identify limits placed on the software by external hardware, available memory, or other existing systems.
- Initiate the communication that is essential to establish the scope of the project.
- But a question and answer meeting format is not an approach that has been overwhelmingly successful for understanding the scope
- The Question **&Answers session** s**hould** be used for the first encounter only and then be replaced by a meeting format that combines elements of problem solving, negotiation, and specification.
- Customers and software engineers often have an unconscious "us and them" mindset.
- **A number of independent investigators have developed a team-oriented approach to requirements gathering that can be applied to help establish the scope of** a project.
- **Facilized application specification techniques (FAST)**, this approach encourages the **creation of a joint team of customers and developers who work together to identify the problem, propose elements of the solution**, negotiate different approaches, and specify a preliminary set of requirements

Feasibility

- Once scope has been identified it is reasonable to ask: “Can we build software to meet this scope? Is the project feasible?”
- Software engineers rush past these questions only to become mired in a project that is doomed from the onset.
- After a few hours or sometimes a few weeks of investigation, you are sure you can do it again.
- Projects on the margins of your experience are not so easy.
- A team may have to spend several months discovering what the central, difficult-to-implement requirements of a new application actually are.

- Do some of these requirements pose risks that would make the project infeasible? Can these risks be overcome?
- The feasibility team ought to carry initial architecture and design of the high-risk requirements to the point at which it can answer these questions.
- In some cases, when the team gets negative answers, a reduction in requirements may be negotiated.
- Once scope is understood, the software team and others must work to determine if it can be done within the dimensions just noted.
- This is a crucial, although often overlooked, part of the estimation process.

Software feasibility has four solid dimensions:

- Technology—
  1) Is a project **technically feasible**?
  2) Is it **within the state of the art**?
  3) Can **defects be reduced** to a level matching the application's needs?

- Finance—
  1) Is it **financially feasible**?
  2) Can development **be completed at a cost the software organization**?
  3) **its client, or the market can afford?**

- Time—Will the project's time-to-market beat the competition?

- Resources—Does the organization have the resources needed to succeed?

Resources

The second planning task is estimation of the resources required to accomplish the software development effort.

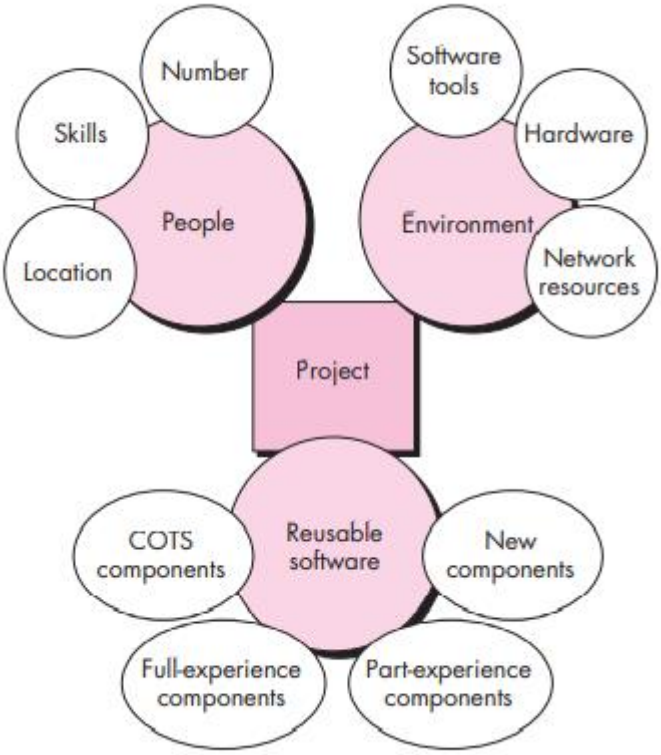


Fig: **Project Resources**

Figure shows the three major categories of software engineering resources—people, reusable software components, and the development environment (hardware and software tools).

