# Contents

# 1 Software And Software Eegineering

## 1.1 How should we define software?

Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information, and (3) descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

## 1.2 Characteristics of software that make it different from hardware?

1. Software is developed or engineered; it is not manufactured in the classical sense .
2. Software doesn't wear out, but it does deteriorate.
3. Although the industry is moving toward component-based construction, most software continues to be custom built.

## 1.3 Seven broad categories of computer software ?

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- Web applications
- Artificial intelligence software

## 1.4 Software engineering layers?

- Tools
- Methods
- Process
- a quality Focus

## 1.5 What is a Process ?

Aprocessis a collection of **activities**, **actions**, and **tasks** that are performed when some work product is to be created.

## 1.6 What are the five generic process framework activities?

- Communication
- Planning
- Modeling
- Construction
- Deployment

## 1.7 What is umbrella activities ?

umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk.

## 1.8 Typical umbrella activities ?

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

## 1.9 Types of Software Myths?

### 1.9.1 Management myths

Myths : We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

Reality : The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable? Is it streamlined to improve time-to-delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no."

Myths : If we get behind schedule, we can add more programmers and catch up (sometimes called the "Mongolian horde" concept).

Reality : "adding people to a late software project makes it later." At first, this statement may seem counterintuitive. However, as new people are added, people who were working must spend time educating the newcomers

Myths : If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality : If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

### 1.9.2 Customer myths

Myths : A general statement of objectives is sufficient to begin writing programs—we can fill in the details later .

Reality : Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster. Unambiguous requirements are developed only through effective and continuous communication between customer and developer.

Myths : Software requirements continually change, but change can be easily accommodated because software is flexible.

Reality : It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small . However, as time passes, the cost impact grows rapidly

### 1.9.3 Practitioner's myths

Myths : Once we write the program and get it to work, our job is done.

Reality : Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myths : Until I get the program "running" I have no way of assessing its quality.

Reality : One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the technical review.

Myths : The only deliverable work product for a successful project is the working program.

Reality : A working program is only one part of a software configuration that includes many elements. A variety of work products (e.g., models, documents, plans) provide a foundation for successful engineering and, more important, guidance for software support.

Myths : Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down

Reality : Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

# 2 Process Models

## 2.1 what is software process ?

software process is a framework for the activities, actions, and tasks that are required to build high-quality software.

## 2.2 Is "process" synonymous with software engineering?

The answer is "yes and no." A software process defines the approach that is taken as software is engineered. But software engineering also encompasses technologies that populate the process—technical methods and automated tools.

## 2.3 the communication activity might have six distinct actions?

- inception
- elicitation
- elaboration
- negotiation
- specification
- validation

## 2.4 What is Task Set?

A task set defines the actual work to be done to accomplish the objectives of a software engineering action .

## 2.5 Process Models

### 2.5.1 The Waterfall Model

The waterfall model,sometimes called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software

### 2.5.2 problems of the waterfall model ?

- Real projects rarely follow the sequential flow that the model proposes.
- It is often difficult for the customer to state all requirements explicitly.
- The customer must have patience. A working version of the program(s) will not be available until late in the project time span.

the linear nature of the classic life cycle leads to "blocking states" in which some project team members must wait for other members of the team to complete dependent tasks. In fact, the time spent waiting can exceed the time spent on productive work!

### 2.5.3 Incremental Process Models

there may be a compelling need to provide a limited set of software functionality to users quickly and then refine and expand on that functionality in later software releases.

The incremental model combines elements of linear and parallel process flows

Each linear sequence produces deliverable "increments" of the software

When an incremental model is used, the first increment is often a core product.

### 2.5.4 Evolutionary Process Models

a set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined.

Evolutionary models are iterative. They are characterized in a manner that enables you to develop increasingly more complete versions of the software.

### 2.5.5 Prototyping

Often, a customer defines a set of general objectives for software, but does not identify detailed requirements for functions and features. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take. In these, and many other situations, a prototyping paradigmmay offer the best approach.

### 2.5.6 The Spiral Model

Originally proposed by Barry Boehm , thespiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software.

Using the spiral model, software is developed in a series of evolutionary releases. During early iterations, the release might be a model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

The spiral model is a realistic approach to the development of large-scale systems .

### 2.5.7 Concurrent Models

The concurrent development model,sometimes called concurrent engineering,allows a software team to represent iterative and concurrent elements of any of the process models .

### 2.5.8 Component-Based Development

The component-based development model incorporates many of the characteristics of the spiral model. It is evolutionary in nature , demanding an iterative approach to the creation of software. However, the component-based development model constructs applications from prepackaged software components.

