**Software Process** is a structured set of activities to develop a software system

**Software process Steps:**

* + **Specification** – defining what the system should do;
  + **Design and implementation** –defining the organization of the system and implementing the system;
  + **Validation** – checking that it does what the customer **wants;**
  + **Evolution** –changing the system in response to changing customer needs.

**Software Process Model** is an abstract representation of a process.

**The waterfall model**

* Plan-driven model.
* **Requirements analysis** (Elicitation, specification & validation) ->**System design** -> **Implementation** -> **Integration** -> **Maintenance**
* Is very inflexible to changes

**Incremental development**

* Specification, development and validation are interleaved.
* Flexible
* Easy to get customer feedback
* Process & progress not visible
* Structure degrades as new increments are added*.*

**Integration and configuration**

* The system is assembled from existing configurable components. May be plan-driven or agile.
* Reuse software components i.e. commercial off the shelf
* Less costly and risky
* Fast delivery
* Requirements may be compromised
* Loss of control on evolution of components

**Verification and validation** is intended to show that a system conforms to its specification and meets the requirements of the system customer.

**Component testing**: Individual components are tested independently;

**System testing**: Testing of the system as a whole.

**Customer testing** : Testing with customer data to check that the system meets the customer’s needs.

**Change anticipation**, where the software process includes activities that can anticipate possible changes before significant rework is required.

**Change tolerance**, where the process is designed so that changes can be accommodated at relatively low cost.

**System prototyping**, where a version of the system or part of the system is developed quickly to check the customer’s requirements and the feasibility of design decisions:

* Improved system usability.
* A closer match to users’ real needs.
* Improved design quality.
* Improved maintainability.
* Reduced development effort.

**Incremental development:** Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;

**Incremental delivery:** Deploy an increment for use by end-users;

* Customer value and interest can be measured
* Early increments act as a prototype to help elicit requirements for later increments.
* Lower risk of overall project failure.
* Allows more testing

**Process measurement:** measure one or more attributes of the software process or product.

**Process analysis:** Identify weaknesses and bottlenecks of current process.

**Process change:** Process changes are proposed to address some of the identified process weaknesses.

**Software evolution** takes place when you change existing software systems to meet new requirements.

**AGILE DEVELOPMENT PRINCIPLES:**

**Customer Involvement:** Customers should be closely involved throughout the development process

**Incremental Delivery:** The software is developed in increments with the customer specifying the requirements to be included in each increment

**People Not Process:** Use of skilled people and their skills should be used and allowed to develop their favorable ways to work

**Embrace Change:** Design system to accommodate any changes in requirements

**Maintain Simplicity:** Simple process and implementation standards

**Extreme Programming**:

* **User stories**
* **Test First automate test cases, incremental test development, User involvement in testing**
* **Refactoring to keep code simple**
* **Pair programming** allows sharing knowledge and understanding of code

**PROS OF TEST FIRST:**

* Simplified debugging
* Regression Testing on whole system
* Test Documentation
* Incremental testing
* Code coverage

**CONS OF TEST FIRST:**

* Programmers may skip certain automated tests
* Some tests are difficult to write incrementally
* Tests may not cover all

**Scrum** is an agile method that focuses on managing iterative development rather than specific agile practices. The whole team attends short daily meetings (Scrums) where all team members describe their progress since the last meeting, problems that have arisen and what is planned for the following day

* The initial phase is an outline planning phase where you establish the general objectives for the project.
* Second: series of sprint cycles, where each cycle develops an increment of the system.
* The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals.

**Scrum Advantages:**

* The product is broken down in small chunks.
* Unstable requirements don’t stop progress.
* Team communication is improved
* On-time delivery of increments
* Trust between customers and developers is established

**Scaling up** is using agile methods for developing large software systems that cannot be developed by a small team.

**Scaling out** is how agile methods can be introduced across a large organization with many years of software development experience.