Human Resources (People)

- The planner begins by evaluating software scope and selecting the skills required to complete development. Both organizational position (e.g., **manager, senior software engineer**) and specialty (e.g., telecommunications, database, client-server) are specified
- For larger projects, the **software team** may be geographically dispersed across a number of different locations. Hence, the location of each human resource is specified. The number of people required for a software project can be determined only after an estimate of development effort (e.g., person-months) is made.

Reusable Software Resources

- Component-based software engineering (CBSE) emphasizes reusability—that is, the creation and **reuse of software building blocks**.

- Such building blocks, often called components, must be cataloged for easy reference, standardized for easy application, and validated for easy integration.
- Existing software that can be **acquired from a third party or from a past project**. COTS (commercial off-the-shelf) components are purchased from a third party, are ready for use on the current project, and have been fully validated.
- Existing specifications**, designs, code, or test data developed for past projects** that are related to the software to be built for the current project but will require substantial modification.
- Ironically, **reusable software components are often neglected during planning**, only to become a paramount concern later in the software process.

Environmental Resources

- **The environment that supports a software project**, often called the software engineering environment (SEE), incorporates hardware and software.
- **Hardware provides platform** that supports the tools (software) required to produce the work products that are an outcome of good software engineering practice.
- When a computer-based system is to be engineered, the software team may require access to hardware elements being developed by other engineering teams.

Software Project Estimation

- Software cost and effort estimation will never be an exact science. Too many variables—**human, technical, environmental**, political—can affect the ultimate cost of software and effort applied to develop it.
- To achieve reliable cost and effort estimates, a number of options arise:

1. we can achieve 100 percent accurate estimates **after the project is complete**
2. Base **estimates on similar projects** that have already been completed.
3. Use relatively **simple decomposition techniques** to generate project cost and effort estimates.
4. Use one or more empirical models for software cost and effort estimation.

- **Longer you wait, the more you know**, and the more you know, the less likely you are to make serious errors in your estimates.
- If the current project is quite similar to past efforts and other project influences (e.g., the customer, business conditions, the software engineering environment, deadlines) are roughly equivalent.
- Ideally, the techniques noted for each option should be applied in tandem; each used as a **cross-check** for the other.
- **Decomposition techniques take a divide-and-conquer** approach to software project estimation.
- By decomposing a project into major functions and related software engineering activities, cost and effort estimation can be performed in a stepwise fashion.
- If no historical data exist, costing rests on a very based foundation.

Decomposition Techniques

- Software project estimation is a form of problem solving,
- Estimation uses one or both forms of partitioning.
- But before an estimate can be made, you must understand the scope of the software to be built and generate an estimate of its "size."
- **Software Sizing** The accuracy of a software project estimate is predicated on a number of things:
  (1) the degree to which you have properly estimated the **size of the product** to be built;
  (2) the ability to translate the **size estimate into human effort, calendar time, and costs**
  (3) the degree to which the project plan reflects the **abilities of the software team**; and
  (4) the stability of product requirements and the environment that supports the **software engineering effort**.
- In the context of project planning, **size refers to a quantifiable outcome of the software project**.
  - If a **direct approach** is taken, size can be measured in lines of code **(LOC).**
  - If **an indirect approach** is chosen, size is represented **as function points** (FP).

Various approaches to the sizing problem:

- ✓ "**Fuzzy logic" sizing**. To apply this approach, the **planner must identify the type of application**, establish its magnitude on a qualitative scale, and then refine the magnitude within the original range.

- ✓ **Function point sizing**. The **planner develops estimates of the information domain** characteristics discussed.
- ✓ **Standard component sizing**. Software is composed of a number of different "standard components" that are generic to a particular application area. For example, the standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC, and object-level instructions.
- ✓ **Change sizing**. use of existing software that must be modified in some way as part of a project. The planner estimates the number and type e.g., reuse, adding code, changing code, deleting code of modifications that must be accomplished.

