Software: Computer programs and associated documentation is called software.
Software products may be developed for a particular customer or may be developed for a general market.

Software Engineering: Software engineering is an engineering discipline that is concerned with all aspects of software production.

Difference between software engineering and computer science:
Computer science is concerned with theory and fundamentals.
Software engineering is concerned with the practicalities of developing and delivering useful software.

Difference between software engineering and system engineering:
System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering.
Software engineering is a part of that process.

Different types of software

1. System software
2. Personal Computer software
3. Web-based software
4. Embedded Software
5. Real-time software
6. Business software
7. Accounting
8. Educational
9. Artificial intelligence software
10. Scientific software

Software process: A set of activities whose goal is the development or evolution of the software.

Software process model: It is a simplified representation of a software process presented from a specific perspective.

Cost of software engineering: Roughly 60% of the costs are development costs, 40% of the costs are testing costs.
For custom software, evolution costs often exceed development costs.

Software engineering methods: Structural approaches to software development which include system models, notations, rules, design advice, and process guidance.

Software process: A software process is the set of activities and associated rules that produce a software product.
There are 4 fundamental process activities that are common for all software processes:

1. Software specification: where customer and engineer define the software to be produced and the constraints on its operations.
2. Software development: where the software is designed and programmed.
3. Software validation: where the software is checked to ensure that this is what the customer required.
4. Software evolution: where the software is modified to adapt it to changing customer and market requirements.

Process model: A software process model is a simplified description of a software process that represents one view of the process.

Approach:

1. A workflow model
2. A dataflow model
3. A role/action model

General model:

1. A waterfall approach
2. Iterative development
3. Component-based software engineering

Attributes of good software:
The software should deliver the required functionality and performance to the user and should be maintainable, dependable, and usable.

Product characteristics	Description
maintainability	Software should be written in such a way that it may evolve to meet the changing needs of customers, this is a critical attribute because software change is an inevitable consequence of a changing business environment
Dependability	Software dependability has a range of characteristics; including reliability, security, and safety, dependable software should not cause any physical & economical damage in the event of a system failure
Efficiency	Software should not make wasteful use of system resources such as

	memory and processor cycle efficiency, therefore, includes responsiveness, processing time, memory utilization
usability	Software must be usable, without undue effort, by the type of user for whom it is designed. This means that it should have an appropriate user interface and adequate documentation

Software key challenges:

1. Coping with increasing diversity (heterogeneity challenge)
2. Demands for using the delivery time (delivery challenge)
3. Developing trustworthy software (trust challenge)

CASE (computer-aided software engineering): software systems that are intended to provide automated support for software process activities are called computer-aided software engineering. CASE systems are often used for method support.

1. Definition phase
2. Development phase
3. Support phase

A generic view of software engineering:
Engineering is the analysis, design, construction, verification, and management of technical entities.

The work associated with software engineering can be categorized into three generic phases, regardless of application area, project size, or complexity. Each phase addresses one or more questions.

What does the definition phase focus on?
It is during the definition phase that the software engineers attempt to identify -
➡What information is to be processed
➡What functions and performances are desired
➡What system behavior can be expected
➡What interfaces are to be established
➡What design constraints exist
➡What validation criteria are required to define a successful system

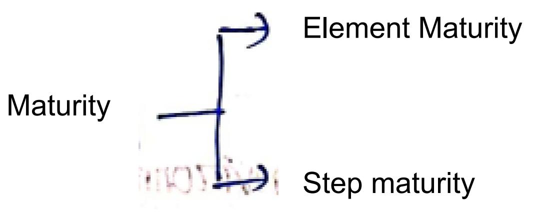
What does the development phase focus on?
It is during this phase that the software engineers attempt to define
➡How data are to be structured
➡How functions are to be implemented within a software architecture

➡How procedural details are to be implemented
➡How interfaces are to be characterized
➡How the design will be translated into a programming language
➡How the testing will be performed

The support phase focuses on quality assurance, testing:
It focuses on change associated with error corrections, adaptations required as the software environment changes and evolves due to environment brought about by changing requirements
The support phase reapplies the steps of the definition and development phases but does so in the context of existing software.
There are 4 types of change encountered in the support phase:

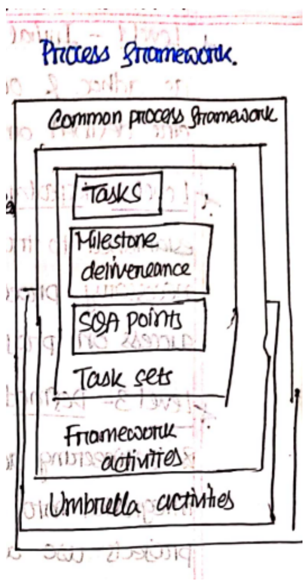
1. Correction: Even with the best quality assurance activities, it is likely that the customer will uncover defects in the software. Corrective maintenance changes the software to correct defects.
2. Adaptation: Over time the original environment (CPU, OS, Business rule, external product characteristics) for which the software was developed is likely to change. Adaptive maintenance results in a modification to the software to accommodate changes to its external environment.
3. Enhancements: As software is used, the customer/user will recognize additional functions that will provide benefit. Perfective maintenance extends the software beyond its original functional requirements.
4. Prevention: Computer Software deteriorates due to change and because of this, preventive maintenance, often called software reengineering, must be conducted to enable the software to serve the needs of its end-user.

In essence, preventive maintenance makes changes to computers and programs so that they can be more easily corrected, adapted, and enhanced.



Umbrella Activities
The small components of total software development are called umbrella activities.
⬇
SQA➡Software Quality Assurance

The software process:
A common process framework is established by defining a small number of framework activities that are applicable to all software projects regardless of their size and complexity.



A number of task-set: Each collection of software engineering tasks, project milestones, work products, and quality assurance points enable the framework activities to be adapted to the characteristics of the software project and the requirements of the software project team.
Umbrella activities: Such as software quality assurance, software configuration management, and measurement.

Process Maturity:
The software engineering approach provides a measure of global effectiveness of the software maturity level which is process maturity level and five process maturity levels are characterized in different manners.

Process maturity level 1:
Level 1 - Initial: The software process is characterized as ad hoc & occasionally even chaotic. Few processes are defined and success depends on individual effort.

Level 2 - Repeatable: Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier success on projects with similar applications.

Level 3 - Defined: The software process for both management and engineering activities are documented, standardized, and integrated into an organization-wide software process. All projects use a documented and approved version of the organization process for the development and support of software.

Level 4 - Managed: Detailed measures of the software process and product quality are collected. Both the software process and the product are quantitatively understood and controlled used detailed measures.

Level 5 - Optimization: Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technology.

Process maturity Level 2:
➡Software configuration management
➡Software quality assurance
➡Software subcontract management
➡Software project tracking and oversight
➡Software project planning
➡Requirement management

