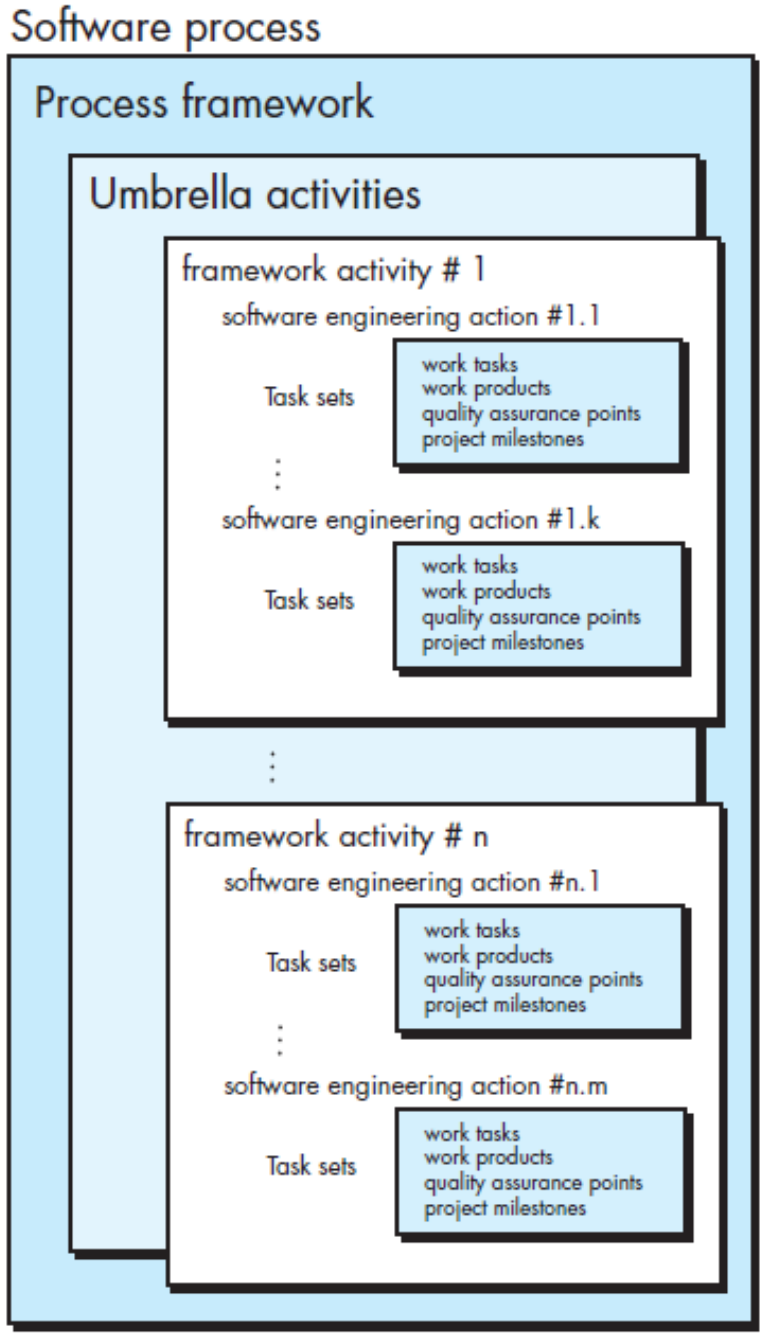
Software Process

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 or complexity. A number of *task sets*—each a collection of software engineering work 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 project team.

Software Process

FIGURE 3.1
A software process framework



Software Process

A software process is a collection of various activities.
There are five generic process framework activities:
1. Communication
The software development starts with the communication between customer and developer.
2. Planning
It consists of complete estimation, scheduling for project development and tracking.
3. Modeling

Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc. The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

Software Process

4. Construction

- Construction consists of code generation and the testing part.
- Coding part implements the design details using an appropriate programming language.
- Testing is to check whether the flow of coding is correct or not.
- Testing also check that the program provides desired output.