# 3 Project Management Concepts

## 3.1 Effective software project management focuses on the four P's

- people
- product
- process
- project

## 3.2 stakeholders can be categorized into one of five constituencies ?

- **Senior managers** who define the business issues that often have a significant influence on the project.
- **Project (technical) managers** who must plan, motivate, organize, and control the practitioners who do software work.
- **Practitioners** who deliver the technical skills that are necessary to engineer a product or application.
- **Customers** who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
- **End users** who interact with the software once it is released for production use.

## 3.3 MOI model of leadership :

What do we look for when choosing someone to lead a software project?

- Motivation
- Organization
- Ideas or innovation

## 3.4 effective project manager emphasizes four key traits

- Problem solving
- Managerial identity
- Achievement
- Influence and team building

## 3.5 Mantei describes seven project factors that should be considered when planning the structure of software engineering teams :

What factors should be considered when the structure of a software team is chosen?

- Difficulty of the problem to be solved
- "Size" of the resultant program(s) in lines of code or function points
- Time that the team will stay together (team lifetime)
- Degree to which the problem can be modularized
- Required quality and reliability of the system to be built
- Rigidity of the delivery date
- Degree of sociability (communication) required for the project

## 3.6 Constantine suggests four "organizational paradigms" for software engineering teams :

- **closed paradigm** structures a team along a traditional hierarchy of authority. Such teams can work well when producing software that is quite similar to past efforts, but they will be less likely to be innovative when working within the closed paradigm.
- **random paradigm** structures a team loosely and depends on individual initiative of the team members. When innovation or technological breakthrough is required, But such teams may struggle when "orderly performance" is required.
- **open paradigm** attempts to structure a team in a manner that achieves some of the controls associated with the closed paradigm but also much of the innovation that occurs when using the random paradigm. Open paradigm team structures are well suited to the solution of complex problems but may not perform as efficiently as other teams.
- **synchronous paradigm** relies on the natural compartmentalization of a problem and organizes team members to work on pieces of the problem with little active communication among themselves.

## 3.7 To achieve a high-performance team :

- Team members must have trust in one another.
- The distribution of skills must be appropriate to the problem.
- Mavericks may have to be excluded from the team, if team cohesiveness is to be maintained.

## 3.8 What is a "jelled" team?

A jelled team is a group of people so strongly knit that the whole is greater than the sum of the parts .

## 3.9 many teams suffer from five factors what Jackman calls “team toxicity” :

- a frenzied work atmosphere
- high frustration that causes friction among team members
- a “fragmented or poorly coordinated” software process
- an unclear definition of roles on the software team
- “continuous and repeated exposure to failure.”

In addition to the five toxins described by Jackman, a software team often struggles with the differing human traits of its members.

## 3.10 How to choose the software process model ?

- the customers who have requested the product and the people who will do the work
- the characteristics of the product itself
- the project environment in which the software team works

## 3.11 simple project work tasks for the communication activity ?

- Develop list of clarification issues.
- Meet with stakeholders to address clarification issues.
- Jointly develop a statement of scope.
- Review the statement of scope with all concerned.
- Modify the statement of scope as required.

## 3.12 complex project work tasks for the communication activity ?

- Review the customer request.
- Plan and schedule a formal, facilitated meeting with all stakeholders.
- Conduct research to specify the proposed solution and existing approaches.
- Prepare a "working document" and an agenda for the formal meeting.
- Conduct the meeting
- Jointly develop mini-specs that reflect data, functional, and behavioral features of the software. Alternatively, develop use cases that describe the software from the user's point of view.
- Review each mini-spec or use case for correctness, consistency, and lack of ambiguity.
- Assemble the mini-specs into a scoping document.
- Review the scoping document or collection of use cases with all concerned.
- Modify the scoping document or use cases as required.

## 3.13 John Reel defines 10 signs that indicate that an information systems project is in jeopardy :

- Software people don't understand their customer's needs.
- The product scope is poorly defined.
- Changes are managed poorly.
- The chosen technology changes.
- Business needs change [or are ill defined].
- Deadlines are unrealistic.
- Users are resistant.
- Sponsorship is lost [or was never properly obtained].
- The project team lacks people with appropriate skills.
- Managers [and practitioners] avoid best practices and lessons learned.

## 3.14 90-90 rule :

The first 90 percent of a system absorbs 90 percent of the allotted effort and time. The last 10 percent takes another 90 percent of the allotted effort and time

## 3.15 John Reel suggests a five-part commonsense approach to software projects :

- Start on the right foot.
- Maintain momentum
- Track progress.
- Make smart decisions.
- Conduct a postmortem analysis.