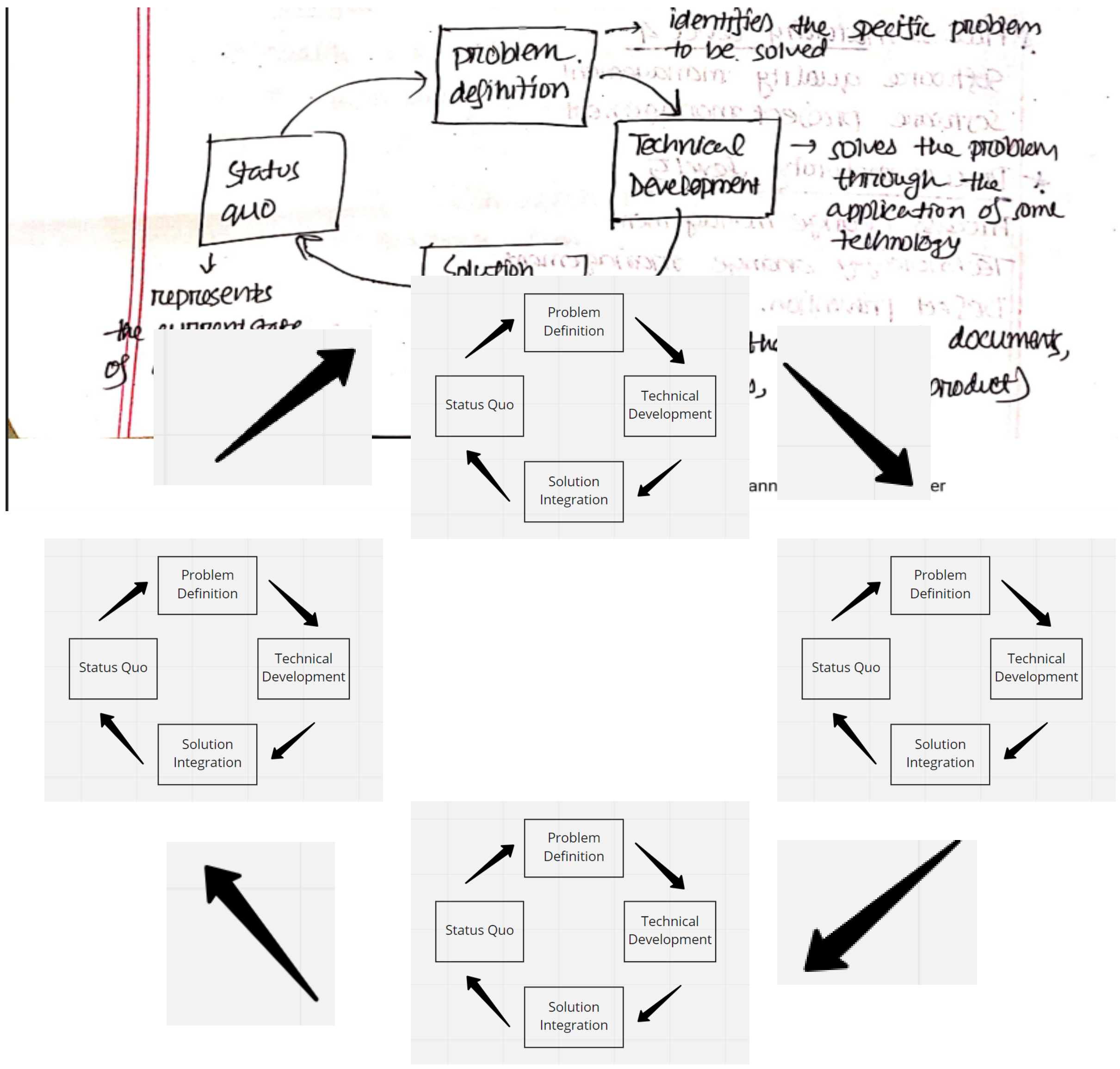
Process maturity level 3:
➡Peer review
➡Intergroup coordination
➡Software product engineering
➡Integrated software management
➡Training program
➡Organization process definition
➡Organization process focus

Process maturity level 4:
➡Software quality management
➡Software project management

Process maturity level 5:
➡Process change management
➡Technology change management
➡Defect prevention

Software engineering as a problem-solving loop:
To solve actual problems, a software engineer or a team of engineers must incorporate a development strategy that encompasses the process method and tools and generic phases. This strategy is often referred to as a process model or a software engineering paradigm. A process model for software engineering is chosen based on the nature of the project and the application, the method and tools to be used, and the controls and deliverables that are

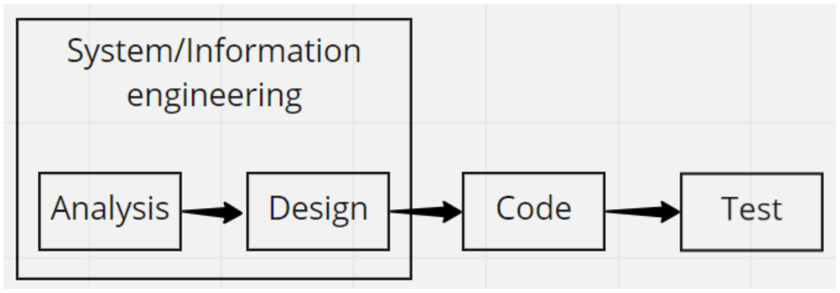
required.
All software development can be characterized as a problem-solving loop in which four distinct stages are encountered. Status quo, problem definition, technical development, and solution integration.