Software Process

5. Deployment

- Deployment step consists of delivering the product to the customer and take feedback from them.
- If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

Umbrella Activities

The phases and related steps described in our generic view of software engineering are complemented by a number of *umbrella activities*. Typical activities in this category include:

- Software project tracking and control
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Document preparation and production
- Reusability management
- Measurement
- Risk management

Umbrella Activities

1. Software project tracking and control:

In this activity, the developing team accesses project plan and compares it with the predefined schedule. If these project plans do not match with the predefined schedule, then the required actions are taken to maintain the schedule.

2. Risk management:

Risk is an event that may or may not occur. If the event occurs, then it causes some unwanted outcome. Hence, proper risk management is required.

Umbrella Activities

3. Software Quality Assurance (SQA):

SQA is the planned and systematic pattern of activities which are required to give a guarantee of software quality. **For example,** during the software development meetings are conducted at every stage of development to find out the defects and suggest improvements to produce good quality software.

4. Formal Technical Reviews (FTR):

FTR is a meeting conducted by the technical staff. The motive of the meeting is to detect quality problems and suggest improvements. The technical person focuses on the quality of the software from the customer point of view.

Umbrella Activities

5. Measurement:

Measurement consists of the effort required to measure the software. The software cannot be measured directly. It is measured by direct and indirect measures. Direct measures like cost, lines of code, size of software etc. Indirect measures such as quality of software which is measured by some other factor. Hence, it is an indirect measure of software.

6. Software Configuration Management (SCM):

It manages the effect of change throughout the software process.

Umbrella Activities

7. Reusability management:

It defines the criteria for reuse the product. The quality of software is good when the components of the software are developed for certain application and are useful for developing other applications.

8. Work product preparation and production:

It consists of the activities that are needed to create the documents, forms, lists, logs and user manuals for developing a software.

Generic Phases of Software Engineering

- The work associated with software engineering can be categorized into three generic phases, regardless of application area, project size, or complexity.

Definition phase :

This phase focuses on *what*. That is, during definition, the software engineer attempts to identify what information is to be processed, what function and performance are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system. The key requirements of the system and the software are identified.

Development phase:

This phase focuses on how. That is, during development a software engineer attempts to define how data are to be structured, how function is 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 (or nonprocedural language), and how testing will be performed.

Generic Phases of Software Engineering

Support phase:

This phase focuses on *change* associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. The support phase reapplies the steps of the definition and development phases but does so in the context of existing software. Four types of change are encountered during the support phase:

- **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.
- **Adaptation:** Over time, the original environment (e.g., CPU, operating system, business rules, external product characteristics) for which the software was developed is likely to change. Adaptive maintenance results in modification to the software to accommodate changes to its external environment.

Generic Phases of Software Engineering

- **Enhancement.** 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.
- **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 users. In essence, preventive maintenance makes changes to computer programs so that they can be more easily corrected, adapted, and enhanced.

Software Evolution

- The process of developing a software product using software engineering principles and methods is referred to as **software evolution.**
- This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.