**AGILE ISSUES:**

* The informality of agile is not compatible with large companies
* Agile methods are most appropriate for new software development rather than software maintenance.
* Agile methods are designed for small co-located teams and projects
* Lack of documentation
* Relies on skills of development team

**Brownfield development:** System interacts and uses preexisting systems

**Greenfield Development:** Entirely new system and doesn’t use preexisting systems

**Software testing:** Concerned with exercising and   
observing product behaviour

**Validation testing (Positive):** To demonstrate that the software meets its requirements

**Defect testing (Negative):** To discover faults or defects in the software

**Verification:** "Are we building the product right”. The software should conform to its specification.

**Validation:** "Are we building the right product”. The software should do what the user really requires.

**Software inspections:** Concerned with analysis of the static system i.e. code & documentation

* Find errors in code
* Test incomplete versions
* Find program defects
* Cant check nonfunctional issues

**Development testing**: system is tested during development.

* **Unit testing:** individual program units or object classes are tested.
* **Component testing:** several individual units are integrated to create composite components.
* **System testing**: where some or all of the components in a system are integrated and the system is tested as a whole.

**Release testing:** where a separate testing team test a complete version of the system before it is released to users.

**User testing**: where users or potential users of a system test the system in their own environment.

**Partition testing:** you identify groups of inputs that have common characteristics and should be processed in the same way.

**Guideline-based testing:** you use testing guidelines to choose test cases.

**Release testing** is testing a particular release of a system that is intended for use outside of the development team.

**Stress testing** is a form of performance testing where the system is deliberately overloaded to test its failure behavior.

**User or customer testing** is a stage in the testing process in which users or customers provide input on system testing.

* **Alpha testing**: Users of the software work with the development team to test the software at the developer’s site.
* **Beta testing**: A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
* **Acceptance testing**: Customers test a system to decide whether or not it is ready.

Testing can only show the presence of errors in a program. It cannot demonstrate that there are no remaining faults.

**Evolution:** Software systems is in operational use and is evolving as new requirements are proposed and implemented in the system.

**Servicing:** Only bug fixes and environmental adaption

**Phase-out:** The software may still be used but no further changes are made to it.

**CHANGE REQUEST PROCESS:** Propose Change -> Requirement Analysis -> Update Requirement -> Implement Change. **Urgent Changes skip the requirement analysis and updating**

**Legacy systems** are older systems that rely on languages and technology that are no longer used for new systems development.They include hardware, support software, application software’s, data, business rules, business process

**Replacing a legacy System:** Depends on the Cost, Quality and the Business value of the software

**Software Maintenance:**

* **Fault Repairs 24%**
* **Environmental Adaption 19%**
* **Functionality addition and modification 57%**

**Software Reengineering:** Restructuring or rewriting part or all of a legacy system without changing its functionality. -> Reduced Risk and costs. It involves Source code translation, reverse engineering, program modularisation, data engineering, and program structuring

**Refactoring** is the process of making improvements to a program to slow down degradation through change i.e. preventive maintenance.

Refactoring takes place throughout development whereas reengineering takes place after development.

**CODE ISSUES TO REFRACTOR**

* Duplicate code
* Long Methods
* Switch cases and if else
* Data clumping i.e. same data being used in multiple places
* Complexity

**SOFTWARE DEPENDABILITY:**

* **Availability:** The probability that the system will be up and running and able to deliver useful services to users.
* **Reliability:** The probability that the system will correctly deliver services as expected by users.
* **Safety**: A judgment of how likely it is that the system will cause damage to people or its environment.
* **Security:** A judgment of how likely it is that the system can resist accidental or deliberate intrusions.
* **Resilience:** A judgment of how well a system can maintain the continuity of its critical services in the presence of disruptive events such as equipment failure and cyberattacks.
* **Reparability:** Reflects the extent to which the system can be repaired in the event of a failure
* **Maintainability:** Reflects the extent to which the system can be adapted to new requirements;
* **Error** **tolerance:** Reflects the extent to which user input errors can be avoided and tolerated.