## 3.16 W5HH Principle

Barry Boehm suggests an approach that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources. He calls it the W5HH Principle,after a series of questions that lead to a definition of key project characteristics and the resultant project plan:

- **Why** is the system being developed?
- **What** will be done?
- **When** will it be done?
- **Who** is responsible for a function?
- **Where** are they located organizationally?
- **How** will the job be done technically and managerially?
- **How much** of each resource is needed?

Boehm's W5HH Principle is applicable regardless of the size or complexity of a software project. The questions noted provide you and your team with an excellent planning outline.

# 4 Process And Project Metrics

## 4.1 reasons that we measure software :

- to characterize
- to evaluate
- to predict
- to improve

## 4.2 Project metricsenable 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
- evaluate the project team’s ability to control quality of software work products.

## 4.3 Software metrics can be categorized to Direct and InDirect measures :

Direct measuresof the software process include cost and effort applied. Direct measures of the product 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, and many other “–abilities”

## 4.4 Size-Oriented Metrics

Size-oriented software metrics are derived by normalizing quality and/or productivity measures by considering thesizeof the software that has been produced.

## 4.5 simple size-oriented metrics

- Errors per KLOC (thousand lines of code)
- Defects per KLOC
- $ per KLOC
- Pages of documentation per KLOC
- Errors per person-month
- KLOC per person-month
- $ per page of documentation

## 4.6 Function-Oriented Metrics

Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. The most widely used function-oriented metric is the function point (FP). Computation of the function point is based on characteristics of the software's information domain and complexity.

## 4.7 Goal of software engineering :

The overriding goal of software engineering is to produce a high-quality system, application, or product within a time frame that satisfies a market need.

## 4.8 Measuring Quality

- Correctness
- Maintainability
- Integrity
- Usability

## 4.9 Correctness

A program must operate correctly or it provides little value to its users. Correctness is the degree to which the software performs its required function. The most common measure for correctness is defects per KLOC , defects are those problems reported by a user of the program after the program has been released for general use.

## 4.10 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, you must use indirect measures. A simple time-oriented metric is mean-time-to-change(MTTC), the time it takes to analyze the change request, design an appropriate modification, implement the change, test it, and distribute the change to all users. On average, programs that are maintainable will have a lower MTTC (for equivalent types of changes)

## 4.11 Integrity

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: programs, data, and documention.

### 4.12 Usability

If a program is not easy to use, it is often doomed to failure, even if the functions that it performs are valuable. Usability is an attempt to quantify ease of use and can be measured in terms of the characteristics

## 5 Estimation for Software Projects

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. Estimation carries inherent risk, and this risk leads to uncertainty.

**Project complexity** has a strong effect on the uncertainty inherent in planning. Complexity, however, is a relative measure that is affected by familiarity with past effort. The first-time developer of a sophisticated e-commerce application might consider it to be exceedingly complex. However, a Web engineering team developing its tenth e-commerce WebApp would consider such work run-of-the-mill.

**Project size** is another important factor that can affect the accuracy and efficacy of estimates. As size increases, the interdependency among various elements of the software grows rapidly. Problem decomposition, an important approach to estimating, becomes more difficult because the refinement of problem elements may still be formidable.

The **degree of structural uncertainty** also has an effect on estimation risk.

The availability of historical information has a strong influence on estimation risk.

If project scope is poorly understood or project requirements are subject to change, uncertainty and estimation risk become dangerously high.

## 5.1 Resources

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

three major categories of software engineering resources—people, reusable software components, and the development environment (hardware and software tools). Each resource is specified with four characteristics:

- description of the resource
- a statement of availability
- time when the resource will be required
- duration of time that the resource will be applied

The last two characteristics can be viewed as a time window.

## 5.2 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.

## 5.3 Bennatan suggests four software resource categories

- Off-the-shelf components
- Full-experience components
- Partial-experience components
- New components

## 5.4 Off-the-shelf components

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.

## 5.5 Full-experience components

Existing specifications, designs, code, or test data developed for past projects that are similar to the software to be built for the current project. Members of the current software team have had full experience in the application area represented by these components. Therefore, modifications required for full-experience components will be relatively low risk.

## 5.6 Partial-experience components

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. Members of the current software team have only limited experience in the application area represented by these components. Therefore, modifications required for partial-experience components have a fair degree of risk.

## 5.7 New components

Software components must be built by the software team specifically for the needs of the current project .

## 5.8 Environmental Resources

The environment that supports a software project, often called the software engineering environment(SEE), incorporates hardware and software. Hardware provides a platform that supports the tools (software) required to produce the work products that are an outcome of good software engineering practice.