Software Evolution

Software evolution

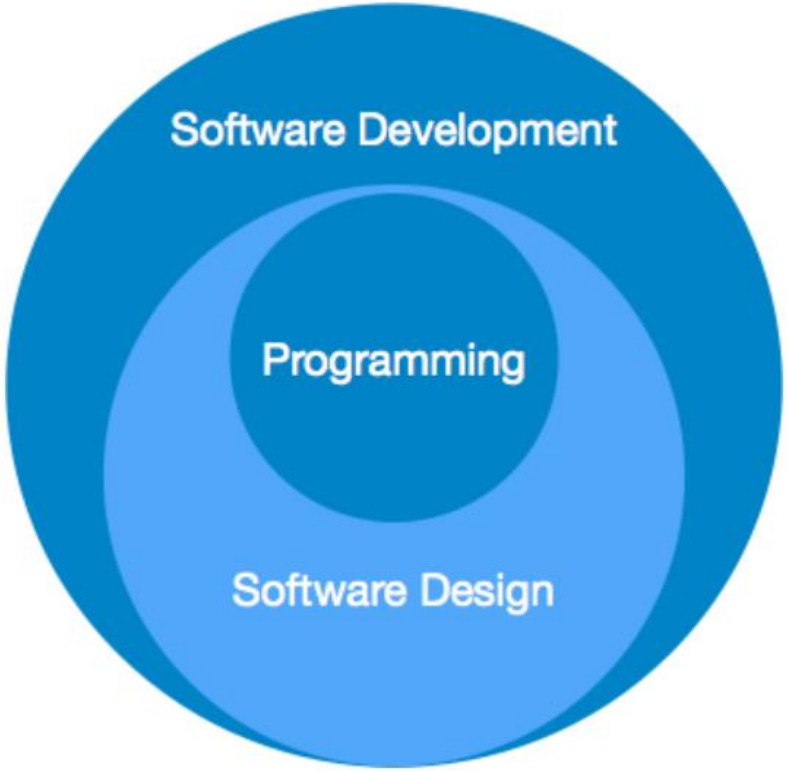
Software Evolution

- Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.
- Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

Software Paradigms

- Software paradigms refer to the methods and steps, which are taken while designing the software.
- There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand.
- These can be combined into various categories, though each of them is contained in one another:

Software Paradigms



Software paradigms

Software Development Paradigm

- Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.
- This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of:
- Requirement gathering
- Software design
- Programming

Software Design Paradigm

- This paradigm is a part of Software Development and includes:
- Design
- Maintenance
- Programming

Programming Paradigm

- This paradigm is related closely to programming aspect of software development. This includes:
- Coding
- Testing
- Integration

Need of Software Engineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software:** It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability:** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost:** As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.

Need of Software Engineering

- **Dynamic Nature:** The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management:** Better process of software development provides better and quality software product.

Software Applications

- **System software:** System software is a collection of programs written to service other programs. Some system software (e.g., compilers, editors, and file management utilities) process complex, but determinate, information structures. Other systems applications (e.g., operating system components, drivers, telecommunications processors) process largely indeterminate data.
- **Real-time software:** Software that monitors/analyzes/controls real-world events as they occur is called *real time*. Elements of real-time software include a data gathering component that collects and formats information from an external environment, an analysis component that transforms information as required by the application, a control/output component that responds to the external environment, and a monitoring component that coordinates all other components so that real-time response can be maintained.

Software Applications

- **Business software:** Business information processing is the largest single software application area. Discrete "systems" (e.g., payroll, accounts receivable/payable, inventory) have evolved into management information system (MIS) software that accesses one or more large databases containing business information.
- **Engineering and scientific software:** Engineering and scientific software have been characterized by "number crunching" algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing. Computer-aided design, system simulation, and other interactive applications have begun to take on real-time and even system software characteristics.

Software Applications

- **Embedded software.** Intelligent products have become commonplace in nearly every consumer and industrial market. Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets. Embedded software can perform very limited and esoteric functions (e.g., keypad control for a microwave oven) or provide significant function and control capability.
- **Personal computer software:** Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications, external network, and database access are only a few of hundreds of applications.

Software Applications

- **Web-based software:** The Web pages retrieved by a browser are software that incorporates executable instructions (HTML, Java), and data. In essence, the network becomes a massive computer providing an almost unlimited software resource that can be accessed by anyone with a modem.
- **Artificial intelligence software:** Artificial intelligence (AI) software makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Expert systems, also called knowledge based systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing are representative of applications within this category.

Professional Software Development

- People in business field write spreadsheet programs to simplify their jobs. Scientists and engineers write programs to process their experimental data.
- However, most software development is a professional activity in which software is developed for business purposes, for inclusion in other devices, or as software products such as information systems and computer-aided design systems.
- The key distinctions are that professional software is intended for use by someone apart from its developer and that teams rather than individuals usually develop the software. It is maintained and changed throughout its life.

Professional Software Development

- Software engineering is intended to support professional software development rather than individual programming. It includes techniques that support program specification, design, and evolution, none of which are normally relevant for personal software development.
- Many people think that software is simply another word for computer programs. However, when we are talking about software engineering, software is not just the programs themselves but also all associated documentation, libraries, support websites, and configuration data that are needed to make these programs useful.