**Redundancy**: Keep more than a single version of critical components so that if one fails then a backup is available. For Availability

**Diversity**: Provide the same functionality in different ways in different components so that they will not fail in the same way. For Resilience

Increasing redundancy and diversity increases the system complexity making it less user friendly

**Dependable process**: A well-defined repeatable process is one that does not depend entirely on individual skills. **Auditable, Diverse, Documented, Robust, Standardized**

* + **Formal methods** are approaches to software development that are based on mathematical representation and analysis of software

**Verification Approach & Refinement Approach**

* + Formal specification;
  + Specification analysis and proof;
  + Transformational development;
  + Program verification.

**Benefits of Formal methods:**

* + Detail analysis of requirements
  + Can be used to analyze completeness and inconsistencies
  + Program testing costs may be reduced

**Why Not Formal methods**

* + Problem owners cannot understand a formal specification and so cannot assess if it is an accurate representation of their requirements.
  + It is easy to assess the costs of developing a formal specification but harder to assess the benefits.
  + Software engineers are unfamiliar with this approach and are therefore reluctant to propose the use of FM.
  + Formal methods are still hard to scale up to large systems.
  + Formal specification is not really compatible with agile development methods.

**Sociotechnical** **systems** include hardware, software and people.

**Security Engineering** Tools, techniques and methods to support the development and maintenance of systems to resist malicious attacks. **Confidentiality, Integrity, Availability**

* **Infrastructure security**, is concerned with maintaining the security of all systems and networks that provide an infrastructure.
* **Application** **security**, is concerned with the security of individual application systems.
* **Operational** **security**, which is concerned with the secure operation and use.

**Security management involves:** User & Permissions, Software deployment and maintenance, attack monitoring and recovery

**Interception threats** allow an attacker to gain access to an asset.

**Interruption threats** allow an attacker to make part of the system unavailable.

**Modification threats** allow an attacker to tamper with a system asset.

**Fabrication threats** allow an attacker to insert false information into a system.

**Security Assurance:**

* **Vulnerability Avoidance**
* **Attack detection and elimination**
* **Limited Exposure**

**Risk assessment and management** involves assessing the possible losses that might occur from attacks on the system and balancing these losses against the costs of security procedures that may reduce these losses.

**Identify Assets -> Value assets->Assess Exposure-> Identify threats->Attack assessment->Propose solutions->Feasibility assessment of solutions->Define security requirements**

* + **Preliminary risk assessment –** identify generic risks
  + **Life cycle / Development risk assessment** – during development
  + **Operational risk assessment** – risks from user behavior

**Secure Design Guidelines:**

* Follow Security Policy
* Avoid Single Failure point (Redundancy)
* Fail Securely (Resilience)
* Balance Security and usability
* Log User Actions
* Diverse and Redundant components
* Format and check inputs

**Security Testing is testing the extent to which the system can protect itself from external attacks.**

**Problems with security testing**

* + Security requirements are ‘shall not’ requirementsi.e. they specify what should not happen.
  + The people attacking a system are intelligent and look for vulnerabilities.
* **Experience-based testing:** The system is reviewed and analysed against the types of attack that are known to the validation team.
* **Penetration testing:** A team is established whose goal is to breach the security of the system by simulating attacks on the system.
* **Tool-based analysis:** Various security tools such as password checkers are used to analyse the system in operation.
* **Formal verification:** The system is verified against a formal security specification.

**Resilience Engineering** is concerned with limiting the costs of these failures and recovering from them. It assumes that good reliability engineering practices have been used. It involves:

* **Recognition:** the system or its operators should recognise early indications of system failure.
* **Resistance:** resistance strategies may be used to reduce the probability that the system will fail for example isolating core parts of the system
* **Recovery:** ensures that critical system services are restored quickly so that system users are not badly affected by failure.
* **Reinstatement** all system services are restored and normal system operation can continue.

**Swiss cheese Model:** Claims that each layer in security has loop holes and vulnerabilities. To increase security, increase the security layers.

Automatic Security process can detect problems, invoke cyberattack resistance if necessary and start automated recovery procedures. However, if the problem can’t be handled by the automated system, there are fewer people available to tackle the problem.