Some process models in Software engineering are:

1. The linear sequential
2. Prototyping
3. Concurrent
4. Win-win
5. RAD → Rapid Application Development
6. Spiral
7. Component-based concurrent

The linear sequential model
LSM, sometimes called the classic life cycle or waterfall model is the linear sequential model which suggests a systematic sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support.
The linear sequential model encompasses the following activities:
➡System/Information engineering and modeling
➡Software requirement analysis
➡Design
➡Code generation
➡Testing
➡Support



System/Information engineering & modeling:
Software is always a part of a larger system. Work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software. The system view is essential when software must interact with other elements such as hardware, people and datasets. System engineering and analysis encompass requirements gathering at the system level with a small amount of top-level design and analysis. Information gathering encompasses requirements gathering at the strategic business level and business area level.

System requirement analysis
The requirement gathering process is intensified and focused on special software. To understand the nature of the program to be built, the software engineers must understand the information domain for the software as well as required functions, behavior performance, and interface. Requirements for both the system and the software are documented and reviewed with the customer.

Design
Software design is actually a multistep process that focuses on 4 distinct attributes of a program: data structure, software architecture, interface representation & procedural details(algorithm). The design process translates requirements in a presentation of the software that can be assessed for quality before coding begins. The design is documented and becomes part of the software configuration.

Code generation
The design must be translated into a machine-readable format. The code generation step performs this work. If the design is performed in a detailed manner, code generation can be accomplished manually.

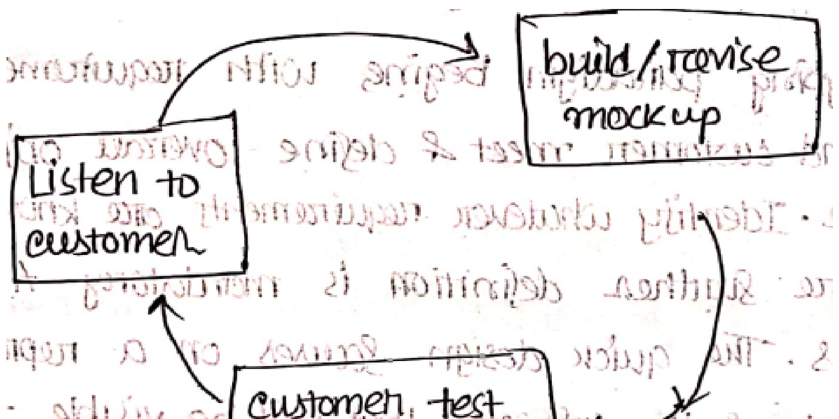
Testing

Once code has been generated, program testing begins. The testing process focuses on the logical intervals of the software, ensuring that all statements have been tested and on the functional externals that are to uncover errors. And ensure that defined input will produce actual results that agree with required results/outputs.

Support
The software will undoubtedly undergo change after it is delivered to the customer. Changes will occur when errors have been encountered because the software must be adapted to accommodate changes in its external environment.

Prototyping model
>>No sequential process is followed
>>SRb (Software Requirement Specification) is created
>>When the customer doesn't know what the requirements are

Often a customer defines a general objective for software but doesn't identify detailed input processing and output requirements. In other cases, the developer may be unsure of the efficacy of the algorithm, the adaptability of an operating system, or the form that human/machine interaction should take. In these and many other situations, a prototyping paradigm may offer the best approach.
The prototyping paradigm begins with gathering requirements. Developers and customers meet and define overall objectives for the software, identify whatever requirements are known, and outline areas where a further definition is mandatory. A quick design occurs after, which focuses on a representation of those aspects of the software that will be visible to the user/customer(input approach and output format). The quick design leads to the construction of a product. The prototype is evaluated by the customer/user and used to refine the requirements of the software to be developed. Iteration occurs on the prototype, tuned to satisfy the needs of the customer, at the same time enabling the developer to gain a better understanding of what needs to be done.
Ideally, the prototype should serve as a mechanism for identifying software requirements. If the working prototype is built, the developer attempts to use existing program software fragments or applies tools that enable working programs to be generated quickly.



