UNIT-I: Software Engineering & Process Models

Dual Role of Software

- Both a product and a vehicle for delivering a product
  - Product
    - Delivers computing potential
    - Produces, manages, acquires, modifies, display, or transmits information
  - Vehicle
    - Supports or directly provides system functionality
    - Controls other programs (e.g., operating systems)
    - Effects communications (e.g., networking software)

Helps build other software (e.g., software tools)

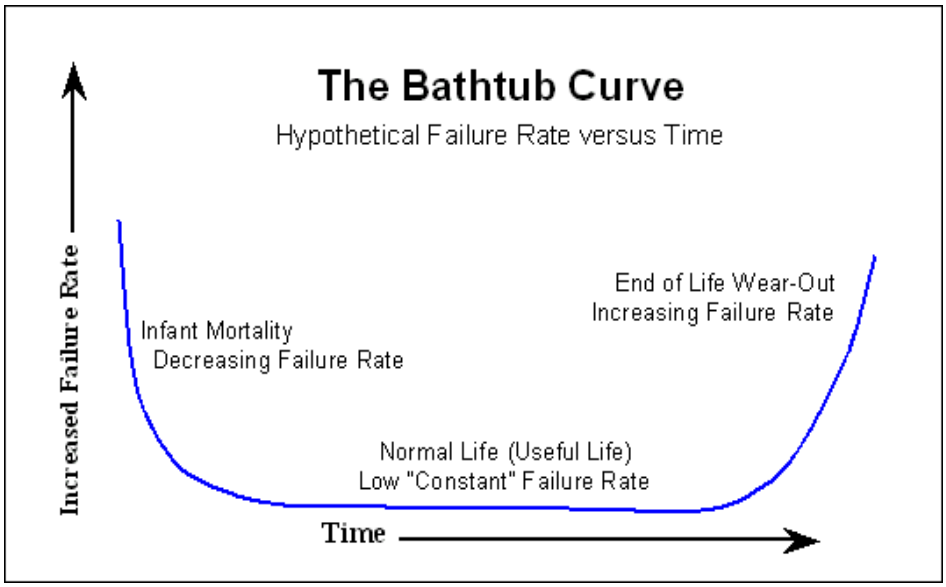
A Definition of Software

- **Instructions** (computer programs) that when executed provide desired features, function, and performance
- **Data structures** that enable the programs to adequately manipulate information
- **Documents** that describe the operation and use of the programs

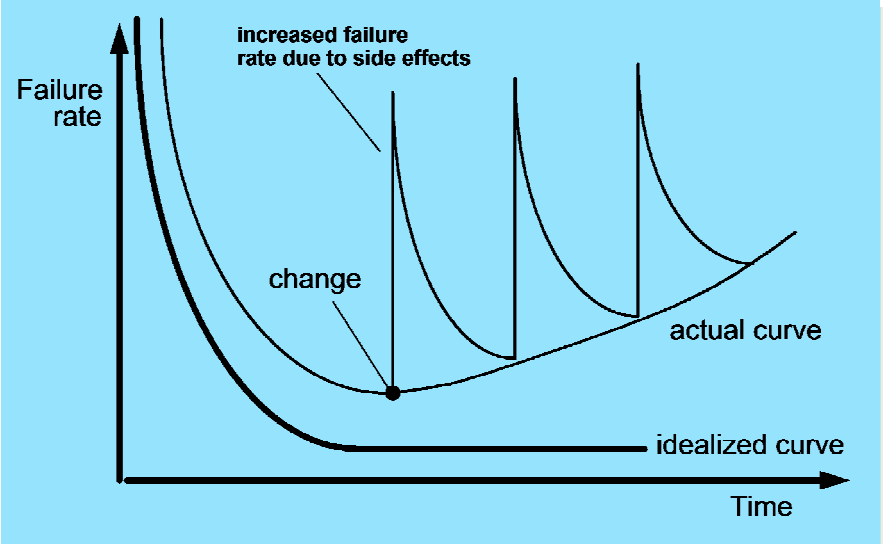
Differences between Software and Hardware

- Software is developed or engineered; it is not manufactured in the classical sense
  - Impacts the management of software projects
- Software doesn't wear out
  - Hardware bathtub curve compared to the software ascending spiked curve
- Industry is moving toward component-based construction, most software continues to be custom built
  - it is still complex to build
  - Reusable components are created so that engineers can concentrate on innovative elements of a design
  - User interfaces are built with reusable components
  - The data structures and processing details are kept in a library for interface construction

Hardware Failure Curve



Software Failure Curve



Changing Nature of Software

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software (e.g., inventory control, word processing, multimedia)
- Web applications
- Artificial intelligence software
- Ubiquitous computing (small, wireless devices)
- Netsourcing (net-wide computing)
- Open source (operating systems, databases, development environments)
- The ".com" marketing applications

Legacy Software – Characteristics

- Support core business functions
- Have longevity and business criticality
- Exhibit poor quality
  - Convoluted code, poor documentation, poor testing, poor change management