**ISOLATE CORE SYSTEMS WHEN ATTACK IS DISCOVERED**

**Software Reuse** involves reusing existing software’s, applications or components.

**PROS OF REUSE**

* Accelerated Development
* Use of specialist to integrate
* Increased dependability
* Low development cost
* Reduced risks
* Standardised system

**CONS OF REUSE**

* Costs of components/ software
* Understanding and finding appropriate software
* Increased maintenance
* Lack of tools
* Lack of control on evolution
* High cost for custom

**Software product lines** are applications with generic functionality that can be adapted and configured for use in a specific context to build further

**Frameworks** are a sub-system design made up of a collection of abstract and concrete classes and the interfaces between them

**An application system product** is a software system that can be adapted for different customers without changing the source code of the system. **ADVANTAGES**:

* + - Faster
    - Possible to see functionality of system
    - Avoid development risks
    - Allows to focus on core activities
    - Regular simple updates as it is **COTS**

**DISADVANTAGES**:

* Adopt requirements to COTS product
* The COTS product may be based on assumptions that are practically impossible to change.
* Difficult to choose the right one
* There may be a lack of local expertise to support systems development.
* The COTS product vendor controls system support and evolution.

**Configurable application systems** are generic application systems that are designed to support a particular business type, business activity or, sometimes, a complete business enterprise.

**An Enterprise Resource Planning** (ERP) system is a generic system that supports common business processes such as ordering and invoicing, manufacturing etc.

**Software Project Management:**

* **Software cannot be seen or touched.** Software project managers cannot see progress by simply looking at the artefact that is being constructed.
* **Many software projects are 'one-off'** projects and same ideas can’t be used in others
* **Software processes are variable** and organization specific.

**Management Activities:**

* Project Planning and scheduling
* Risk Management
* People Management
* Reporting
* Proposal Writing

**Risk management** is concerned with identifying risks and drawing up plans to minimise their effect on a project.

**People Management**: To keep all team members happy and motivated. A team has to be balanced with all types

* **Task-oriented**: The motivation for doing the work is the work itself;
* **Self-oriented**: The work is a means to an end which is the achievement of individual goals - e.g. to get rich, to play tennis, to travel etc.;
* **Interaction-oriented**: The principal motivation is the presence and actions of   
  co-workers.

**Project scheduling** is deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed.

* Split project into tasks and estimate time and resources required to complete each task.
* Organize tasks concurrently to make optimal use of workforce.
* Minimize task dependencies to avoid delays caused by one task waiting for another to complete.
* Dependent on project manager’s intuition and experience.

**Scheduling Difficulties:**

* Estimating the difficulty of problems and hence the cost of developing a solution is hard.
* Productivity is not proportional to the number of people working on a task.
* Adding people to a late project makes it later because of communication overheads.
* The unexpected always happens. Always allow contingency in planning.

**Milestones** are points in the schedule against which you can assess progress

**Deliverables** are work products that are delivered to the customer

**Story Based Planning:**

* The planning game is based on user stories that reflect the features that should be included in the system.
* The project team read and discuss the stories and rank those in order of the amount of time they think it will take to implement the story.
* Stories are assigned ‘effort points’ reflecting their size and difficulty of implementation
* The number of effort points implemented per day is measured giving an estimate of the team’s ‘velocity’
* This allows the total effort required to implement the system to be estimated

**Release Planning:**  select and refine stories that will be implemented in the next release

**Task planning:** Developers further break down stories into smaller tasks. This allows the whole team to get an overview of the iteration and Developers get a sense of ownership of these tasks to be completed

**In Agile, Delivery schedule is never extended. Features included in the release may be cut off**

**Estimation techniques:**

* **Experience-based techniques:** the estimated effort is done based on the managers past experience: **ISSUES:** Projects may not be similar at all, software development techniques change, new languages and platforms
* **Algorithmic cost modeling:** formula approach is used to compute the project effort based on estimates of product attributes, such as size, and process characteristics, such as experience of staff involved.

**Estimation models used in COCOMO II.**

* The application composition model,
* The early design model,
* The reuse model,
* The post-architecture model.

**EARLY DESIGN MODEL:** Used after requirements have been agreed

**APPLICATION COMPOSITION MODEL:** Used for prototypes and reuse

**REUSE MODEL:** Takes into account black-box code that is reused without change and code that has to be adapted to integrate it with new code.