Professional Software Development

- A professionally developed software system is often more than a single program.
- A system may consist of several separate programs and configuration files that are used to set up these programs. It may include system documentation, which describes the structure of the system, user documentation, which explains how to use the system, and websites for users to download recent product information.

Software Engineering Practice

- Consists of a collection of concepts, principles, methods, and tools that a software engineer calls upon on a daily basis
- Equips managers to manage software projects and software engineers to build computer programs
- Provides necessary technical and management how to ‘s in getting the job done
- Transforms a haphazard unfocused approach into something that is more organized, more effective, and more likely to achieve success

The Essence of Problem Solving

- **Understand the problem (communication and analysis)**
  - Who has a stake in the solution to the problem?
  - What are the unknowns (data, function, behavior)?
  - Can the problem be compartmentalized?
  - Can the problem be represented graphically?

- **Plan a solution (planning, modeling and software design)**
  - Have you seen similar problems like this before?
  - Has a similar problem been solved and is the solution reusable?
  - Can sub-problems be defined and are solutions available for the sub-problems?

The Essence of Problem Solving

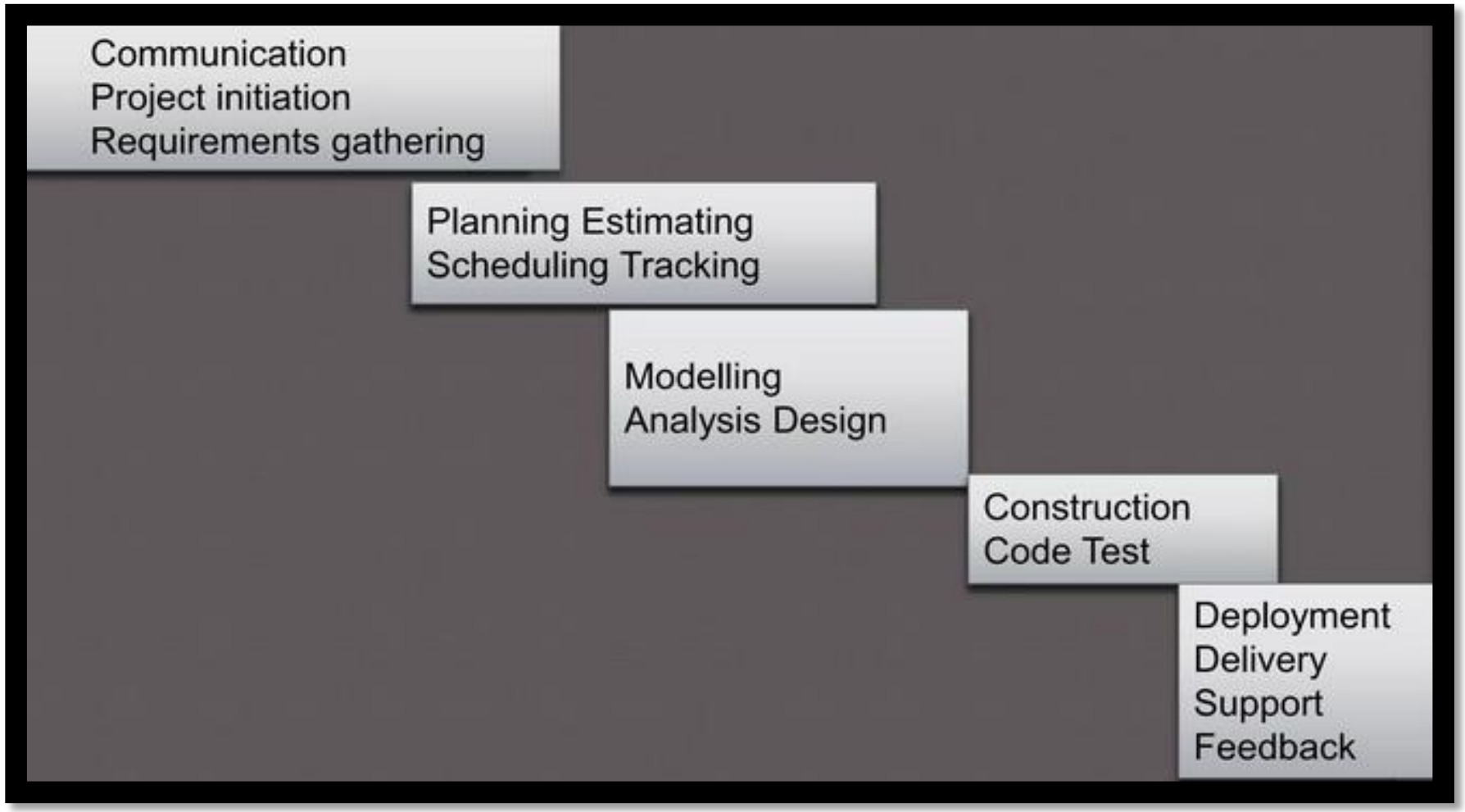
- **Carry out the plan (construction code generation)**
  - Does the solution conform to the plan? Is the source code traceable back to the design?
  - Is each component of the solution correct? Has the design and code been reviewed?