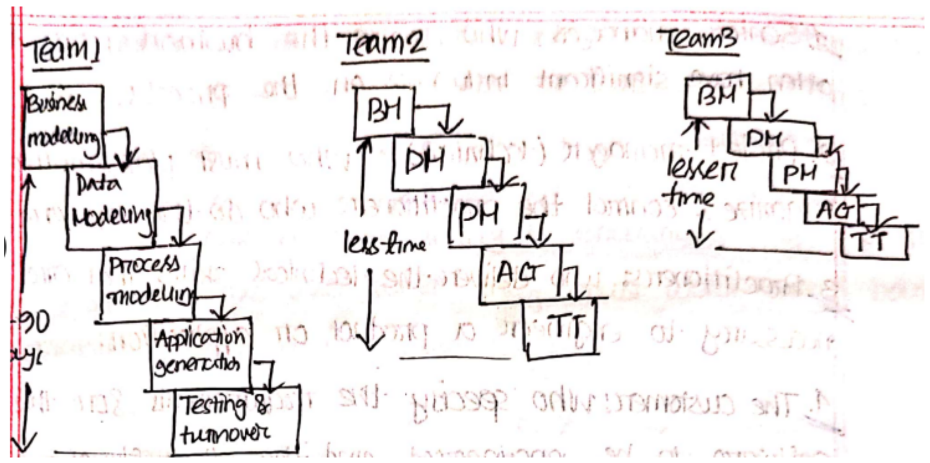
RAD model
*Compartmental
*Incremental
*RAD is a high-speed adaptation of the linear sequential model.
Rapid Application Development is an incremental software development process model that emphasizes an extremely short development cycle. Here rapid development is achieved by component-based construction if requirements are well defined and well understood and product scope is constrained, the RAD process enables a development team to create a fully functional system within a very short time period. The RAD approach encompasses the following phases:

1. Business model: The information flow among business functions is modeled in a way that answers the following questions
   >What information drives the business process?
   >What information is generalized?
   >Who generalizes it?
   >Where does the information go?
   >Who processes the information?

2. Data model: The information flow, defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics of each object are identified and the relationships between these objects are defined.
3. Process modeling: The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Process descriptions are created for adding, modifying, deleting, or retrieving a data project.
4. Application generation: RAD assumes the use of fourth-generation technologies. Rather than creating software using conventional third-generation programming languages, the RAD process works to reuse existing program components or create reusable

components. In all cases, automated tools are used to facilitate the construction of the software.

5. Testing and turnover: Since the RAD emphasizes on reuse many of the components have already been tested. These reduce overall testing time. However new components must be tested and all interfaces must be fully exercised.



People:

➡practitioners - who designs and tests the code
➡customer - who demands the project
➡end user - who uses the software at the end
➡nontechnical - manager
➡technical - who was approached by technical

The players:

The software process is populated by players who can be categorized into one of the five constituencies:

1. Senior Manager: Who defines the business issues that often have significant influences on the process
2. Project Manager (technical): Who must plan, motivate, organize and control the practitioners who do the software work.
3. Practitioners: Who delivers the technical skills that are necessary to engineer a product or application.
4. The customer: Who specifies the requirements for the software to be engineered and the other stakeholders who have a peripheral interest in the outcome.
5. End-user: Who interacts with the software once it is released for production use.

Team Leaders:

Project management is a people-intensive activity, and for this reason, competent practitioners may make poor team leaders.

A team leader may have the following leadership qualities:

1. Motivation: The ability to encourage (by push or pull) technical people to produce to their best ability.

2. <u>Organization</u>: The ability to mold existing procedures that will enable the initial concept to be translated into a final product.
3. <u>Ideas/Innovation</u>: The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software.
4. Problem Solving
5. Managerial Identity
6. Achievement
7. Influence and team building

The software team

m
m
t
k
people
task
team
year

There are almost as many organizational structures for software development as there are organizations that develop software.
For better or for worse, organizational structures cannot be easily modified. Concerned with the practical and political consequences of organizational change are not within the project manager's scope and responsibility.
However, the organization of the people directly involved in a new software project is within the project manager's purview. The following options are available for applying human measures to a project that will require 'n' people working for 'k' years.

