SOFTWARE ENGINEERING (CA725)

Dr. Ghanshyam S. Bopche
Assistant Professor
Dept. of Computer Applications

August 25, 2021

SOFTWARE ENGINEERING (CA725)

Syllabus

- **Unit-I**: Introductory concepts, The evolving role of software, Its characteristics, components and applications, A layered technology, The software process, Software process models, Software Development Life cycle, Software process and project metrics, Measures, Metrics and Indicators, Ethics for software engineers.
- **Unit-II**: Software Project Planning, Project planning objectives, Project estimation, Decomposition techniques, Empirical estimation models, System Engineering, Risk management, Software contract management, Procurement Management.
- **Unit-III**: Analysis and Design, Design concept and Principles, Methods for traditional, Real time of object oriented systems, Comparisons, Metrics, Quality assurance

SOFTWARE ENGINEERING (CA725) (cont.)

- **Unit-IV**: Testing fundamentals, Test case design, White box testing, Basis path testing, Control structure testing, Black box testing, Strategies: Unit testing, integration testing, Validation Testing, System testing, Art of debugging, Metrics, Testing tools
- **Unit-V**: Formal Methods, Clean-room Software Engineering, Software reuse, Re-engineering, Reverse Engineering, Standards for industry.
- **References**:

1. Rajib Mall, "Fundamentals of Software Engineering", 4th Edition, PHI, 2014.
2. Roger S. Pressman, "Software Engineering-A practitioner's approach", 7 th Edition, McGraw Hill, 2010.
3. Ian Sommerville, "Software engineering", 10th Edition, Pearson education Asia, 2016.

SOFTWARE ENGINEERING (CA725) (cont.)

4. Pankaj Jalote, "An Integrated Approach to Software Engineering", Springer Verlag, 1997.
5. James F Peters, Witold Pedrycz, "Software Engineering – An Engineering Approach", John Wiley and Sons, 2000.
6. Ali Behforooz, Frederick J Hudson, "Software Engineering Fundamentals", Oxford University Press, 2009.
7. Bob Emery , "Fundamentals of Contract and Commercial Management",Van Haren Publishing, Zaltbommel, 2013

What is a Software?

- Computer programs (which are stored in and executed by computer hardware) and associated data (which also is stored in the hardware) that may be dynamically written or modified during execution. - National Institute of Standards and Technology (NIST), USA

What is a Software?

- It is a product (delivers computing potential), and at the same time, the vehicle for delivering the product (basis for the control of computer, the communication of information, and the creation and control of other programs).
- An **information transformer** - producing, managing, acquiring, modifying, displaying, or transmitting information.

What is a Software?

- Indispensable technology for business, science, and engineering.
- Enable the creation of new technologies for e.g., genetic engineering, and nanotechnology.
- Enable extension of existing technologies for e.g., telecommunications.
- Brought radical changes in older technologies for e.g., the printing industry.
- Driving force for personal computer revolution.

What does Software Engineer do?

- Solve problems economically by developing high-quality software.
- Software engineers design software systems.

Characteristics of a Software

- Software is intangible.
  - One cannot feel the shape of a piece of software.
  - Software design can be hard to visualize.
  - Does the feel of shape and design of a particular artifact matters?
- The mass-production of duplicate pieces of software is trivial.
  - Software is developed or engineered; it is not manufactured.
  - Cost of software is in its development, not in its manufacturing.
- The software industry is labor intensive.
  - Require truly "intelligent" machines to fully automate software design or programming.
  - Very little success is achieved till date.

Characteristics of a Software (cont.)

- Detection of bugs (errors or defects) in software is not so easy.
- Software is physically easy to modify; however, because of its complexity it is very difficult to make changes that are correct.
  – New bugs may get introduced as a side effect of modification.
- Software does not wear out with use, but instead its design deteriorates as it is changed repeatedly.

Characteristics of a Software (cont.)

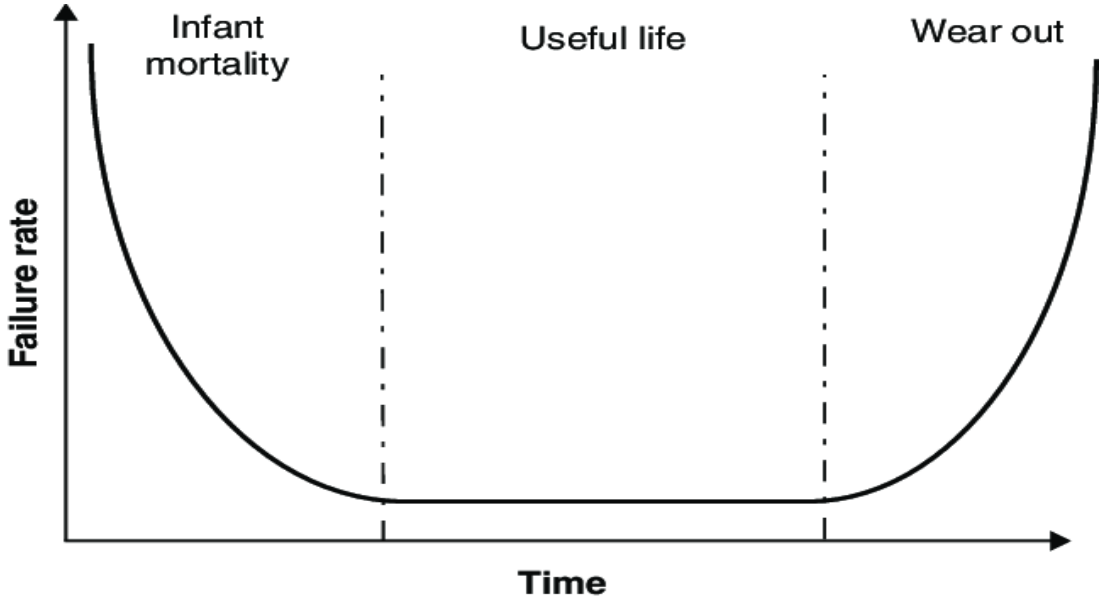


Figure: Failure Curve for Hardware

Characteristics of a Software (cont.)

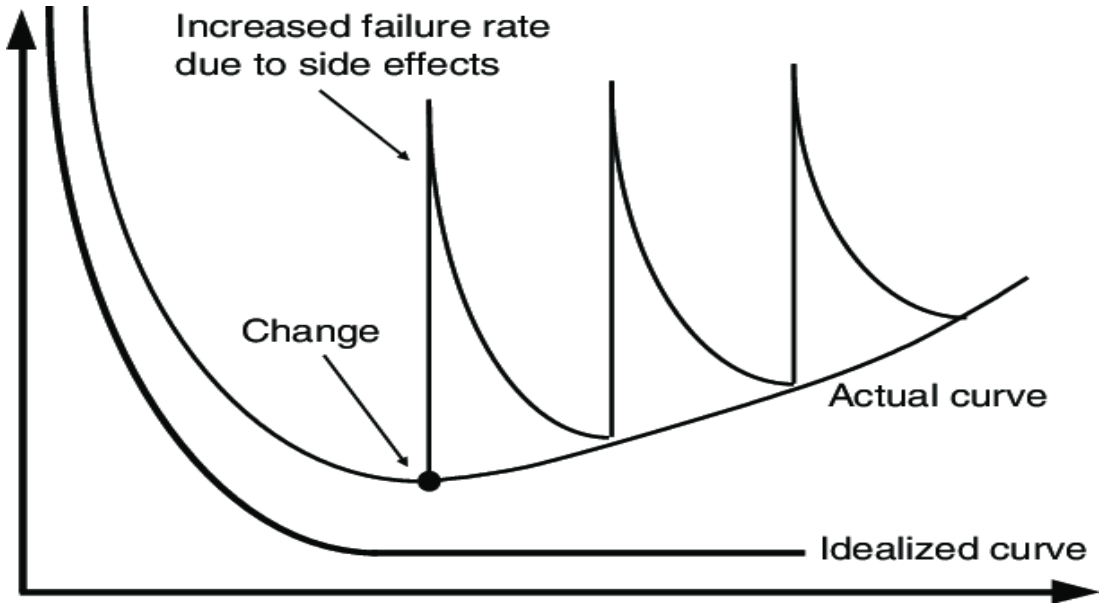


Figure: Failure Curve for Software

Software Crisis

- Existing softwares are of poor quality and required repeated changes further deteriorates the quality.
  - Strong demand for new and changed software.
- Customers expect software to be of high quality and to be produced rapidly.
- Project manager expect to launch a software product within time-to-market window.
  - Software developer maybe not able to meet the expectations of their managers and customers.
  - Leads to software products either never delivered or delivered late and over budget.
  - Already delivered software systems require major modification before they can be actually used.
- **Solution**: appropriate use of software engineering methods.

Software Application Domains

1. **System Software**
   - A collection of programs written to service other programs, for example, Operating System, drivers, compilers, networking softwares, file management utilities, etc.
   - **Characteristics**:
     - Heavy interaction with hardware.
     - Heavy usage by multiple users.
     - Concurrent operations (require scheduling, resource sharing, and sophisticated process management).
     - Complex data structures.
     - Multiple external interfaces.

Software Application Domains (cont.)

2. **Application Software**
   - Stand alone programs that solve a specific business need.
   - Process business or technical data in a way that facilitate business operations or management/technical decision making.
   - Used to control business functions in real-time (e.g., point-of-sale transaction processing, real-time manufacturing process control).
3. **Engineering/Scientific Software**
   - Characterized by "number crunching" algorithms.
   - **Applications**: austronomy, volcanology, space shuttle orbital dynamics, molecular biology, automated manufacturing, etc.

Software Application Domains (cont.)

4. **Embedded Software**
   - Resides within a product or system.
   - Can perform limited and esoteric functions.
   - Used to implement and control features and functions for the end user and for the system itself.
5. **Product-line Software**
   - Designed to provide a specific capability for use by many different customers.
   - Can focus on a limited and esoteric marketplace (e.g., inventory control products) or address mass consumer products.

Software Application Domains (cont.)

6. **Web Applications (WebApps)**
   - Network centric software category.
   - With the emergence of Web 2.0, WebApps evolved as sophisticated computing environments.
7. **Artificial Intelligence Software**
   - Uses nonnumerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis.
   - **Applications**: Robotics, expert systems, pattern recognition (image and voice), etc.

Types of software

1. **Custom software**
   - Developed to meet the specific needs of a particular customer (of little use to others).
   - Typically used by only a few people and its success depends on meeting their needs.
   - **Examples** - web sites, air-traffic control systems and software for managing the specialized finances of large organizations, etc.
   - Comes under conventional projects (1960 - 1970, 100% custom built).

Types of software (cont.)

2. **Generic software**
   - Designed to be sold on the open market, to perform functions that many people need, and to run on general purpose computers.
   - Requirements are determined largely by market research.
   - Often called Commercial Off-The-Shelf software (COTS).
   - Examples - word processors, spreadsheet processors, compilers, web browsers, operating systems, computer games and accounting packages for small businesses, etc.

Types of software (cont.)

3. **Embedded software**
   - Runs on specific hardware devices which are typically sold on the open market.
   - Examples - washing machines, DVD players, microwave ovens, automobiles, etc.
   - Users cannot usually replace embedded software or upgrade it without also replacing the hardware.