* + **Black-box** reuse where code is not modified. An effort estimate (PM) is computed.
  + **White-box** reuse where code is modified. A size estimate equivalent to the number of lines of new source code is computed. This then adjusts the size estimate for new code.

**Multipliers** reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.

**Product attributes:** Concerned with required characteristics of the software product being developed.

**Computer attributes:** Constraints imposed on the software by the hardware platform.

**Personnel attributes:** Multipliers that take the experience and capabilities of the people working on the project into account.

**Project attributes:** Concerned with the particular characteristics of the software development project.

The price charged for a system does not just depend on its estimated development costs and the profit required by the development company. Organizational factors may mean that the price is increased to compensate for increased risk or decreased to gain competitive advantage.   
**QULAITY MANAGEMENT IS CONCERNED WITH:**

* Organizational level, quality management is concerned with establishing a framework of organizational processes and standards that will lead to high-quality software.
* Project level, quality management involves the application of specific quality processes and checking that these planned processes have been followed.
* Quality management is also concerned with establishing a quality plan for a project. The quality plan should set out the quality goals for the project and define what processes and standards are to be used

**The quality plan** should therefore define the most important quality attributes for the software that is being developed.

Quality managers should aim to develop a ‘quality culture’ where everyone responsible for software development is committed to achieving a high level of product quality.

**Product standards:** Apply to the software product being developed. They include document standards, such as the structure of requirements documents, documentation standards, such as a standard comment header for an object class definition, and coding standards, which define how a programming language should be used.

**Process standards:** These define the processes that should be followed during software development. Process standards may include definitions of specification, design and validation processes, process support tools and a description of the documents that should be written during these processes.

**ISSUES WITH STANDARDS**

* They may not be seen as relevant and up-to-date by software engineers.
* They often involve too much bureaucratic form filling.
* If they are unsupported by software tools, tedious form filling work is often involved to maintain the documentation associated with the standards.

**The ISO 9001** standard is a framework for developing software standards. It sets out general quality principles, describes quality processes in general and lays out the organizational standards and procedures that should be defined. These should be documented in an organizational quality manual

* **Pre-review activities**: Pre-review activities are concerned with review planning and review preparation
* **The review meeting**: During the review meeting, an author of the document or program being reviewed should ‘walk through’ the document with the review team.
* **Post-review activities**: These address the problems and issues that have been raised during the review meeting.

**ISSUES WITH PAIR PROGRAMMING:**

***Mutual misunderstandings***: Both members of a pair may make the same mistake in understanding the system requirements. Discussions may reinforce these errors.

***Pair reputation***: Pairs may be reluctant to look for errors because they do not want to slow down the progress of the project.

***Working relationships***: The pair’s ability to discover defects is likely to be compromised by their close working relationship that often leads to reluctance to criticize work partners.

**Software Metric:** Any type of measurement which relates to a software system, process or related documentation

* **Time for a particular process**
* **Resources needed for a process**
* **Number of occurrences of an event**

**Metrics are used:**

* To assign a value to system quality attributes
* To identify the system components whose quality is sub-standard

**Dynamic metrics** are closely related to software quality attributes

**Static metrics** have an indirect relationship with quality attributes

**ISSUES WITH MEASUREMENTS:**

* It is impossible to quantify the return on investment of introducing an organizational metrics program.
* There are no standards for software metrics or standardized processes for measurement and analysis.
* In many companies, software processes are not standardized and are poorly defined and controlled.
* Most work on software measurement has focused on code-based metrics and plan-driven development processes. However, more and more software is now developed by configuring ERP systems or COTS.
* Introducing measurement adds additional overhead to processes.

**CODE METRICS:**

**Fan-in/ Fan-out** = Numebr of times a function is called and the number of function calls made inside a function

Length of code

Length of variables

Depth of conditional nesting

Cyclomatic complexity

Inheritance tree size

Number of children

Coupling in classes

***Software analytics*** *is analytics on software data for managers and software engineers with the aim of empowering software development individuals and teams to gain and share insight from their data to make better decisions.*

* + **Version management**: Keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other.
  + **System building**: The process of assembling program components, data and libraries, then compiling these to create an executable system.
  + **Change management**: Keeping track of requests for changes to the software from customers and developers, working out the costs and impact of changes, and deciding the changes should be implemented.
  + **Release management**: Preparing software for external release and keeping track of the system versions that have been released for customer use.