Reasons for Evolving the Legacy Software

- (Adaptive) Must be adapted to meet the needs of new computing environments or more modern systems, databases, or networks
- (Perfective) Must be enhanced to implement new business requirements
- (Corrective) Must be changed because of errors found in the specification, design, or implementation

Software Myths – Management

- "We already have a book that is full of standards and procedures for building software. Won't that provide my people with everything they need to know?"
  - Not used, not up to date, not complete, not focused on quality, time, and money
- "If we get behind, we can add more programmers and catch up"
  - Adding people to a late software project makes it later
  - Training time, increased communication lines

- "If I decide to outsource the software project to a third party, I can just relax and let that firm build it"
  - Software projects need to be controlled and managed

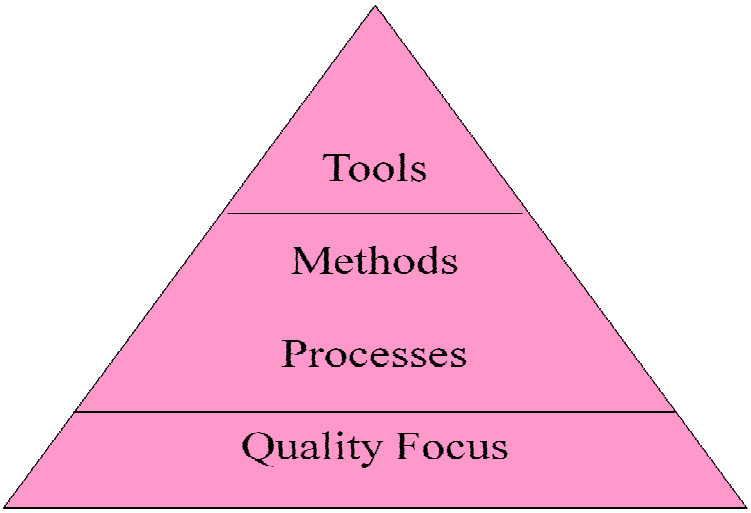
Software Myths – customer

- "A general statement of objectives is sufficient to begin writing programs – we can fill in the details later"
  - Ambiguous statement of objectives spells disaster
- "Project requirements continually change, but change can be easily accommodated because software is flexible"
  - Impact of change depends on where and when it occurs in the software life cycle (requirements analysis, design, code, test)

Software Myths - Practitioner

- "Once we write the program and get it to work, our job is done"
  - 60% to 80% of all effort expended on software occurs after it is delivered
- "Until I get the program running, I have no way of assessing its quality
  - Formal technical reviews of requirements analysis documents, design documents, and source code (more effective than actual testing)
- "The only deliverable work product for a successful project is the working program"
  - Software, documentation, test drivers, test results
- "Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down"
  - Creates quality, not documents; quality reduces rework and provides software on time and within the budget

Software Engineering is a Layered Technology



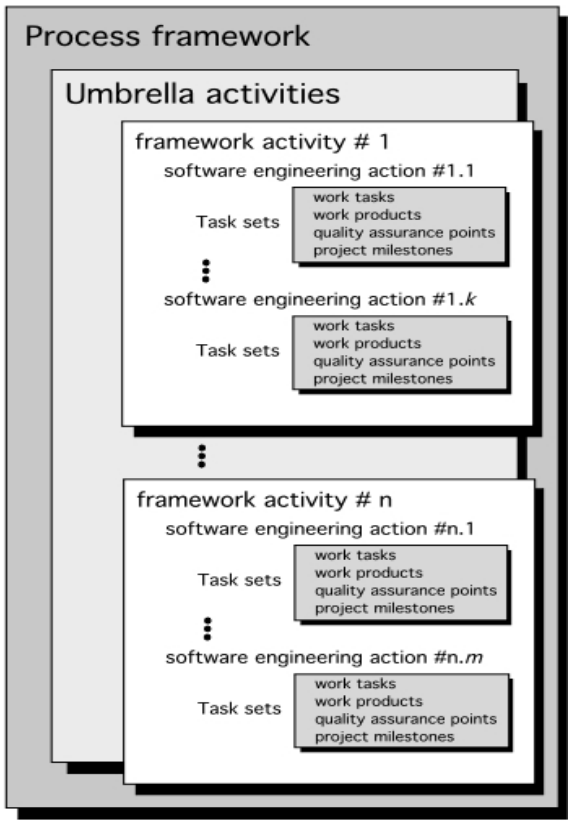
- A Quality Focus
  - Engineering approach must rely on quality
  - Total Quality Management (TQM), six sigma leads to continuous improvement in culture.
  - This culture ultimately helps to develop more effective approaches to SE.
- Process
  - Provides the glue that holds the layers together;
  - enables balanced and timely development;

  - provides a framework for effective delivery of technology;
  - forms the basis for management; provides the context for technical methods, work products, milestones, quality measures, and change management
- Methods
  - Provide the technical "how to" for building software;
  - rely on a set of basic principles;
  - encompass a broad array of tasks; include modeling activities
- Tools
  - Provide automated or semi-automated support for the process and methods (i.e., CASE tools)

Generic Process Framework

- Communication
  - Involves communication among the customer and other stake holders; encompasses requirements gathering
- Planning
  - Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule
- Modeling (Analyze, Design)
  - Encompasses the creation of models to better understand the requirements and the design
- Construction (Code, Test)
  - Combines code generation and testing to uncover errors
- Deployment
  - Involves delivery of software to the customer for evaluation and feedback

Software process



What is a Process?