Other categories of software

1. **Real-time Software**
   - Has to react immediately (i.e. in real time) to stimuli from the environment.
   - Design criteria: responsiveness must be always guaranteed.
   - All embedded systems operate in real time (with the use of special-purpose hardware).
   - Generic applications, such as spreadsheets and computer games - have soft real-time characteristics.
   - Key concern: safety
   - Types of Real-time Software
     1.1 **Hard Real-time System**: within a fraction of delay system will collapse for e.g., Air Traffic Control.
     1.2 **Soft Real-time System**: within a fraction of delay system will not collapse immediately, instead there will be some deterioration in functionality.

Other categories of software (cont.)

2. **Data Processing Software**
   - Used to run businesses and performs functions such as recording sales, managing accounts, printing bills etc.
   - Design issue: organization of data in order to provide useful information to the users so they can perform their work effectively.
   - Key concern: accuracy and security of the data.

What is software engineering?

Software engineering is the process of solving customers' problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints.

Stakeholders in software engineering

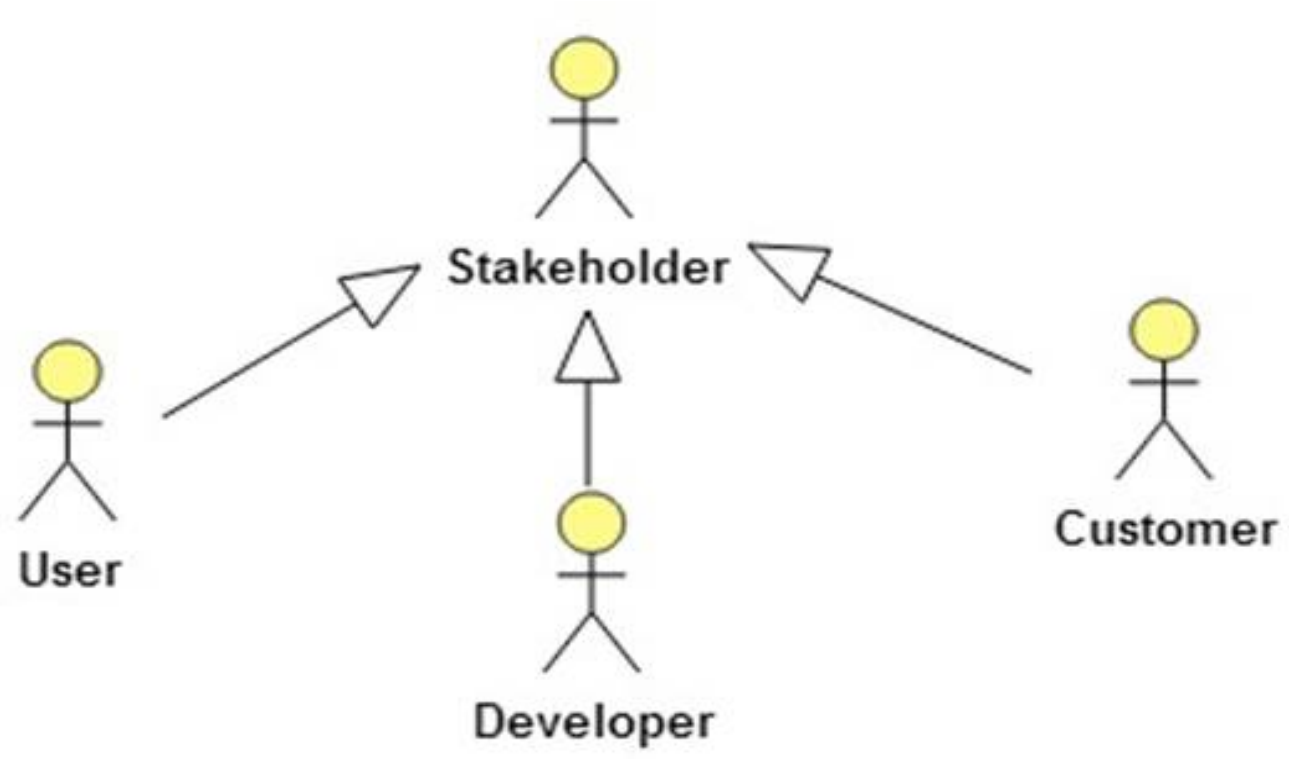


Figure: Stakeholders in Software Engineering

Software quality

For a stakeholder, a software is said to be of good quality if the outcome of its development and maintenance helps them meet their personal objectives.

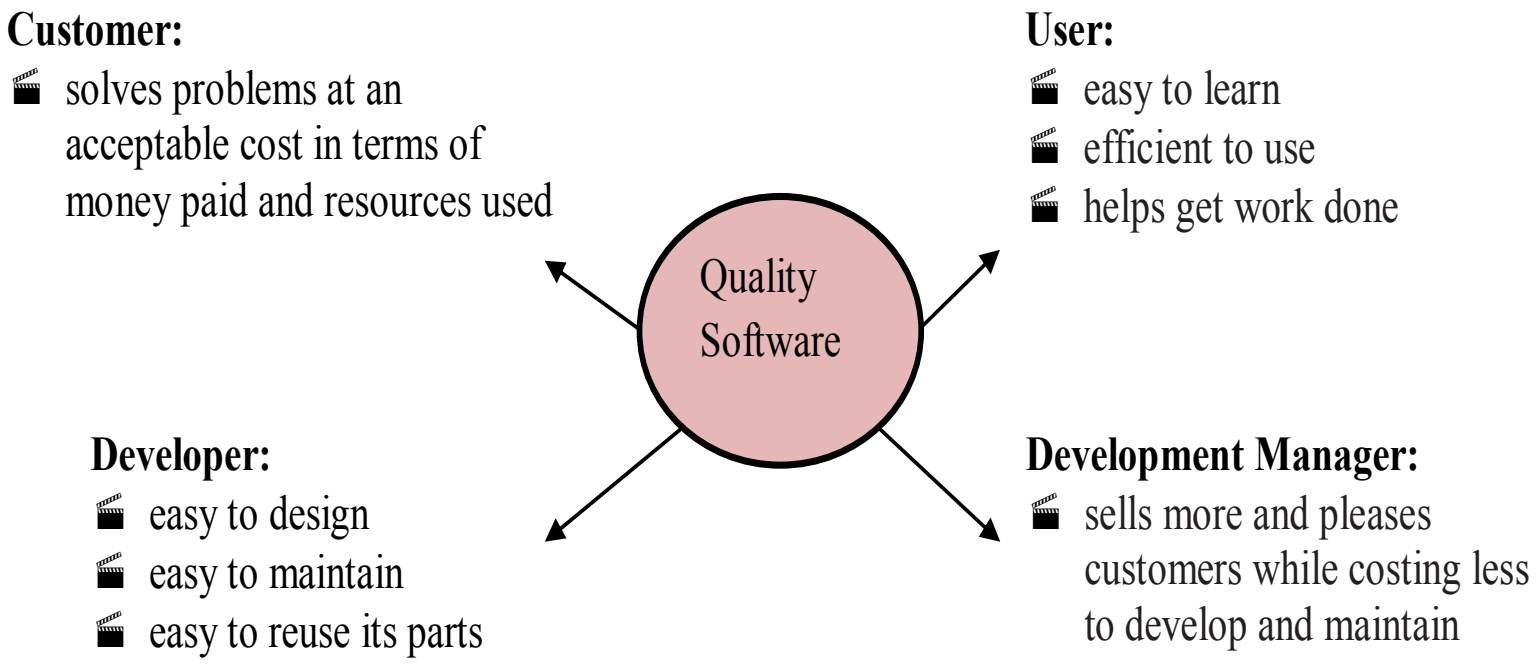


Figure: What software quality means to different stakeholders

Attributes of software quality

1. **Usability**
   - User friendliness (learn, understand, and use)
   - The higher the usability of software, the easier it is for users to work with it.
   - **Aspects of usability**: learnability for novices, efficiency of use for experts, and handling of errors.
2. **Efficiency**
   - The more efficient software is, the less it uses of CPU-time, memory, disk space, network bandwidth and other resources.
   - Customer is always concerned about the software efficiency.

Attributes of software quality (cont.)

3. **Reliability**
   - Software is more reliable if it has fewer failures.
   - Reliability depends on the number and type of mistakes a software engineer make.
   - How to improve software reliability?
     - Ensure that the software is easy to implement and change.
     - Test it thoroughly.
     - Ensure that if failures occur, the system can handle them or can recover easily.

Question: An organization has developed a scientific application for a client. This application has 45 minute failure in a year. Calculate the reliability of this product?

Attributes of software quality (cont.)

4. **Maintainability**
   - Refers to the ease with which software engineer can change the software.
   - Software that is more maintainable can result in reduced costs for both developers and customers.
5. **Reusability**
   - A software component is reusable if it can be used in several different systems with little or no modification.
   - High reusability can reduce the long-term costs faced by the development team.

- **Note:** These external software quality attributes can be observed by the stakeholders and have a direct impact on them.

Software quality trade-offs

- Improving efficiency
  ← May make a design less easy to understand. This can reduce maintainability, which leads to defects that reduce reliability.
- Achieving high reliability: often involves repeatedly checking for errors and adding redundant computations.
  ← However, achieving high efficiency, in contrast, may require removing such checks and redundancy.
- Improving usability may require adding extra code to provide feedback to the users.
  ← However, it might led to reduce overall efficiency and maintainability.

Internal software quality criteria

- **The amount of commenting of the code**
  - Can be measured as the fraction of total lines in the source code that are comments.
  - Impacts maintainability, and indirectly it impacts reliability.
- **The complexity of the code**
  - Can be measured in terms of the nesting depth, the number of branches and the use of certain complex programming constructs.
  - Directly impacts maintainability and reliability.

Categories of software engineering projects

1. **Evolutionary projects**
   - Projects that involve modifying an existing system.
   - Evolutionary or maintenance projects can be of several different types:
     - **Corrective projects**: involve fixing defects.
     - **Adaptive projects**: involve changing the system in response to changes in the environment in which the software runs.
     - **Enhancement projects**: involve adding new features for the users/customers.
     - **Re-engineering or perfective projects**: involve changing the system internally so that it is more maintainable, without making significant changes that the user will notice.

Categories of software engineering projects (cont.)

2. **Greenfield projects**
   - Projects that involve starting to develop a system from scratch.
   - Comes under conventional projects (1960 - 1970, 100% custom built).
3. **Modern Projects** (2000 and onward, 70% reusable components, 30% custom built).
   - Projects that involve building most of a new system from existing components, while developing new software only for missing details.

Ethics in Software Engineering

IEEE/ACM code of ethics: Software engineers shall

1. act consistently with the public interest.
2. act in the best interests of their client or employer, as long as this is consistent with the public interest.
3. develop and maintain their product to the highest standards possible.
4. maintain integrity and independence when making professional judgments.
5. promote an ethical approach in management.
6. advance the integrity and reputation of the profession, as long as doing so is consistent with the public interest.
7. be fair and supportive to colleagues.
8. participate in lifelong learning.
9. ...