1. 'n' individuals are assigned to 'm' different functional tasks, relatively little combined work occurs. Coordination is the responsibility of a software project manager who may have six other projects to be concerned with.
2. 'n' individuals are assigned to 'm' different functional tasks (m<n) so that informal teams are established and ad-hoc team leaders may be appointed. Coordination among teams is the responsibility of the software project manager.
3. 'n' individuals are organized into 't' teams. Each team is assigned one or more functional tasks; each team has a specific structure that is defined for all teams working on a project. Coordination is controlled by a team of software project managers.

Finally, the best team structure depends on the management style of the organization, the number of people who will populate the team and their skill leads, and overall problem difficulty.

Generic teams:
➡<u>Democratic decentralized (DD):</u>
This software engineer team has no permanent leader. Rather, task coordinators are appointed for short durations and then replaced by others who may coordinate different tasks. Decisions on problems and approaches are made by group consensus.
Communication among team members is horizontal.
➡<u>Controlled decentralized (CD):</u>
This software engineering team has a defined leader who coordinates specific tasks and secondary leaders that have responsibility for sub-tasks. Problem-solving maintains a group

activity, but the implementation of the solution is partitioned among subgroups by team leaders. Communication among subgroups and individuals is horizontal.
Vertical communication among the central hierarchy also occurs.
➡Controlled centralized (CC):
Top-level problem solvers and internal coordinators are managed by team leaders.
Communication between the leader and team members is vertical.

Organizational paradigms:

1. Closed Paradigm
2. Random Paradigm
3. Open Paradigm
4. Synchronous Paradigm

Coordination & communication
➡Formal, impersonal approaches: Includes software engineering documents and deliverable, technical memos, project milestones, schedules, and project control tools, change requests and related documentation, error tracking reports, and repository data
➡Formal, interpersonal approaches: Focuses on quality assurance activities applied to software engineering work products. This includes - status review meetings, design and code inspection
➡Informal, interpersonal approach: Includes group meetings for information dissemination and problem-solving and coordination of requirement and development staff.

Coordination & communication issues
➡Electronic communication: Encompasses electronic mail, electronic bulletin boards, and by extension video-based conferencing systems
➡Interpersonal networking: Includes informed discussions with team members and those outside, who may have experience or that can assist team members.

Project
A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance.
In order to manage successful projects, we must understand what can go wrong and how to do it right. There are some significant signs that indicate an information system project is in jeopardy.

1. Software people don't understand their customer's needs
2. The product scope is poorly designed
3. Changes are managed poorly
4. The chosen technology changes
5. Business needs change [or are ill defined]
6. Deadlines are unrealistic

7. Users are resistant
8. Sponsorship is lost [or was never properly obtained]
9. The project team lacks people with appropriate skills
10. Managers and practitioners avoid best practices and lessons learned

There are five-part common-sense approaches to software projects.

1. Start off on the right foot: This is accomplished by working hard (very hard) to understand the problem to be solved and then setting realistic objects and expectations for everyone who will be involved in the project. It is reinforced by building the right team and giving the team the autonomy, authority, and technology needed to do the job
2. Maintaining momentum: Many projects get off to a good start and then disintegrate. To maintain momentum, the project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team that emphasizes quality in every task it performs, and management should do everything possible to stay out of the team’s way.
3. Track progress; For a software project, progress is tracked as a work product (eg: specification, sources, code, sets of test cases) as a part of quality assurance activity. In addition software process and product measures can be collected and used to assess progress against average development for the software development organization.
4. Make smart decisions: In essence, the decision of the project manager and the software team should be to ‘keep it simple’. Whenever possible, decide to use commercial off-the-shelf software and existing components, and decide to avoid custom interfaces. When standard approaches are available; decide to identify and then avoid obvious risks, and decide to allocate more time than you think is necessary for complex or risky tasks.
5. Conduct a post mortem analysis: Establish a consistent mechanism for extracting lessons learned for each project. Evaluate the planned and actual schedules, collect and analyze software project metrics, get feedback from team members and customers, and record findings in written form.