A **codeline** is a sequence of versions of source code with later versions in the sequence derived from earlier versions.

**Baselines** A baseline is a collection of component versions that make up a system

**Pros of CI**

* + The advantage of continuous integration is that it allows problems caused by the interactions between different developers to be discovered and repaired as soon as possible.
  + The most recent system in the mainline is the definitive working system.

**Cons of CI**

* + If the system is very large, it may take a long time to build and test, especially if integration with other application systems is involved.
  + If the development platform is different from the target platform, it may not be possible to run system tests in the developer’s private workspace.

**System releases** include executable code, data files, configuration files and documentation. Release management involves making decisions on system release dates, preparing all information for distribution and documenting each system release

**Holistic system design:** Interaction between all components and layers in a system

**Quality plan structure**

* + Product introduction;
  + Product plans;
  + Process descriptions;
  + Quality goals;
  + Risks and risk management.

**Software engineering** is a field largely concerned with the application of engineering processes to the creation, maintenance, and design of software for a variety of different purposes.

**Computer science** takes a broad approach to the study of the principles and use of computers that covers both theory and application. This field involves the understanding and application of both abstract and concrete knowledge.

**A thin client** is designed to be especially small so that the bulk of the data processing occurs on the server.

**A thick client** (also called a fat client) is one that will perform the bulk of the processing in client/server applications.

**Latent conditions** are vulnerabilities and weaknesses in a system that, at some stage, may contribute to system failure.

**Active failures**: some operational event or human error that triggers a sequence of events that could lead to system failure.

**Coping with failures**: Design operational processes to be flexible and adaptable

**Recognition strategy** for detecting the symptoms of a problem that may lead to system failure.

**Resistance strategy** invoking actions that reduce the probability that a system will fail after a problem of cyberattack has been detected.

**Recovery strategy:** Critical services are restored as quickly as possible so that the consequences of a failure or cyberattack are minimized.

**Reinstatement strategy:** All system services are restored and the system brought back to normal operation

**Cybersecurity:** Sociotechnical issue, protection of citizens**,** protection of businesses**,** protection of critical infrastructures

**Cybersecurity** **threats**, Threats to confidentiality, integrity and availability of assets

**Swiss cheese model**: claims that defensive layers have vulnerabilities, Vulnerabilities are dynamic, Failure occurs when holes line up

ERP: Enterprise Resource Planning system

1. **PUBLIC** – Software engineers shall act consistently with the public interest.

2. **CLIENT AND EMPLOYER** – Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3. **PRODUCT** – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. **JUDGMENT** – Software engineers shall maintain integrity and independence in their professional judgment.

5. **MANAGEMENT** – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6. **PROFESSION** – Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

7. **COLLEAGUES** – Software engineers shall be fair to and supportive of their colleagues.

8. **SELF** – Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

**Principle 1: PUBLIC**

Software engineers shall act consistently with the public interest. In particular, software engineers shall, as appropriate:

1.01. Accept full responsibility for their own work.

1.02. Moderate the interests of the software engineer, the employer, the client and the users with the public good.

1.03. Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good.

1.04. Disclose to appropriate persons or authorities any actual or potential danger to the user, the public, or the environment, that they reasonably believe to be associated with software or related documents.

1.05. Cooperate in efforts to address matters of grave public concern caused by software, its installation, maintenance, support or documentation.

1.06. Be fair and avoid deception in all statements, particularly public ones, concerning software or related documents, methods and tools.

1.07. Consider issues of physical disabilities, allocation of resources, economic disadvantage and other factors that can diminish access to the benefits of software.

1.08. Be encouraged to volunteer professional skills to good causes and contribute to public education concerning the discipline.

**Principle 2: CLIENT AND EMPLOYER**

Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest. In particular, software engineers shall, as appropriate:

2.01. Provide service in their areas of competence, being honest and forthright about any limitations of their experience and education.