- **Examine the results for accuracy (testing and quality assurance)**
  - Is it possible to test each component of the solution?
  - Does the solution produce results that conform to the data, function, and behavior that are required?

Types of Practice

- Communication Practice
- Planning Practice
- Modeling Practice
- Construction Practice
- Testing Practice
- Deployment Practice

Types of Practice



Planning Principle

1) Understand the scope of the project
2) Involve the customer in the planning activity
3) Recognize that planning is iterative; things will change
4) Estimate based only on what you know
5) Consider risk as you define the plan
6) Be realistic on how much can be done each day by each person and how well
7) Adjust granularity as you define the plan
8) Define how you intend to ensure quality
9) Describe how you intend to accommodate change
10) Track the plan frequently and make adjustments as required

Communication Principle

1) Listen to the speaker and concentrate on what is being said
2) Prepare before you meet by researching and understanding the problem
3) Someone should facility the meeting and have an agenda
4) Face-to-face communication is best, but also have a document or presentation to focus the discussion
5) Take notes and document decisions
6) Strive for collaboration and consensus
7) Stay focused on a topic; modularize your discussion
8) If something is unclear, draw a picture
9) Move on to the next topic a) after you agree to something, b) if you cannot agree to something, or c) if a feature or function is unclear and cannot be clarified at the moment
10) Negotiation is not a contest or a game; it works best when both parties win

Modeling Practice

1) The information domain of a problem (the data that flows in and out of a system) must be represented and understood
2) The functions that the software performs must be defined
3) The behavior of the software (as a consequence of external events) must be represented
4) The models that depict information, function, and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion
5) The analysis task should move from essential information toward implementation detail

Construction Practice (Before coding)

1) Understand the problem you are trying to solve
2) Understand basic design principles and concepts
3) Pick a programming language that meets the needs of the software to be built and the environment in which it will operate
4) Select a programming environment that provides tools that will make your work easier
5) Create a set of unit tests that will be applied once the component you code is completed

Construction Practice (When Coding Begin)

1) Constrain your algorithms by following structured programming practices
2) Select data structures that will meet the needs of the design
3) Understand the software architecture and create interfaces that are consistent with it
4) Keep conditional logic as simple as possible
5) Create nested loops in a way that makes them easily testable
6) Select meaningful variable names and follow other local coding standards
7) Write code that is self-documenting
8) Create a visual layout (e.g., indentation and blank lines) that aids code understanding

Constructing Practice (When coding ends)

1) Conduct a code walkthrough
2) Perform unit tests (black-box and white-box) and correct errors you have uncovered
3) Refactor the code

Testing Practice

1) All tests should be traceable to the software requirements
2) Tests should be planned long before testing begins
3) The Pareto principle applies to software testing
   - 80% of the uncovered errors are in 20% of the code
4) Testing should begin “in the small” and progress toward testing “in the large”
   - Unit testing --> integration testing --> validation testing --> system testing
5) Exhaustive testing is not possible

Deployment Practice

1) Customer expectations for the software must be managed
   - Be careful not to promise too much or to mislead the user
2) A complete delivery package should be assembled and tested
3) A support regime must be established before the software is delivered
4) Appropriate instructional materials must be provided to end users
5) Buggy software should be fixed first, delivered later