Activities common to software projects

1. **Requirements specification and analysis**
   Goal: understand the customer's problems, the customer's business environment, and the available technology which can be used to solve the problems.
   The overall process may include the following activities:
   - 1.1 **Domain Analysis**: understanding the background needed in order to understand the problem and make intelligent decisions.
   - 1.2 **Defining the problem**: narrowing down the scope of the system by determining the precise problem that needs to solved.
   - 1.3 **Requirements gathering**: obtaining all the ideas people have about what the software should do.
   - 1.4 **Requirements analysis**: organizing the information that has been gathered, and making decisions about what in fact the software should do.

Activities common to software projects (cont.)

1.5 **Requirements specification**: writing a precise set of instructions that define what the software should do (only software behavior not implementation).

Figure: Specification types

Activities common to software projects (cont.)

2. **Design**: process of deciding how the requirements should be implemented using the available technology.
   Important activities during software design include:
   - 2.1 **System Engineering**: deciding what requirements should be implemented in hardware and what in software.
   - 2.2 **Software architecture**: deciding how the software is to be divided into subsystems and how the subsystems are to interact (with the help of architectural patterns or styles).
   - 2.3 **Detailed design**: deciding how to construct the details of each subsystem. Such details include the data structures, classes, algorithms and procedures.
   - 2.4 **User interface design**: deciding in detail how the user is to interact with the system, and the look and feel of the system.
   - 2.5 **Database selection**: deciding how the data will be stored on disk in databases or files.

Activities common to software projects (cont.)

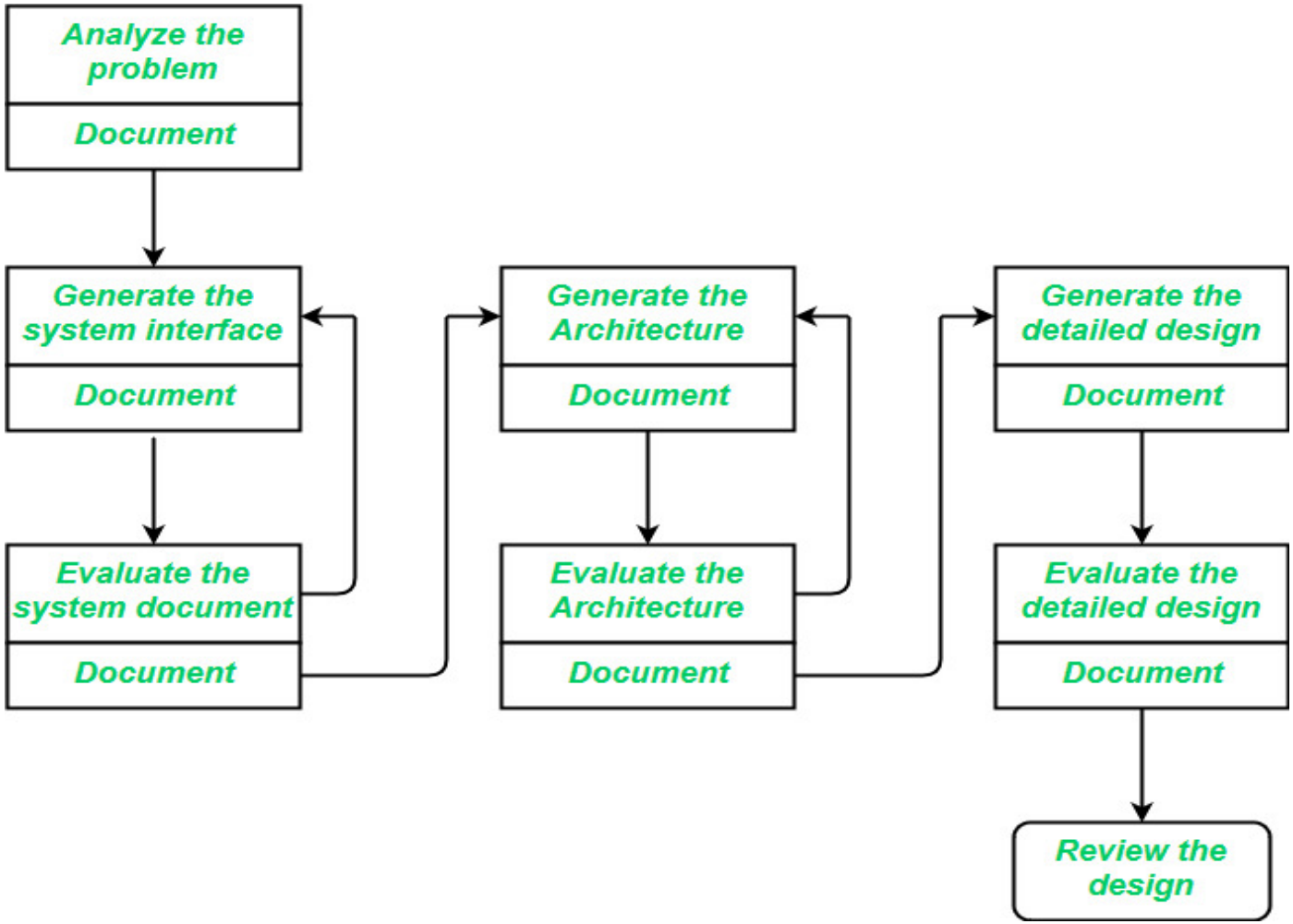


Figure: Software design process

Activities common to software projects (cont.)

3. **Modeling**: process of creating a representation of the domain or the software.
   Note: Modeling can be performed visually, using diagrams, or else using semi-formal or formal languages that express the information systematically or mathematically.
   Example: Unified Modeling Language (UML) - a visual language that uses semi-formal notations and diagrams.
   Various modeling approaches used during both requirements analysis and design are as follows:
   - 3.1 **Use case modeling**: representing the sequences of actions performed by the users of the software.
   - 3.2 **Structural modeling**: representing the classes and objects present in the domain or in the software.
   - 3.3 **Dynamic and behavioral modeling**: representing the states that the system can be in, the activities it can perform, and how its components interact.

Activities common to software projects (cont.)

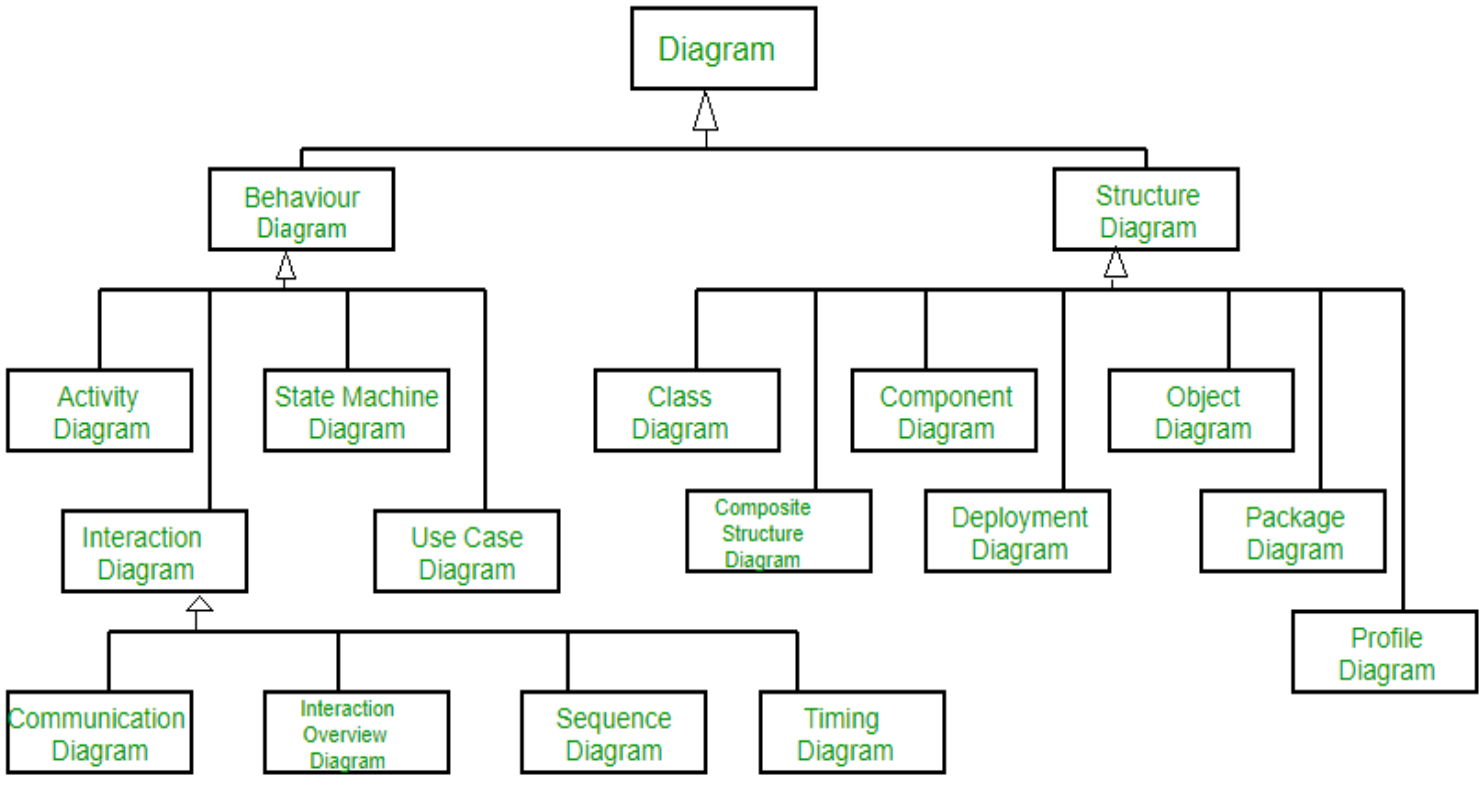
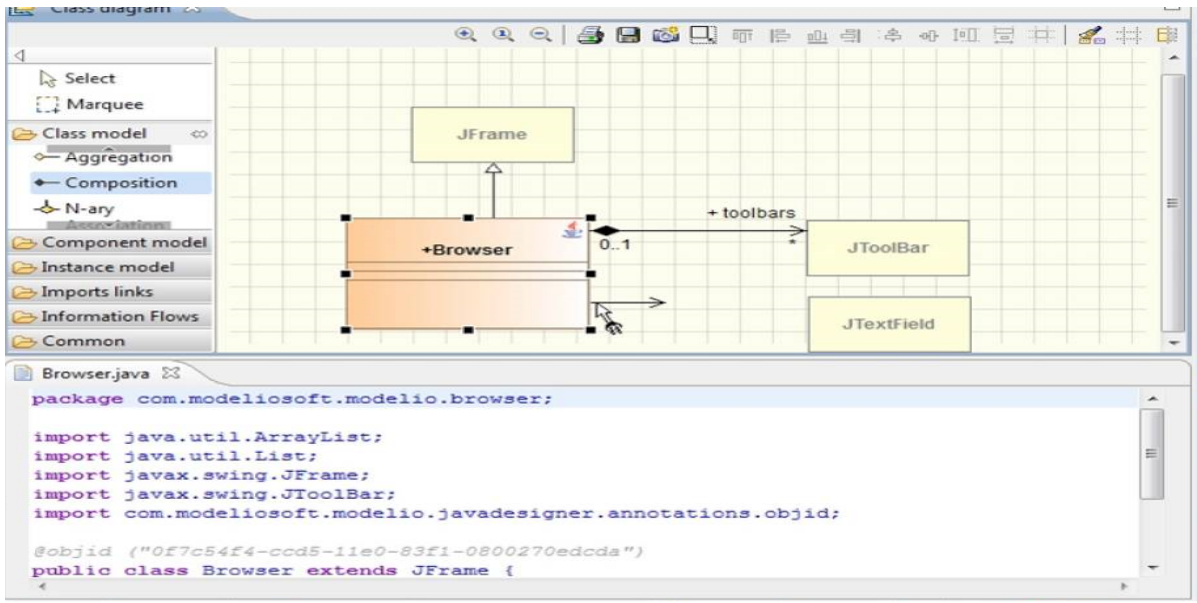


Figure: Software modeling - use of UML diagrams

Activities common to software projects (cont.)