Principles that guide the process:

1. Be agile
2. Focus on quality at every step
3. Be ready to adapt
4. Build an effective team
5. Establish mechanisms for communication and coordination
6. Manage change
7. Assess Risk

Measurement

Software measurement can be categorized into two classes

1. Direct Measures: Direct measures of the software engineering process includes cost and effort applied. Direct measures of the product include lines of code produced (LOC), execution speed, memory size, and defects reported over the same set period of time.

2. Indirect measures: These measures include functionality, quality, complexity, efficiency, reliability, and maintainability.

There are some metrics for the measurement of software products.

1. Size oriented metrics
2. Function oriented metrics
3. Extended function point metrics

Function oriented metrics

Function-oriented 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. Function-oriented metrics measure some points called function points. Function points are derived using an empirical relationship based on the countable (direct) measure of software's information domain and assessment of software complexity.

Function points are computed by completing a table. Five information domain characteristics are determined and counts are provided in the appropriate table location. Information domain values are defined in the following manner.

▸Number of user inputs: Each user input that provides distinct application-oriented data to the software is concerned. Input 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 online input that results in the generation of some functional values. Software response is the true form of online output. Each distinct query is counted.

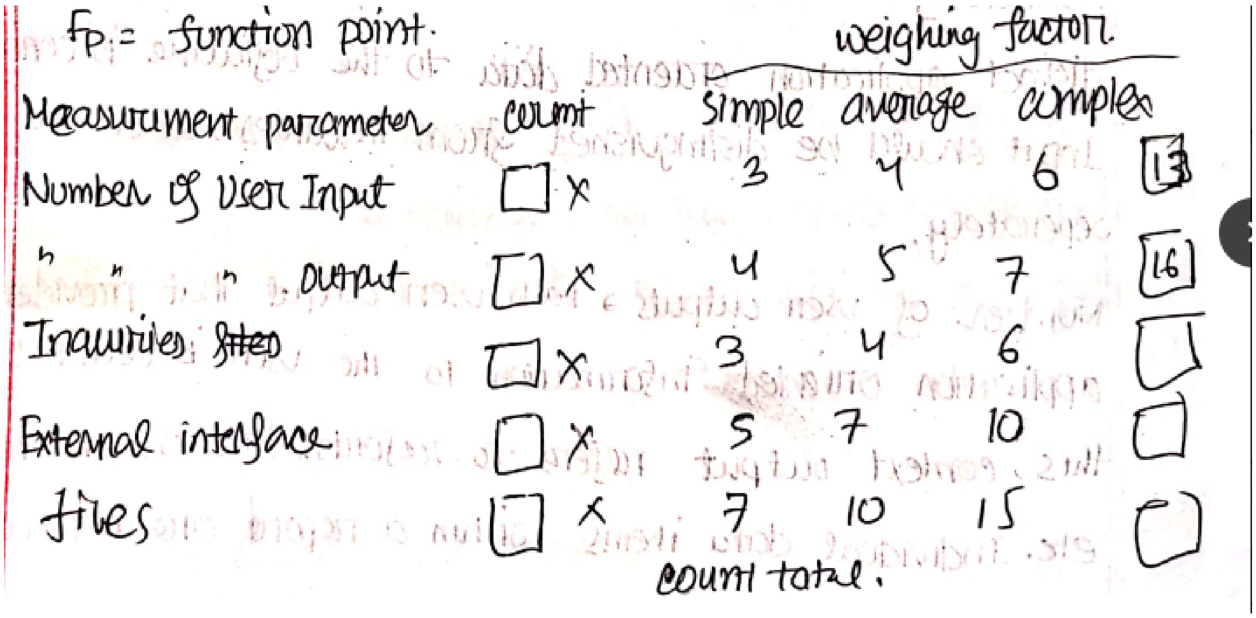
▸Number of files: Each logical master file meaning a logical grouping of data that may be one part of a larger database, may be counted.

▸Number of external interfaces: All machine-readable interfaces i.e data files on storage media that are used to transport information are counted.

F.P = count total X $[0.05 + 0.01 \times \Sigma(f_i)]$

F_i = complexity adjustment value

F = Function point



FP = function point.

Measurement parameter	count		weighing factor simple	average	complex	
Number of User Input	□	×	3	4	6	13
" " " output	□	×	4	5	7	16
Inquiries; ~~Steo~~	□	×	3	4	6	□
External interface;	□	×	5	7	10	□
files	□	×	7	10	15	□
			count total:			

Quality

Quality may be defined as a character or attribute of a product or something.