Example: LOC-Based Estimation

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

2.3

Software Risks

- Risk always involves two characteristics:
    a. Uncertainty—the risk may or may not happen; that is, there are no 100 percent
    b. 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 **project schedule will slip and that costs will increase**.
    - ✓ **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.
    - ✓ **Business risks** 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).**

Risk Identification

- Risk identification is a systematic attempt to specify threats to the project plan
- The checklist can be used for risk identification
    - ✓ 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 buil**t 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.

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**.

Risk Projection

- Risk projection, also called risk estimation
- Four risk projection steps:
  1. **Establish a scale** that reflects the perceived likelihood of a risk.
  2. **Sketch the consequences** of the risk.
  3. **Estimate the impact of the risk** on the project and the product.
  4. **Assess the overall accuracy of the risk projection** so that there will be no misunderstandings.
- The intent of these steps is to consider risks in a manner that leads to **prioritization**

- **By prioritizing risks, you can allocate resources** where they will have the most impact
- **Risk Projection by developing Risk Table**

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

Here,

PS: project size risk,
BU implies a business risk
CU: cost
TE: Technology
Testify

- The software team defines a project risk in the following manner:
  Risk identification. 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. Likely as chances.
  Risk impact. Impact on schedule, cost
  Risk exposure. Again, requirement of funds

Risk Refinement

- 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.**

- **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,
- 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.

Risk Mitigation Monitoring and Management

2.4

Software Configuration Management

- The items that comprise **all information produced as part of the software process** are collectively called a software configuration.
- Software configuration **management is a set of activities that have been developed to manage change** throughout the life cycle of computer software.
- **SCM can be viewed as a software quality assurance activity that is applied throughout the software process**.

SCM Repository

- In the early days of software engineering, software configuration items were maintained as paper documents (or punched computer cards!), placed in file folders or This approach was problematic for many reasons:
- During the early history of software engineering, the repository was indeed a person—the **programmer who had to remember the location of all information relevant** to a software project, who had **to recall information** that was never written down and **reconstruct information that had been lost**
- **Today, the repository is a "thing"— a database that acts as the center for both accumulation and storage of software engineering information.**
- The role of the person (the software engineer) is to interact with the repository using tools that are integrated with it.

Advantages

- **finding a configuration item** when it was needed is easier
- Determining **which items were changed**, **when and by whom** was often successfully
- **constructing a new version** of an existing program was
- **describing detailed or complex relationships** between configuration items are easy

The Role of the Repository

- The SCM repository is the set of mechanisms and data structures that **allow a software team to manage change** in an effective manner.

- It provides the obvious functions of a **modern database management system by** ensuring data integrity, sharing, and integration.
- In addition, the SCM repository **provides a hub for the integration of software tools,**
- Is **central to the flow of the software process, and can enforce uniform structure and format for software engineering work products**.
- It determines how **data can be accessed by tools** and viewed by software engineers,
- It determines how well **data security** and **integrity** can be maintained,
- It determines **how easily the existing model can be extended** to accommodate new needs

General Features and Content

- The features and content of the repository are best understood from **two perspectives:**
  1. **what is to be stored** in the repository and
  2. **what specific services are provided** by the repository?
- A repository provides two different classes of services:
  1. the same types of **services that might be expected from any sophisticated database management system** and
  2. services that **are specific to the software engineering** environment.
- A repository that serves a software engineering team should also
  1. integrate with or **directly support process management functions**
  2. support **specific rules** that govern the **SCM function** and the data **maintained within the repository**,
  3. Provide an **interface to other software engineering tools, and**
  4. Accommodate **storage of sophisticated data objects** (e.g., text, graphics, video, audio).

SCM Features

Versioning.