2.02. Not knowingly use software that is obtained or retained either illegally or unethically.

2.03. Use the property of a client or employer only in ways properly authorized, and with the clients or employer’s knowledge and consent.

2.04. Ensure that any document upon which they rely has been approved, when required, by someone authorized to approve it.

2.05. Keep private any confidential information gained in their professional work, where such confidentiality is consistent with the public interest and consistent with the law.

2.06. Identify, document, collect evidence and report to the client or the employer promptly if, in their opinion, a project is likely to fail, to prove too expensive, to violate intellectual property law, or otherwise to be problematic.

2.07. Identify, document, and report significant issues of social concern, of which they are aware, in software or related documents, to the employer or the client.

2.08. Accept no outside work detrimental to the work they perform for their primary employer.

2.09. Promote no interest adverse to their employer or client, unless a higher ethical concern is being compromised; in that case, inform the employer or another appropriate authority of the ethical concern.

**Principle 3: PRODUCT**

Software engineers shall ensure that their products

and related modifications meet the highest professional standards possible. In particular, software engineers shall, as appropriate:

3.01. Strive for high quality, acceptable cost and a reasonable schedule, ensuring significant tradeoffs are clear to and accepted by the employer and the client, and are available for consideration by the user and the public.

3.02. Ensure proper and achievable goals and objectives for any project on which they work or propose.

3.03. Identify, define and address ethical, economic, cultural, legal and environmental issues related to work projects.

3.04. Ensure that they are qualified for any project on which they work or propose to work by an appropriate combination of education and training, and experience.

3.05. Ensure an appropriate method is used for any project on which they work or propose to work.

3.06. Work to follow professional standards, when available, that are most appropriate for the task at hand, departing from these only when ethically or technically justified.

3.07. Strive to fully understand the specifications for software on which they work.

3.08. Ensure that specifications for software on which they work have been well documented, satisfy the users’ requirements and have the appropriate approvals.

3.09. Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project on which they work or propose to work and provide an uncertainty assessment of these estimates.

3.10. Ensure adequate testing, debugging, and review of software and related documents on which they work.

3.11. Ensure adequate documentation, including significant problems discovered and solutions adopted, for any project on which they work.

3.12. Work to develop software and related documents that respect the privacy of those who will be affected by that software.

3.13. Be careful to use only accurate data derived by ethical and lawful means, and use it only in ways properly authorized.

3.14. Maintain the integrity of data, being sensitive to outdated or flawed occurrences.

3.15 Treat all forms of software maintenance with the same professionalism as new development.

**Principle 4: JUDGMENT**

Software engineers shall maintain integrity and independence in their professional judgment. In particular, software engineers shall, as appropriate:

4.01. Temper all technical judgments by the need to support and maintain human values.

4.02 Only endorse documents either prepared under their supervision or within their areas of competence and with which they are in agreement.

4.03. Maintain professional objectivity with respect to any software or related documents they are asked to evaluate.

4.04. Not engage in deceptive financial practices such as bribery, double billing, or other improper financial practices.

4.05. Disclose to all concerned parties those conflicts of interest that cannot reasonably be avoided or escaped.

4.06. Refuse to participate, as members or advisors, in a private, governmental or professional body concerned with software related issues, in which they, their employers or their clients have undisclosed potential conflicts of interest.

**Principle 5: MANAGEMENT**

Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance. In particular, those managing or leading software engineers shall, as appropriate:

5.01 Ensure good management for any project on which they work, including effective procedures for promotion of quality and reduction of risk.

5.02. Ensure that software engineers are informed of standards before being held to them.

5.03. Ensure that software engineers know the employer’s policies and procedures for protecting passwords, files and information that is confidential to the employer or confidential to others.

5.04. Assign work only after taking into account appropriate contributions of education and experience tempered with a desire to further that education and experience.

5.05. Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project on which they work or propose to work, and provide an uncertainty assessment of these estimates.