As an attribute of an item, quality refers to measurable characteristics. Two types of quality may be encountered:-

1. <u>Quality of design</u> refers to a characteristic that designers specify for an item. The grade of material, tolerance, and performance specifications all contribute to the quality of design. As higher-grade materials are used/designed, higher tolerance and greater levels of performance are specified, the design quality of a product increases if the product is manufactured according to specification.
2. <u>Quality of conformance</u> is the degree to which the design specifications are followed during manufacture. The greater the degree of conformance, the higher the quality of conformance.
   In software development, quality of design encompasses requirements, specifications, and the design of the system.
   Quality of conformance is an issue focused primarily on implementation. If the implementation follows the design and the resulting system meets its requirements and performance goals, conformance goals are met.

Quality control

Quality control involves a series of inspections, reviews, and tests throughout the software process to ensure each work product meets the requirements placed upon it. Quality control activities may be fully automated, entirely manual, or a combination of automated tools and human interaction.

Quality assurance

Quality assurance consists of the auditing and the reporting function of management. The goal of quality assurance is to provide management with data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goal.

Cost of quality
Cost of quality includes all costs incurred in the pursuit of quality or in the performance of quality-related activities. Cost of quality studies is conducted to provide a baseline for the current cost of quality, identify opportunities for reducing the cost of quality and provide a normalized base of companies.

Metrics for Software Quality
The overriding goal of software engineering is to produce a high-quality system, application or product. To achieve this goal, software engineers must apply effective methods coupled with modern tools within the context of a mature software process.
The quality of a system, application or product is only as good as the requirements that describe the problem, the design that models the solution, the code that leads to an executable program and the tests that exercise the software to uncover errors.

An overview of Factors that Affect Quality
The factors assess software from three distinct points of view: (1) product operation (using it), (2) product revision (changing it), and (3) product transition (modifying it to work on a different environment i.e "porting it").
The relationship between these quality factors (what they call a framework) and other aspect of software engineering process:
First, the framework provides a mechanism for the project manager to identify what qualities are important. These qualities are attributes of the software in addition to its functional correctness and performance which have life cycle implications. Such factors as maintainability and portability have been shown in recent years to have significant life cycle cost impact...
Secondly, the framework provides a means for quantitatively assessing how well the development is progressing relative to the quality goals established...
Thirdly, the framework provides for more interaction of QA personnel throughout the development effort…
Lastly, … quality assurance personal can use indications of poor quality to help identify [better] standards to be enforced in the future.

Measuring Quality
Although there are many measures of software quality, correctness, maintainability, integrity and usability provide useful indicators for the project team. Glib suggests definitions and measures for each.

<u>Correctness</u>**:** 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, where a defect is defined as a verified lack of conformance to requirements. When considering the overall quality of a software

product, defects are those problems reported by a user of the program after the program has been released for general use. For quality assessment purposes, defects are counted over a standard period of time, typically one year.
Maintainability: Software maintenance accounts for more effort than any other software engineering activity. 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 measure. 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) than programs that are not maintainable.
Hitachi has used a cost-oriented metric for maintainability called spoilage- the cost to correct defects encountered after the software has been released to end users. When the ratio of spoilage to overall project cost(for many projects) is plotted as a function of time, a manager can determine whether the maintainability of software produced by a software development organization is improving. Actions can then be taken in response to the insight gained from this information.
Integrity: Software integrity has become increasingly important in the age of hackers and firewalls. The 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 documents.
To measure integrity, two additional attributes must be defined: threat and security. Threat is the probability(which can be estimated or derived from empirical evidence) that an attack of a specific type will occur within a given time. Security is the probability(which can be estimated or derived from empirical evidence) that the attack of a specific type will be repelled. The integrity of a system can be defined as:

integrity=summation[(1 - threat) X (1 - security)]

Where threat and security are summed over each type of attack.
Usability: The catch phrase "user-friendliness" has become ubiquitous in discussion of software products. 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.
4. A subjective assessment (sometimes obtained through a questionnaire) of users' attitudes toward the system.