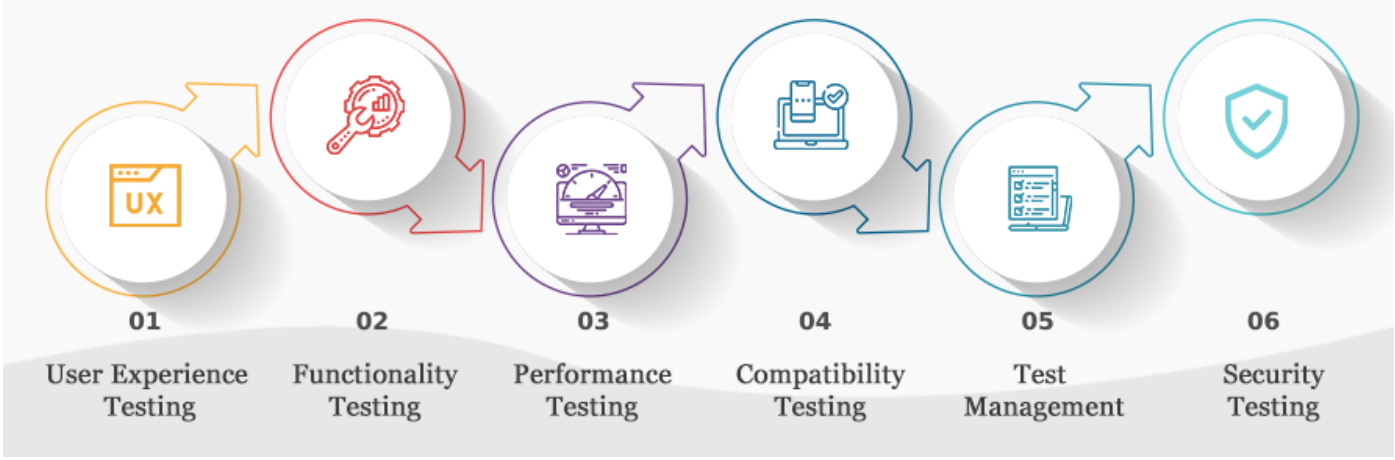
4. Development/Programming



- Involves the translation of higher-level designs into code using particular programming languages.
- Final stage of design and it involves making decisions about the appropriate use of programming language constructs.

Activities common to software projects (cont.)

5. **Quality assurance (QA)**: include all the processes needed to ensure that the quality objectives of the software being developed are met.



→ Quality assurance occurs throughout a project, and includes many activities, including the following:

   5.1 **Reviews and inspections**: formal meetings organized to discuss requirements, designs or code to see if they are satisfactory.

   5.2 **Testing**: process of systematically executing the software to see if it behaves as expected.

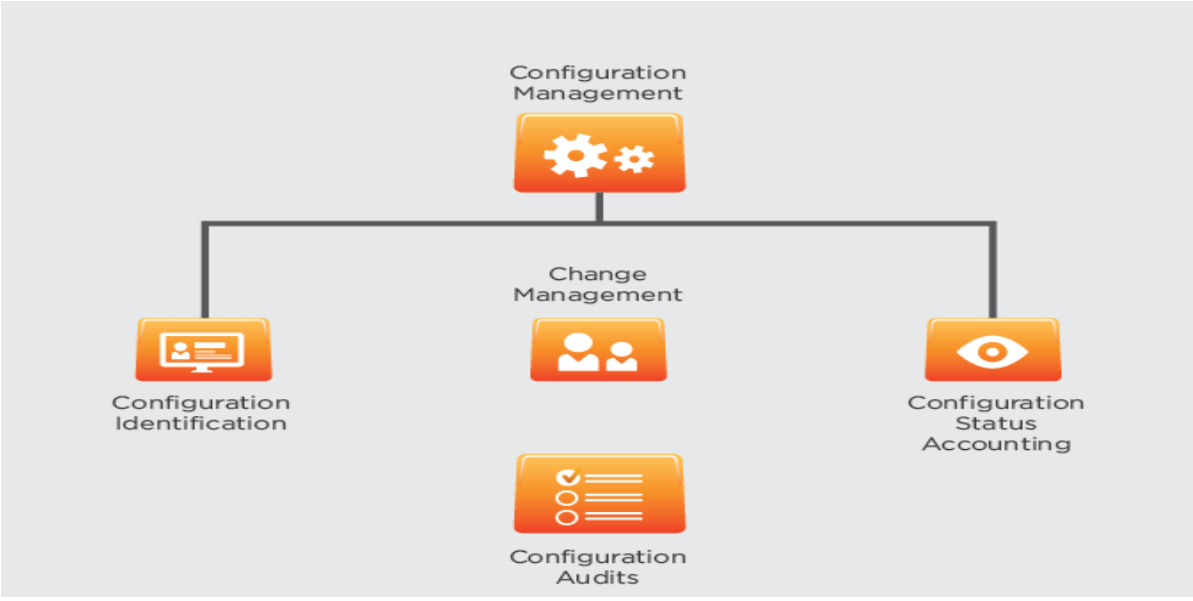
Activities common to software projects (cont.)

6. **Deployment**

- Involves distributing and installing the software and any other components of the system such as databases, special hardware etc.
- It also involves managing the transition from any previous system.

Activities common to software projects (cont.)

7. Managing software configurations



- Involves identifying all the components that compose a software system, including files containing requirements, designs and source code.
- It also involves keeping track of configurations as they change, and ensuring that changes are performed in an organized way.

Activities common to software projects (cont.)

8. **Managing the process**: integral part of software engineering.
   Goal: ensure that the project is at all times under control (all risks to the project are identified and managed).
   $\rightarrow$ In particular, project manager has to undertake the following tasks:
   - 8.1 **Estimating the cost of the system**: involves studying the requirements and determining how much effort they will take to design and implement.
   - 8.2 **Planning**: process of allocating work to a team of developers, and setting a schedule with deadlines.

   Note: Both estimated software development cost and development plans need to be examined and revised on a regular basis, since initial estimates will only be rough.

Software Engineering Layers

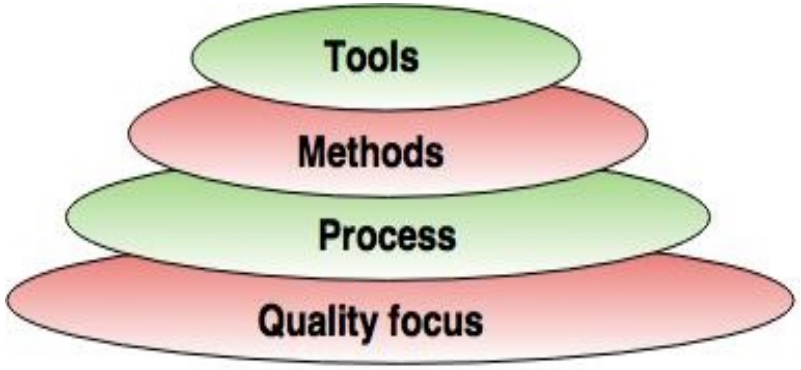


Figure: Software Engineering a layered technology

- **Quality Focus**: the bedrock that support software engineering.
  - Organizational commitment to quality.
- **Process**: the foundation of software engineering.
  - Process defines a framework that must be established for effective delivery of software engineering technology.

Software Engineering Layers (cont.)

- **Methods**: provide the technical how-to's for building software.
  - Include broad array of tasks such as communication, requirement analysis, design modeling, program construction, testing, and support.
- **Tools**: provide automated and semiautomated support for the process and the methods.

Software Process

Software Process: a framework for the activities, actions, and tasks that are required to build high-quality software.