5.06. Attract potential software engineers only by full and accurate description of the conditions of employment.

5.07. Offer fair and just remuneration.

5.08. Not unjustly prevent someone from taking a position for which that person is suitably qualified.

5.09. Ensure that there is a fair agreement concerning ownership of any software, processes, research, writing, or other intellectual property to which a software engineer has contributed.

5.10. Provide for due process in hearing charges of violation of an employer’s policy or of this Code.

5.11. Not ask a software engineer to do anything inconsistent with this Code.

5.12. Not punish anyone for expressing ethical concerns about a project.

**Principle 6: PROFESSION**

Software engineers shall advance the integrity and reputation of the profession consistent with the public interest. In particular, software engineers shall, as appropriate:

6.01. Help develop an organizational environment favorable to acting ethically.

6.02. Promote public knowledge of software engineering.

6.03. Extend software engineering knowledge by appropriate participation in professional organizations, meetings and publications.

6.04. Support, as members of a profession, other software engineers striving to follow this Code.

6.05. Not promote their own interest at the expense of the profession, client or employer.

6.06. Obey all laws governing their work, unless, in exceptional circumstances, such compliance is inconsistent with the public interest.

6.07. Be accurate in stating the characteristics of software on which they work, avoiding not only false claims but also claims that might reasonably be supposed to be speculative, vacuous, deceptive, misleading, or doubtful.

6.08. Take responsibility for detecting, correcting, and reporting errors in software and associated documents on which they work.

6.09. Ensure that clients, employers, and supervisors know of the software engineer’s commitment to this Code of ethics, and the subsequent ramifications of such commitment.

6.10. Avoid associations with businesses and organizations which are in conflict with this code.

6.11. Recognize that violations of this Code are inconsistent with being a professional software engineer.

6.12. Express concerns to the people involved when significant violations of this Code are detected unless this is impossible, counter-productive, or dangerous.

6.13. Report significant violations of this Code to appropriate authorities when it is clear that consultation with people involved in these significant violations is impossible, counter-productive or dangerous.

**Principle 7: COLLEAGUES**

Software engineers shall be fair to and supportive of their colleagues. In particular, software engineers shall, as appropriate:

7.01. Encourage colleagues to adhere to this Code.

7.02. Assist colleagues in professional development.

7.03. Credit fully the work of others and refrain from taking undue credit.

7.04. Review the work of others in an objective, candid, and properly-documented way.

7.05. Give a fair hearing to the opinions, concerns, or complaints of a colleague.

7.06. Assist colleagues in being fully aware of current standard work practices including policies and procedures for protecting passwords, files and other confidential information, and security measures in general.

7.07. Not unfairly intervene in the career of any colleague; however, concern for the employer, the client or public interest may compel software engineers, in good faith, to question the competence of a colleague.

7.08. In situations outside of their own areas of competence, call upon the opinions of other professionals who have competence in that area.

**Principle 8: SELF**

Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession. In particular, software engineers shall continually endeavor to:

8.01. Further their knowledge of developments in the analysis, specification, design, development, maintenance and testing of software and related documents, together with the management of the development process.

8.02. Improve their ability to create safe, reliable, and useful quality software at reasonable cost and within a reasonable time.

8.03. Improve their ability to produce accurate, informative, and well-written documentation.

8.04. Improve their understanding of the software and related documents on which they work and of the environment in which they will be used.

8.05. Improve their knowledge of relevant standards and the law governing the software and related documents on which they work.

8.06 Improve their knowledge of this Code, its interpretation, and its application to their work.

8.07 Not give unfair treatment to anyone because of any irrelevant prejudices.

8.08. Not influence others to undertake any action that involves a breach of this Code.

8.09. Recognize that personal violations of this Code are inconsistent with being a professional software engineer.

|  |  |
| --- | --- |
| **Configurable application systems** | **Application system integration** |
| Single product that provides the functionality required by a customer | Several heterogeneous system products are integrated to provide customized functionality |
| Based around a generic solution and standardized processes | Flexible solutions may be developed for customer processes |
| Development focus is on system configuration | Development focus is on system integration |
| System vendor is responsible for maintenance | System owner is responsible for maintenance |
| System vendor provides the platform for the system | System owner provides the platform for the system |