- As a project progresses, **many versions** of individual **work products will be created**.
- **The repository** must be able to **save all of these versions** to enable effective management of product releases and

- **permit developers to go back to previous versions during testing and debugging**.
- A mature repository **tracks version.**

Dependency tracking and change management.

- The **repository manages a wide variety of relationships among the data elements stored** in it,
- Include **relationships between enterprise entities and processes**, among the parts of an application design, between design components and the enterprise information architecture, between design elements and deliverables, and so on.
- Some of these relationships are merely **associations**, and some are **dependencies** or mandatory relationships.
- The ability to keep track of all of these relationships **is crucial to the integrity of the information stored in the repository**

.

Requirements tracing.

- **provides the ability to track all the design and construction components** and deliverables that result from a specific requirements specification (forward tracing).
- it provides the ability to **identify which requirement generated** any given work product (backward tracing).

Configuration management.

A configuration management facility keeps **track of a series of configurations representing specific project milestones or production releases**.

Audit trails.

An audit trail establishes additional information about **when, why, and by whom changes are made**.

SCM Process

The software configuration management process four primary objectives:

1) to **identify all items** that collectively define the software configuration
2) **to manage changes to one or more of these items**
3) to facilitate the **construction of different versions of an application**
4) to **ensure that software quality is maintained** as the configuration evolves

Identification of Objects in the Software Configuration

- To control and manage software configuration items, **each should be separately named and then organized using an object-oriented approach**.
- Two types of objects can be identified
  - basic objects and
  - aggregate objects.
- A basic object is a unit of **information that you create during analysis, design, code,** or test.
  - For example, a basic object might be a section of a requirements specification, **part of a design model, source code for a component**, or a suite of test cases that are used to exercise in code.
- An aggregate object **is a collection of basic objects and other** aggregate objects.
  - For example, a **Design Specification is an aggregate** object.
- Each **object has a set of distinct features** that identify it uniquely: **a name, a description, a list of resources, and a "realization."**
- The **object name is a character string** that identifies the object unambiguously.
- The **object description is a list of data items** that identify the SCI type (e.g., model element, program, data)