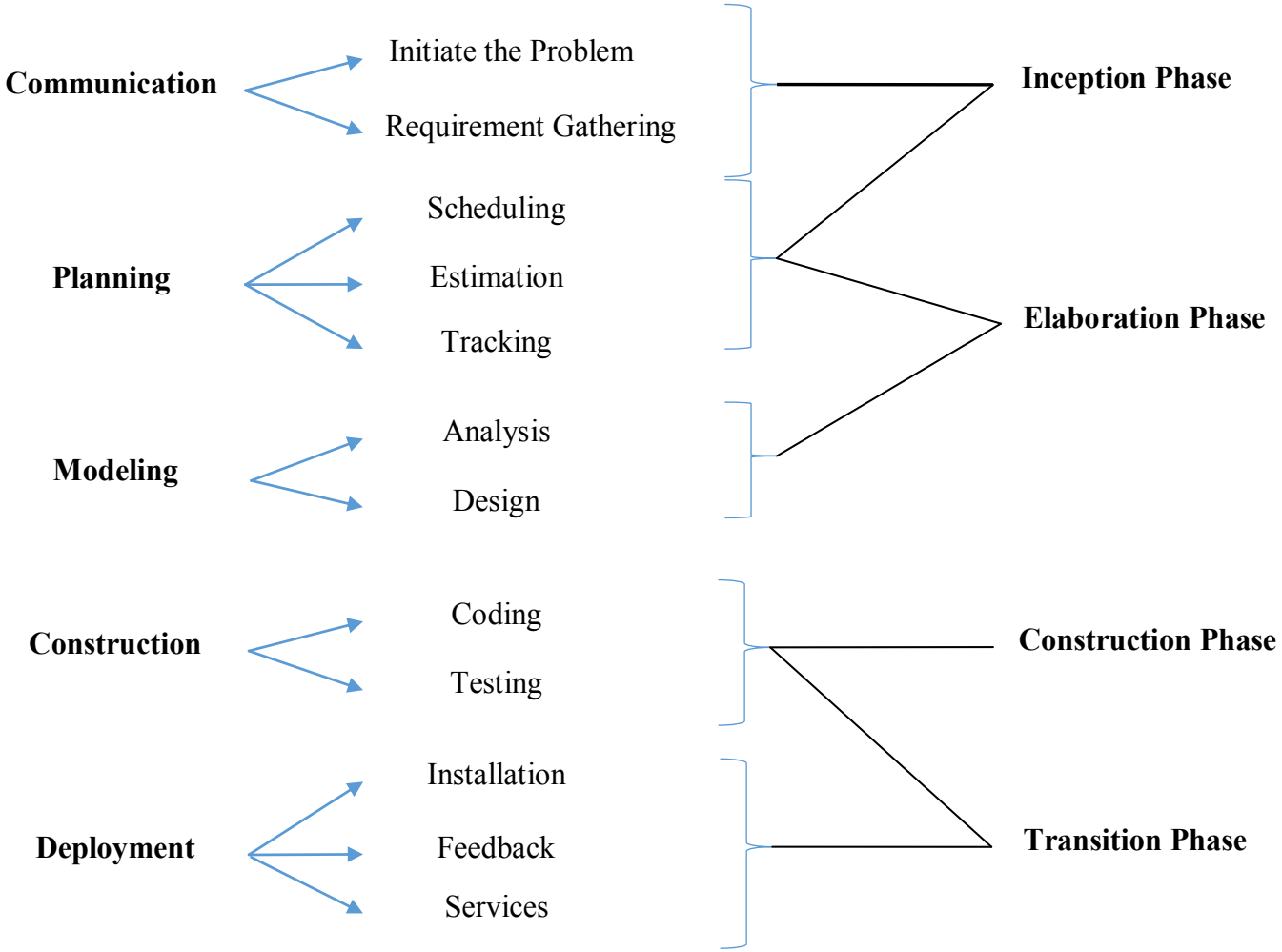


Figure: Activities of a generic Software Process framework

Software Process

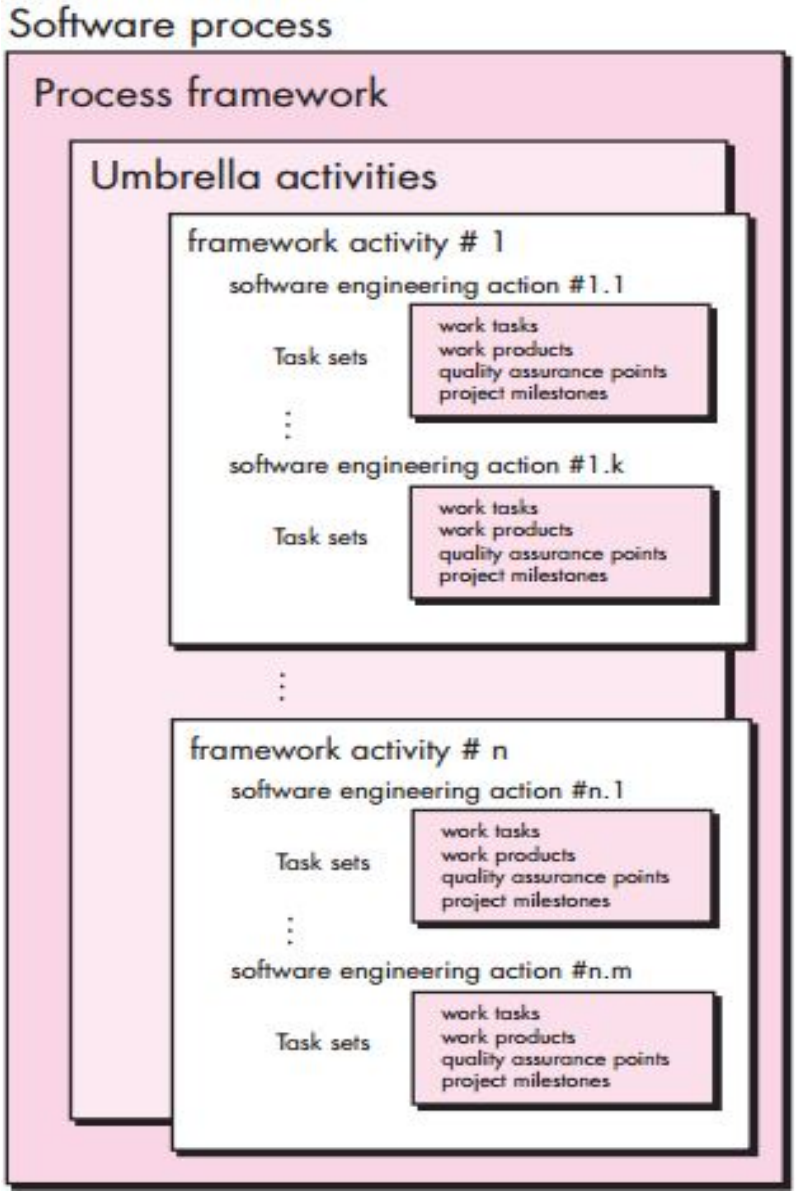


Figure: A software process framework

Process Flow

Process Flow: describes "how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to the sequence and time".

Types of Process Flow

- **Linear process flow**: executes each of the five framework activities in sequence, beginning with communication and culminating with deployment.

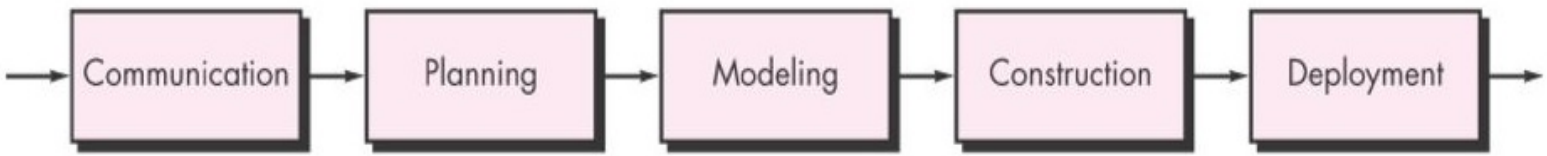


Figure: Linear process flow

- **Iterative process flow**: repeats one or more of the activities before proceeding to the next.

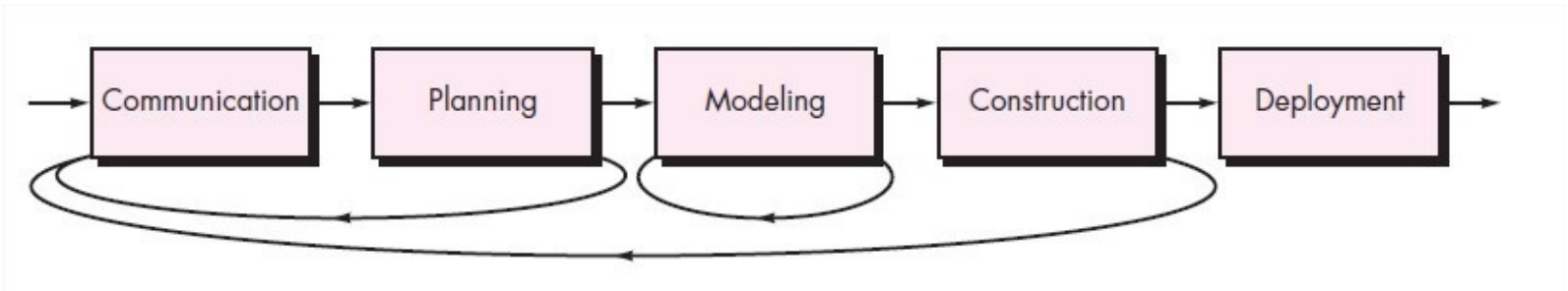


Figure: Iterative process flow

Types of Process Flow (cont.)

- **Evolutionary process flow**: executes the activities in a "circular" manner.
  Each circuit through the five activities leads to the more complete version of the software.

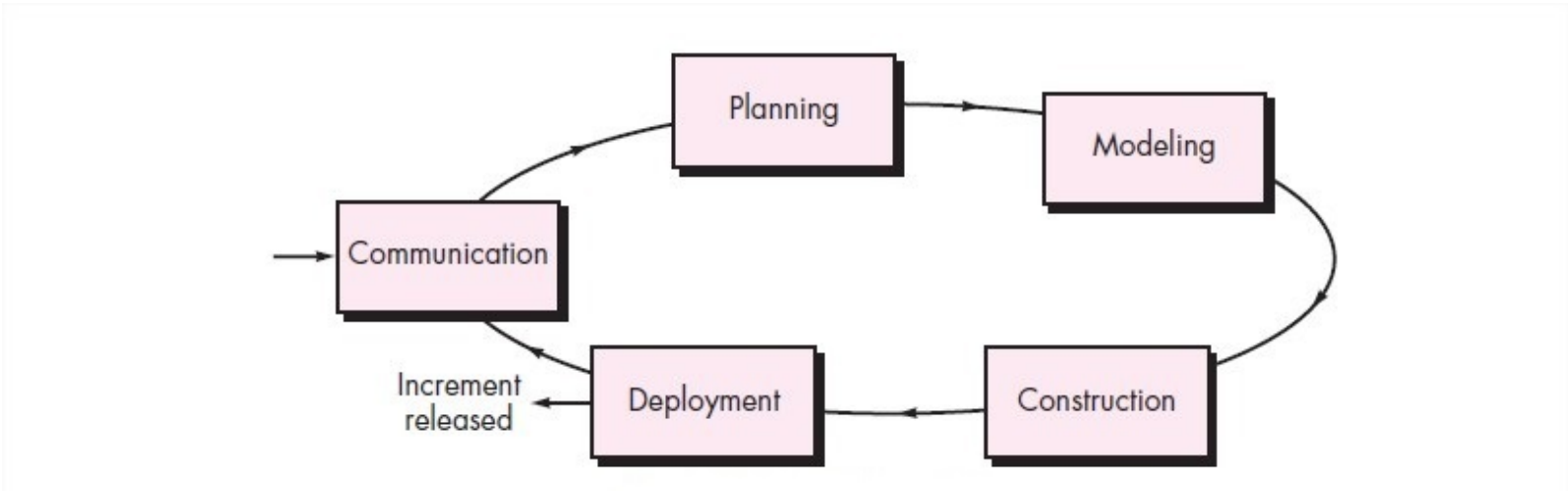


Figure: Evolutionary process flow

Types of Process Flow (cont.)

- **Parallel process flow**: executes one or more activities in parallel with other activities (e.g., modeling for one aspect of the software might be executed in parallel with construction of another aspect of the software).