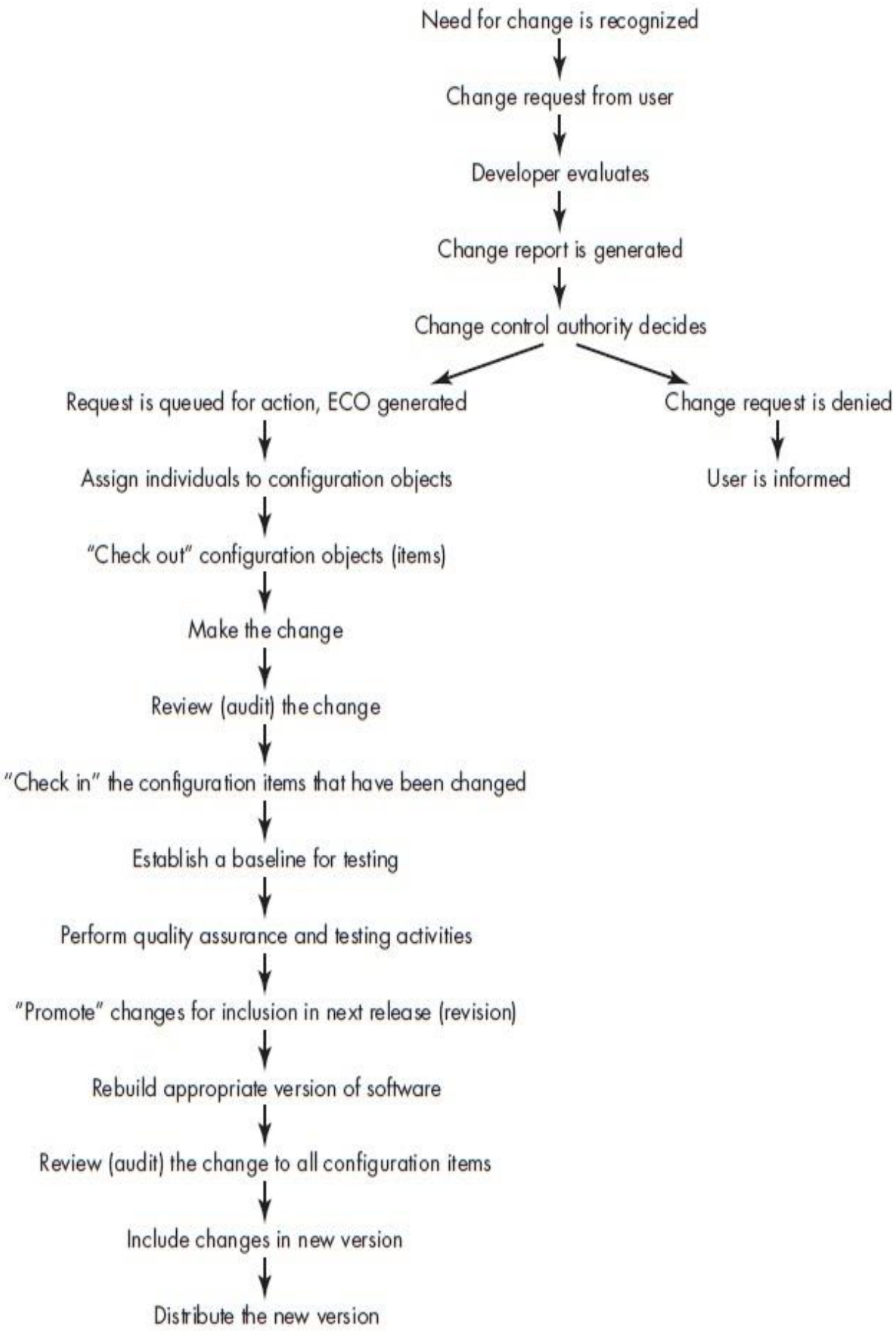
Version Control

- Version control combines procedures and tools to **manage different versions** of configuration objects **that are created during the software process**.
- A version control system implements or **is directly integrated with given a project database (repository)** that stores all relevant configuration objects,
- a **version management capability that stores all versions of a configuration object**
- It enables any **version to be constructed using differences from past versions**
- It enables to collect all relevant configuration and objects and **construct a specific version of the software**.
- In addition, version control **enables the team to record and track the status of all outstanding issues associated with each configuration object**.

- A number of version control systems establish a **collection of all changes that are required to create a specific version** of the software.
- A version control **enables to construct a version of the software by specifying the change sets**

Change Control

- Worry about change because it can create a big failure in the product, but it can also fix a big failure.
- Worry about change because a single developer could sink the project; but a change control process could effectively discourage them from doing creative.
- For a large software project, uncontrolled change rapidly leads to Failure. For such projects, change control combines human procedures and automated tools to provide a mechanism for the control of change.
- The change control process is illustrated schematically in Figure
  A **change request is submitted and evaluated to assess technical merit, potential side effects, overall impact on other configuration objects and system functions, and the projected cost of the change**.
- The results of the evaluation are presented as a change report, which is used by a change control authority (CCA)—a person or group that makes a final decision.
- An engineering change order (ECO) is generated for each approved change.
- The ECO describes the change to be made, the constraints that must be respected.


Need for change is recognized
Change request from user
Developer evaluates
Change report is generated
Change control authority decides
Request is queued for action, ECO generated
Change request is denied
Assign individuals to configuration objects
User is informed
"Check out" configuration objects (items)
Make the change
Review (audit) the change
"Check in" the configuration items that have been changed
Establish a baseline for testing
Perform quality assurance and testing activities
"Promote" changes for inclusion in next release (revision)
Rebuild appropriate version of software
Review (audit) the change to all configuration items
Include changes in new version
Distribute the new version