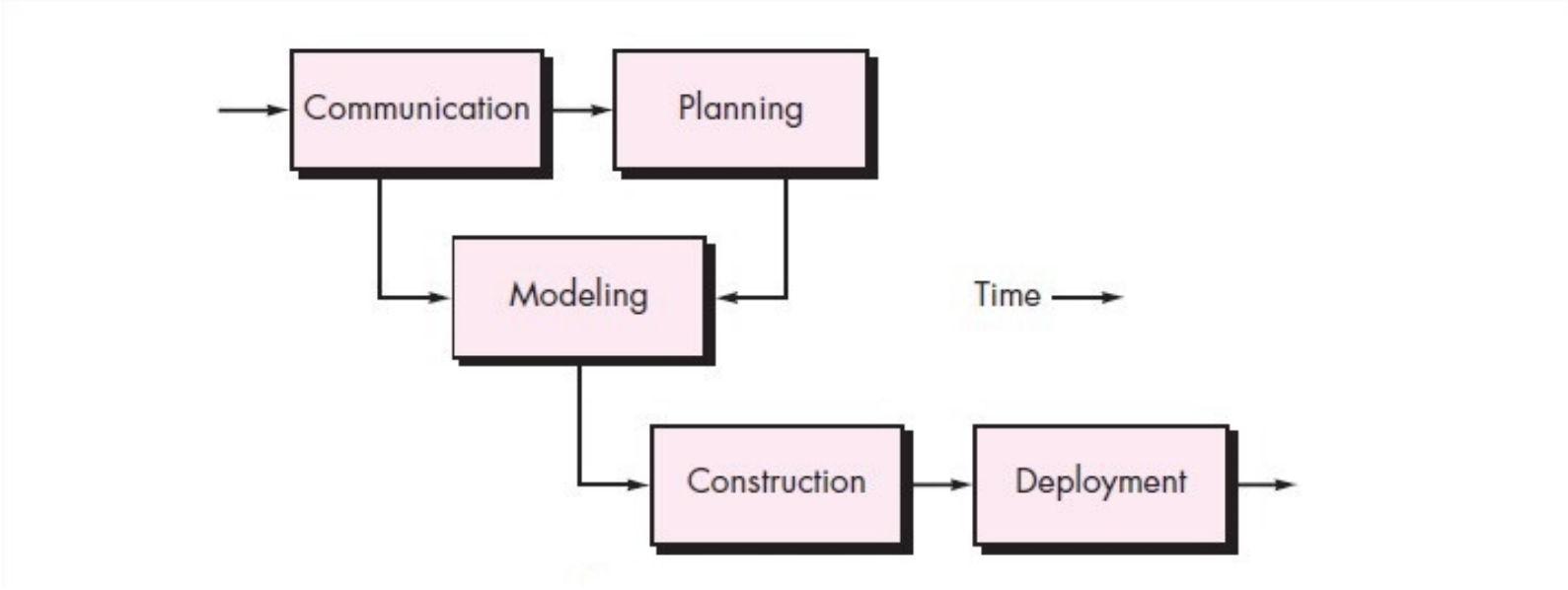


Figure: Parallel process flow

Process Model

- Roadmap/guidelines for
  - Software development
  - Software maintenance
- Without using process model one can not
  - achieve quality
  - achieve continuous improvement
  - estimate resources
  - control process engineering activities

The Waterfall Model

- Systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software.
- Also called as classic life cycle model or Royce model (proposed by Winston Royce in 1970) or linear sequential process model.
- Ideal in situation where requirements are fixed and work is to proceed to completion in a linear fashion.
- Document driven approach (i.e., customer has to state all the requirements explicitly).

The Waterfall Model (cont.)

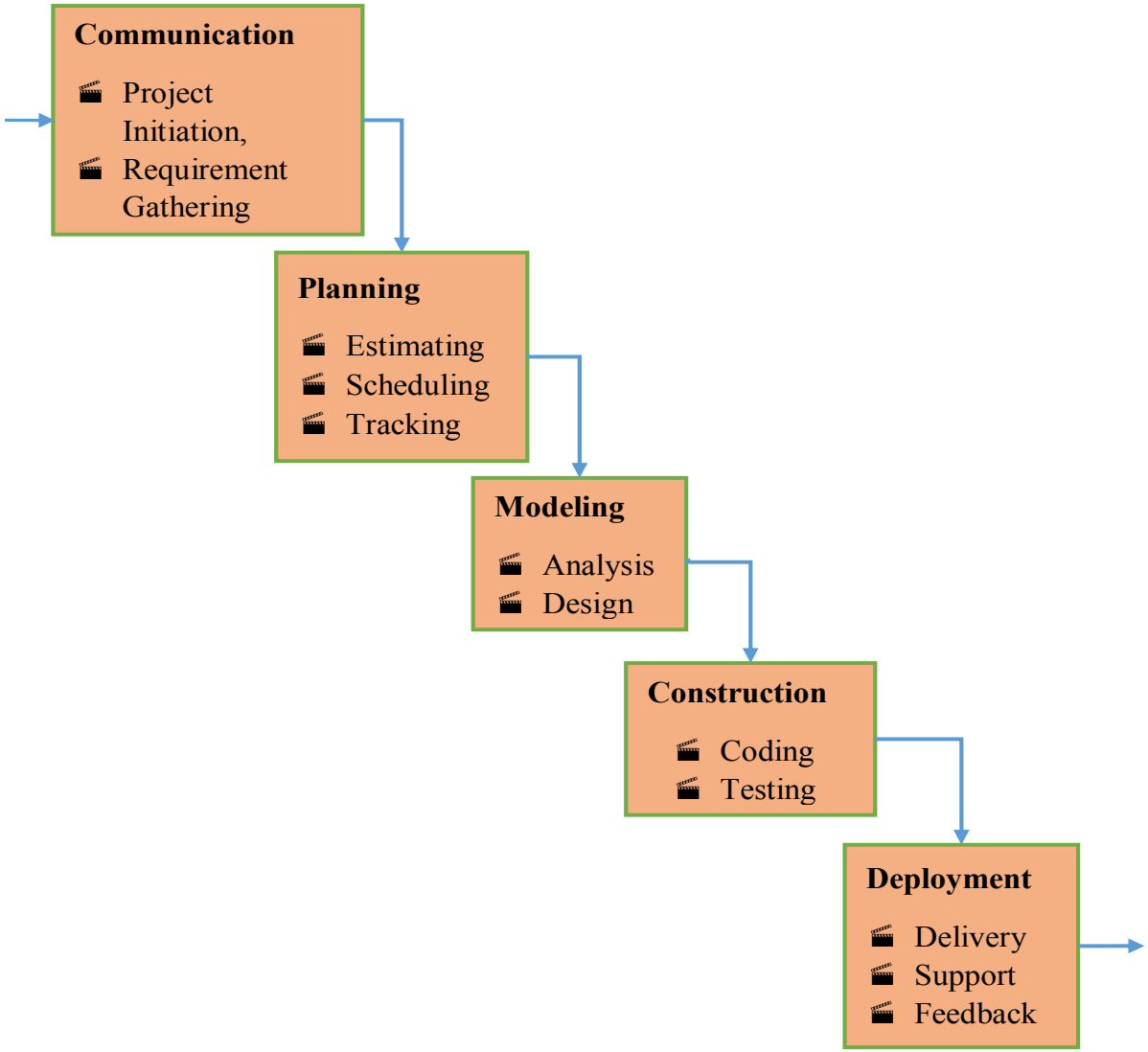


Figure: The waterfall model

The Waterfall Model (cont.)

- A working version of the program(s) will not be available until late in the project time span, i.e., only core/final product will be delivered to customer at the end.
- Suffers from blocking state problem.

Incremental Process Model

- **Motivation**: compelling need to provide a limited set software functionality (core product with minimal features) to users quickly.
- Incremental process model produce the software in increments (pilot approach).
- Incremental process model combines elements of linear and parallel process flows.

Incremental Process Model (cont.)

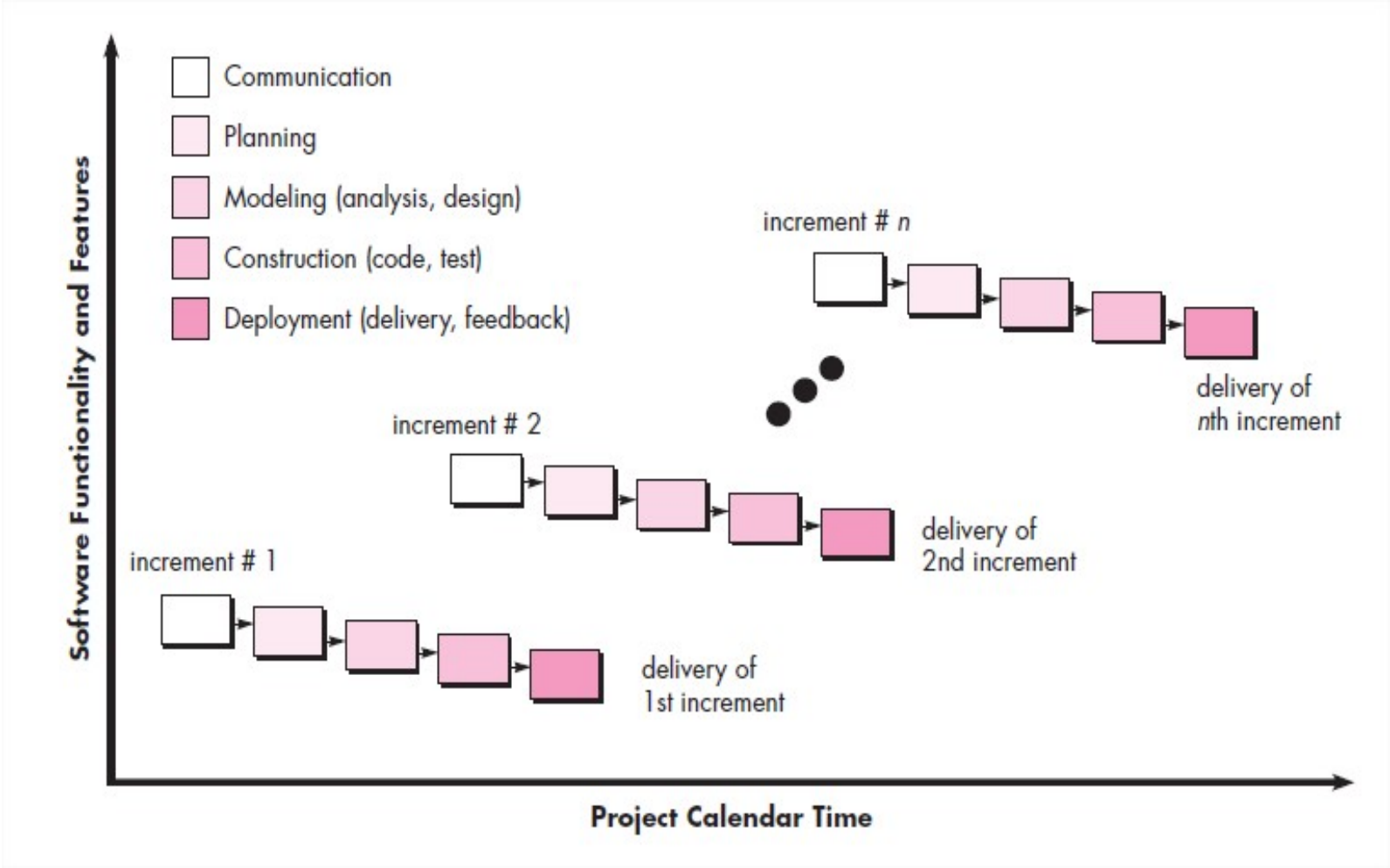


Figure: The incremental model

Evolutionary Process Model

- **Motivation**: a set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined.
- Evolutionary process models are explicitly designed to accommodate a product that evolves over time.
- Evolutionary process models are iterative in nature.
- Types of evolutionary process models:
  1. **Prototyping**
  2. **The Spiral Model**

Prototyping

- **Motivation**: used when the software requirements are fuzzy.
  - Customer defines a set of general objectives for software, but does not identify detailed requirements for functions and features.
  - Developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take.

Prototyping (cont.)

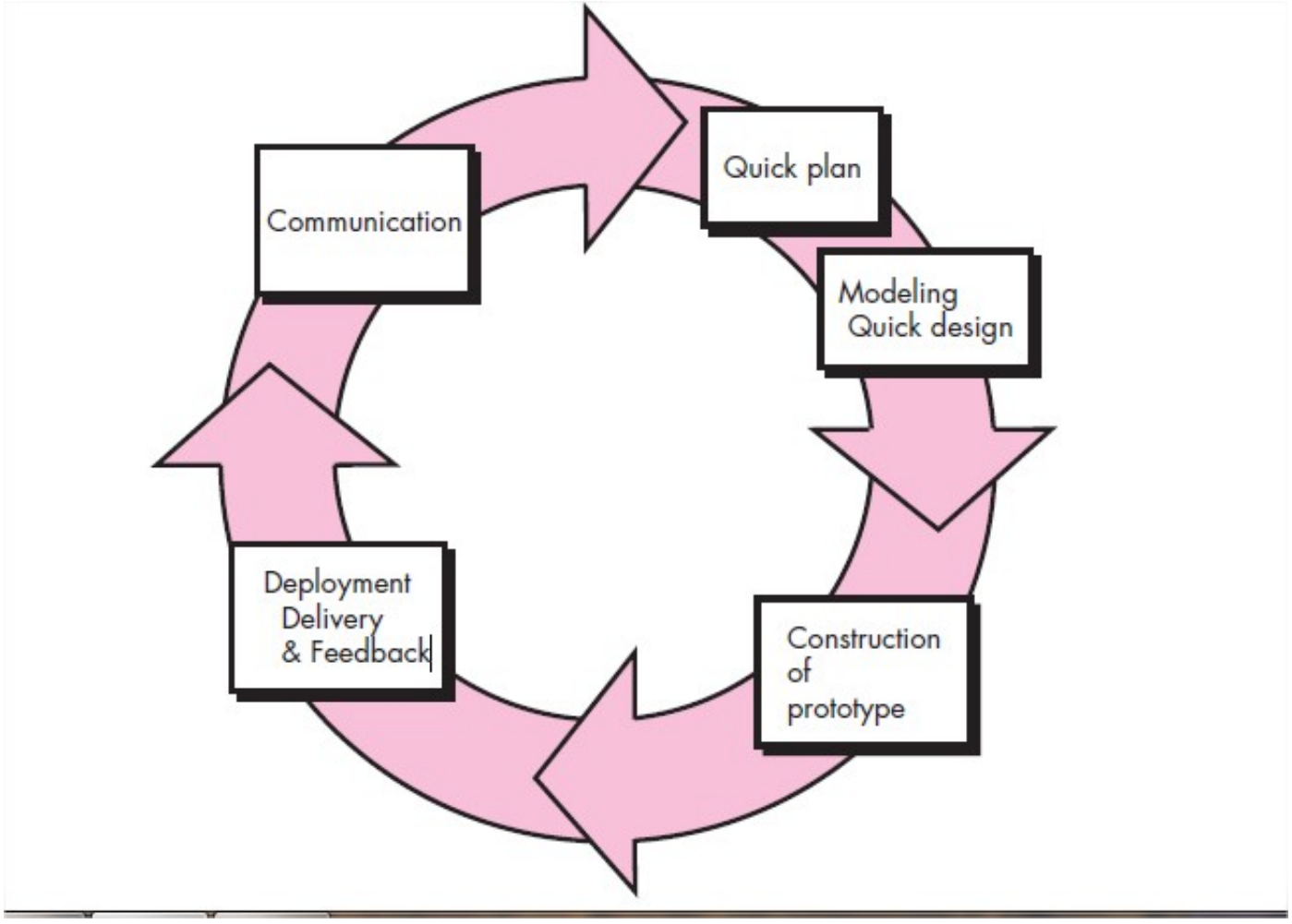


Figure: The prototyping paradigm

Prototyping (cont.)

- Prototyping serves as a mechanism of identifying software requirements.
- Prototyping make use of templates.
- Prototype models can be of following types:
  1. **Rapid application development (RAD)**: high speed adaptation of linear sequential model.
     $\rightarrow$ used to develop organic mode software (no database, development duration: 30 - 90 days).
  2. **Throwaway Prototype**: make use of more than one template.
  3. **Evolutionary Prototype**: make use of only one template (prototype slowly evolve into the actual system).

The Spiral Model

- Proposed by Barry Boehm (1988).
- Couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.
- Enables the rapid development of more complete version of the software (final product).
- Risk-driven process model.
- Unlike other process models that end when software is delivered, the spiral model can be adopted to apply throughout the life cycle of the computer software.

The Spiral Model (cont.)

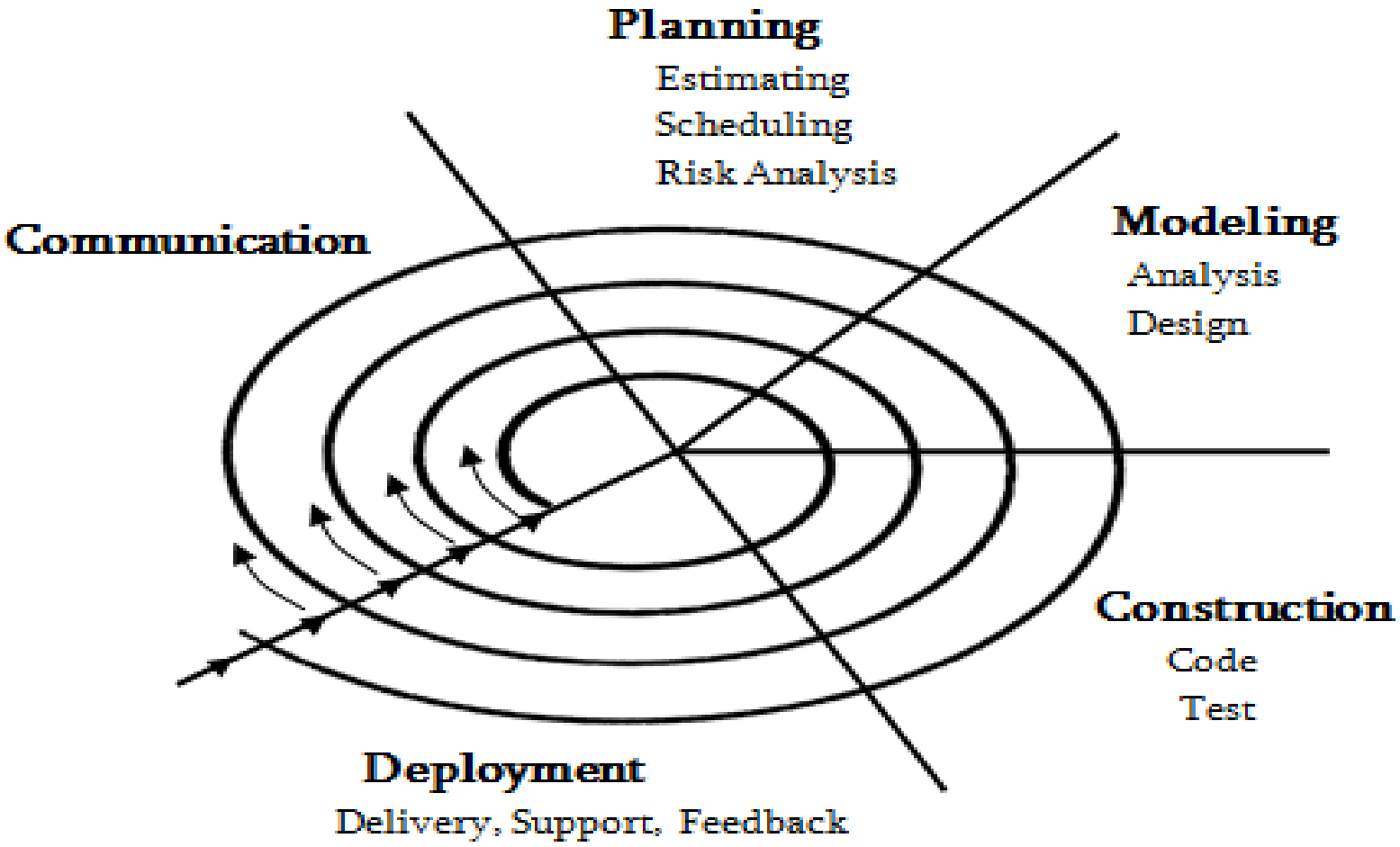


Figure: The typical spiral model

Concurrent Process Model

- Allows a software team to represent iterative and concurrent elements of any of the process model.
- For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the software engineering actions: prototyping, analysis, and design.
- The modeling activity may be in any one of the states noted at any given time.

Concurrent Process Model (cont.)

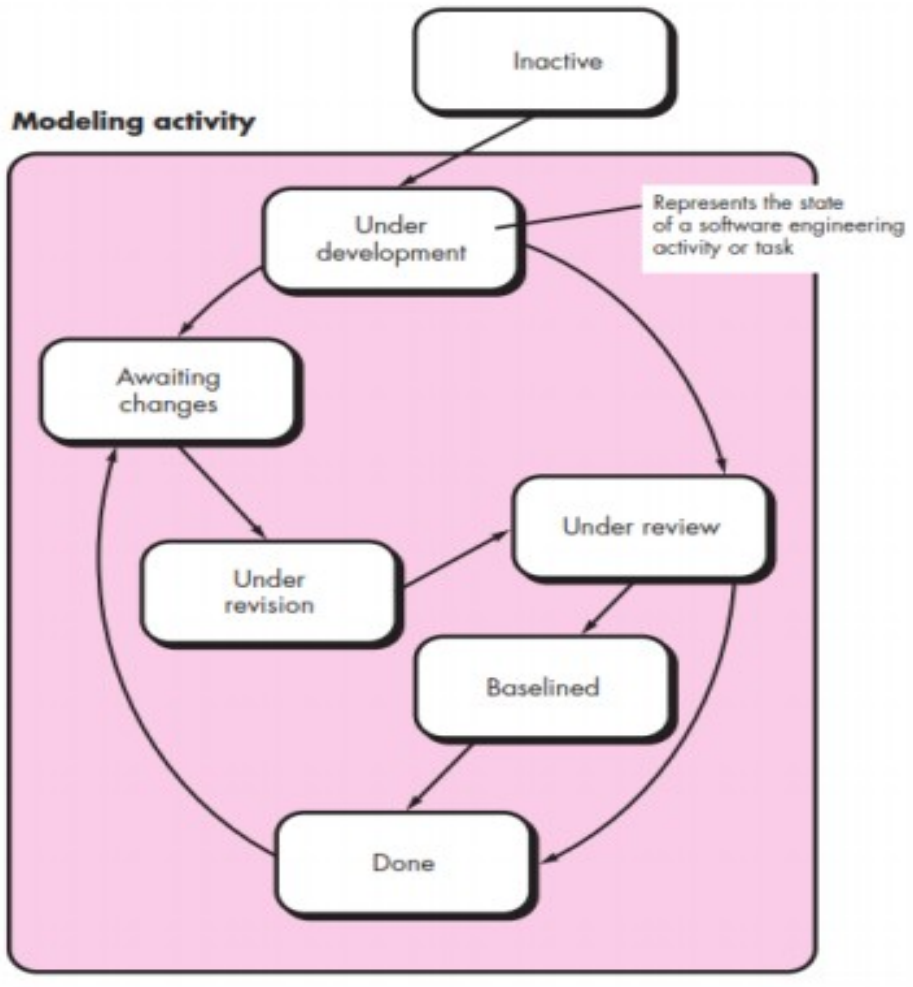


Figure: Concurrent Process Model

Figure: One element of the concurrent process model

Concurrent Process Model (cont.)

- The modeling activity which existed in the inactive state while initial communication was completed, now makes a transition into the under development state.
  If, however, the customer indicates that changes in requirements must be made, the modeling activity moves from the under development state into the awaiting changes state.
- Similarly, other activities, actions, or tasks (e.g., communication or construction) can be represented in an similar manner.
- All software engineering activities exist concurrently but reside in different states.
- Concurrent modeling defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions, or tasks.
- Ideal for small projects with minimal duration.

Component-based Development Model

- Focus is on re-usability of software components or object-oriented classes or packages.
- Constructs applications from prepackaged software components for e.g. commercial-off-the-shelf (COTS) software components.
- **Advantage**
  - Reduction in development cycle time and
  - Reduction in project cost.

Component-based Development Model (cont.)

- Steps involved in Component-based Development Model:
  - Available component-based products are researched and evaluated for the application domain in question.
  - Component integration issues are considered.
  - A software architecture is designed to accommodate the components.
  - Components are integrated into the architecture.
  - Comprehensive testing is conducted to ensure proper functionality.

The Formal Methods-based Model

- Use set of activities that lead to formal mathematical specification of computer software.
- Enable programmer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notations.
- If used properly, formal methods provides a mechanism for eliminating problems (e.g., ambiguity, incompleteness, inconsistency) that are difficult to overcome using other software engineering paradigms.
- Promise defect-free product.
- The development of formal models is quite time consuming and expensive.
- Mostly used for building safety-critical softwares and scientific softwares.

The Unified Process Model

- A framework for object-oriented software engineering using unified modeling language (UML).
- UML provides robust notation for the modeling and development of object-oriented systems.

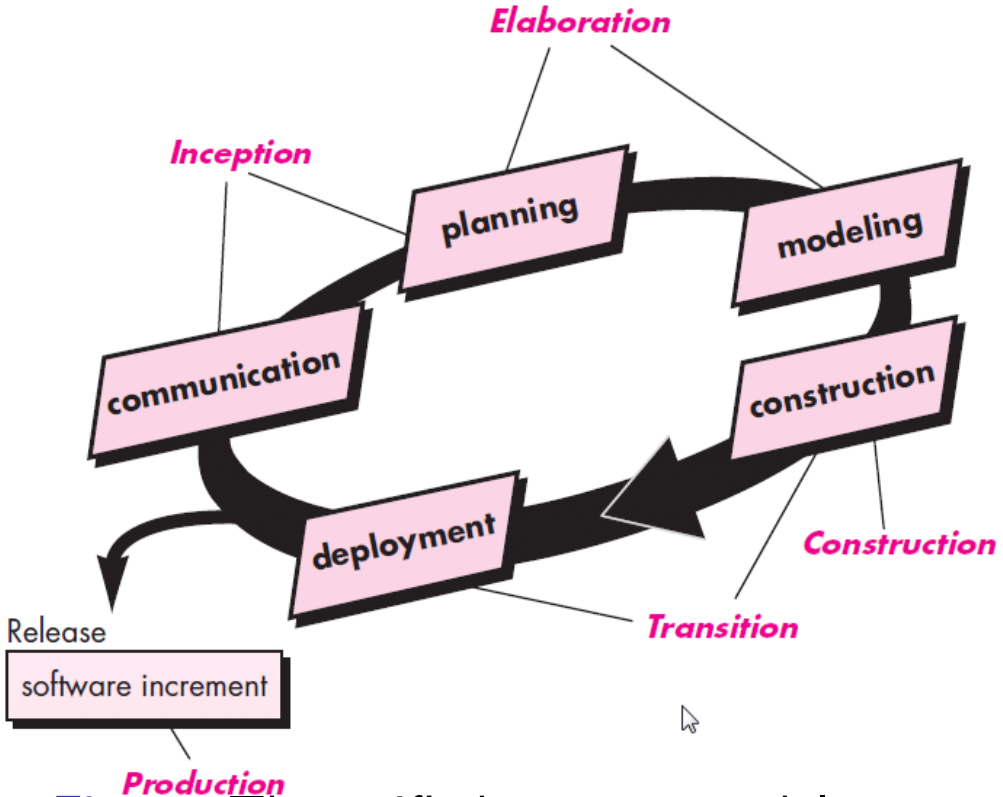


Figure: The unified process model

The Unified Process Model (cont.)

Phases of the Unified Process Model

- **Inception Phase**
  - Encompasses both customer communication and planning activities.
  - With the help of stakeholders, business requirements for the software are identified; a rough architecture for the system is proposed; and a plan for the iterative, incremental nature of the ensuing project is developed.
  - Planning identifies resources, assesses major risks, defines a schedule, and establishes a basis for the phases that are to be applied as the software increment is developed.

The Unified Process Model (cont.)

- **Elaboration Phase**
  - Encompasses the planning and modeling activities of the generic process model.
  - Elaboration refines and expands the preliminary use cases and the architectural representation to include five different views of the software - the use case model, the requirement model, the design model, the implementation model, and the deployment model.
- **Construction Phase**
  - Using the architectural model as input, the construction phase develops or acquires the software components that will make each use case operational for end users.
  - Unit tests are designed and executed for each software component.
  - Integration activities (component assembly and integration testing) are conducted.

The Unified Process Model (cont.)

- **Transition Phase**
  - Software is given to end users for beta testing and user feedback is collected to report both defects and necessary changes.
  - Software team creates the necessary support documents (e.g., user manuals, troubleshooting guides, installation procedures) that is required for the usable software release.
- **Production Phase**: coincides with the deployment phase of the generic process.
  - The ongoing use of the software is monitored, support for the operating environment (infrastructure) is provided, and defect reports and request for changes are submitted and evaluated.

Personal and Team Process Model

- **Key ingredients**: measurement, planning, and self-direction.
- **Personal Software Process (PSP)**
  - Focus on personal measurement of both the work product that is produced and the resultant quality of the work product.
  - Practitioner is responsible for project planning (e.g., estimating and scheduling).
  - Empowers the practitioner to control the quality of all software work products that are developed.
  - **PSP model activities**: planning, high-level design, high-level design review, development, and postmortem.
- **Team Software Process (TSP)**
  - **Goal**: building a " self-directed" project team that organizes itself to produce high-quality software.
  - **PSP model activities**: project launch, high-level design, implementation, integration and test, and postmortem.

Capability Maturity Model (CMM)

- CMM measures the effectiveness of the software process.
  - **Microsoft** - Synchronized/Stabilized Process Model
  - **IBM** - Rationalized Unified Process Model (RUP)
  - **Infosys** - Waterfall Model $\Rightarrow$ Spiral Model.
- Based on the principle of total quality management (TQM).

Capability Maturity Model (CMM) (cont.)

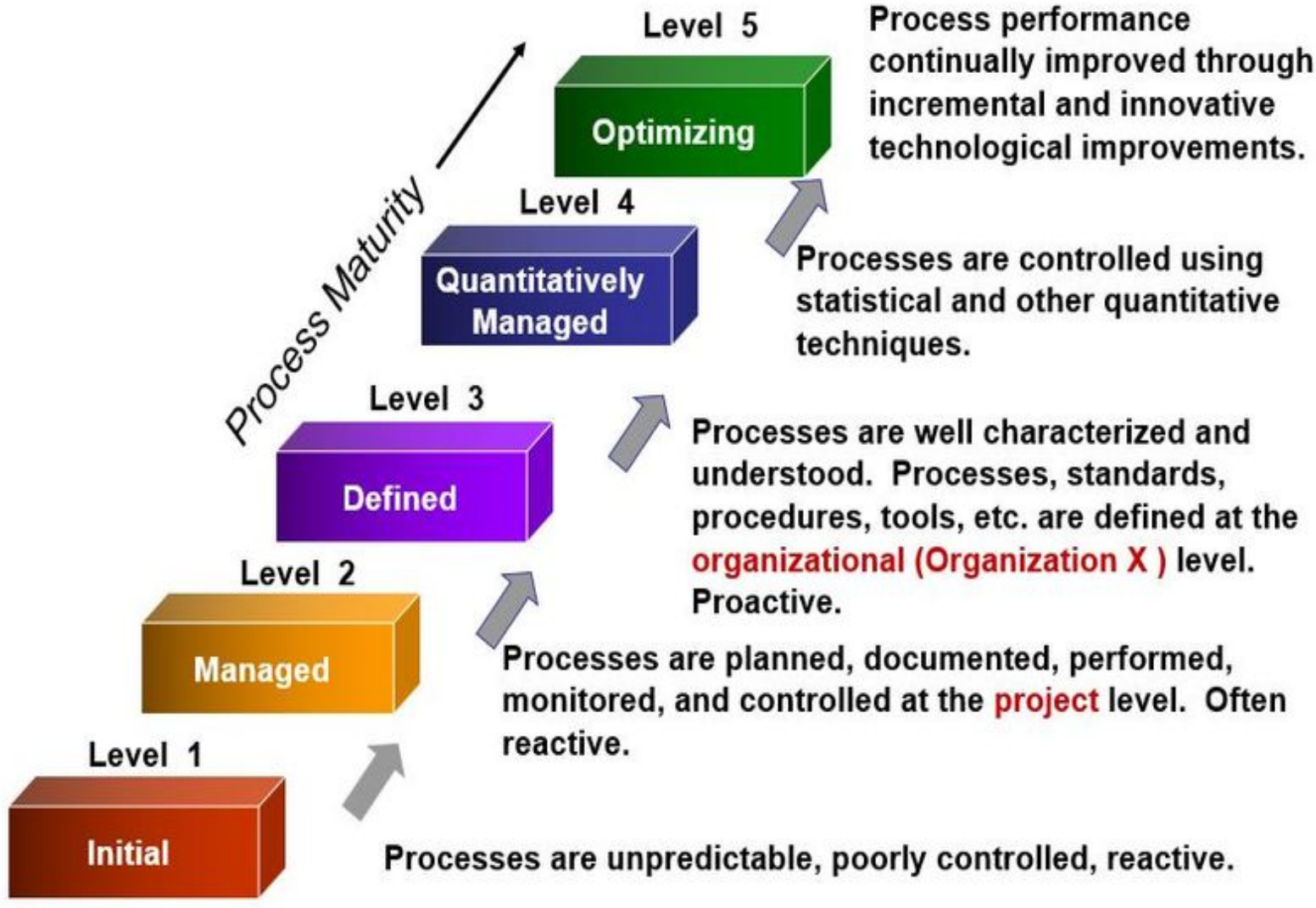


Figure: Capability Maturity Model