- (Webster) A system of operations in producing something; a series of actions, changes, or functions that achieve an end result
- (IEEE) A sequence of steps performed for a given purpose

What is a Software Process?

- A set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals)
- As an organization matures, the software process becomes better defined and more consistently implemented throughout the organization
- Software process maturity is the extent to which a specific process is explicitly defined, managed, measured, controlled, and effective
- Prescriptive Process Models

Process Models

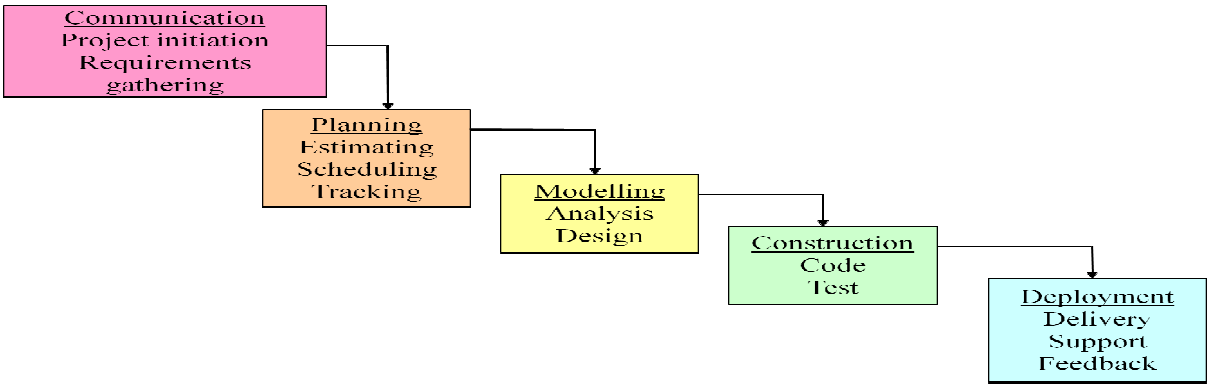
- Generic process framework (revisited)
- Traditional process models
- Specialized process models
- The unified process
- Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software
- The activities may be linear, incremental, or evolutionary

Generic Process Framework

Communication
Involves communication among the customer and other stake holders; encompasses requirements gathering
Planning
Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule
Modelling (Analyze, Design)
Encompasses the creation of models to better under the requirements and the design
Construction (Code, Test)
Combines code generation and testing to uncover errors
Deployment
Involves delivery of software to the customer for evaluation and feedback

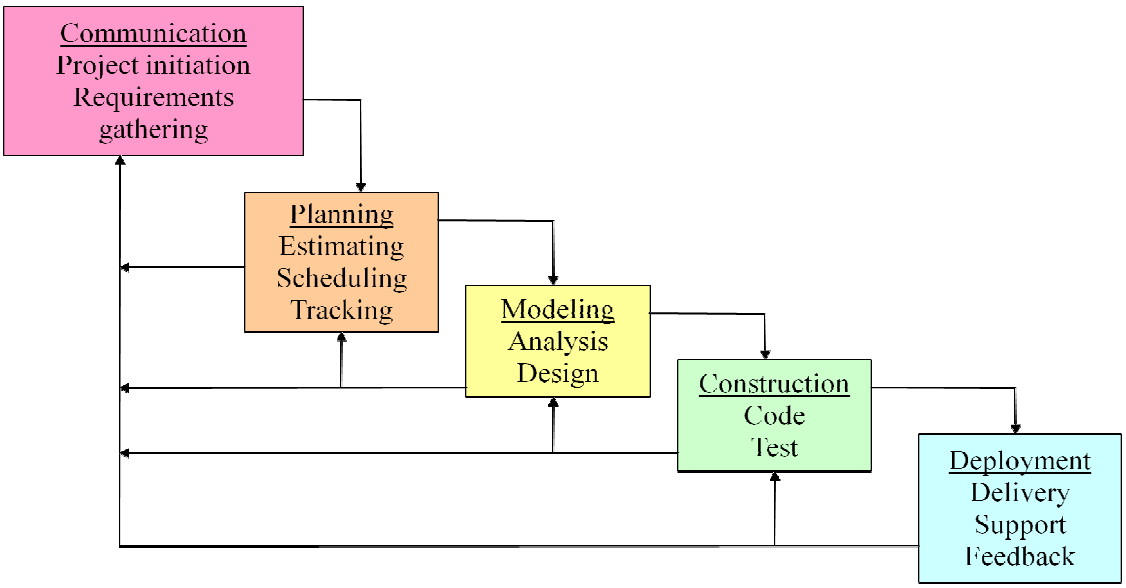
Traditional Process Models

The Waterfall Model

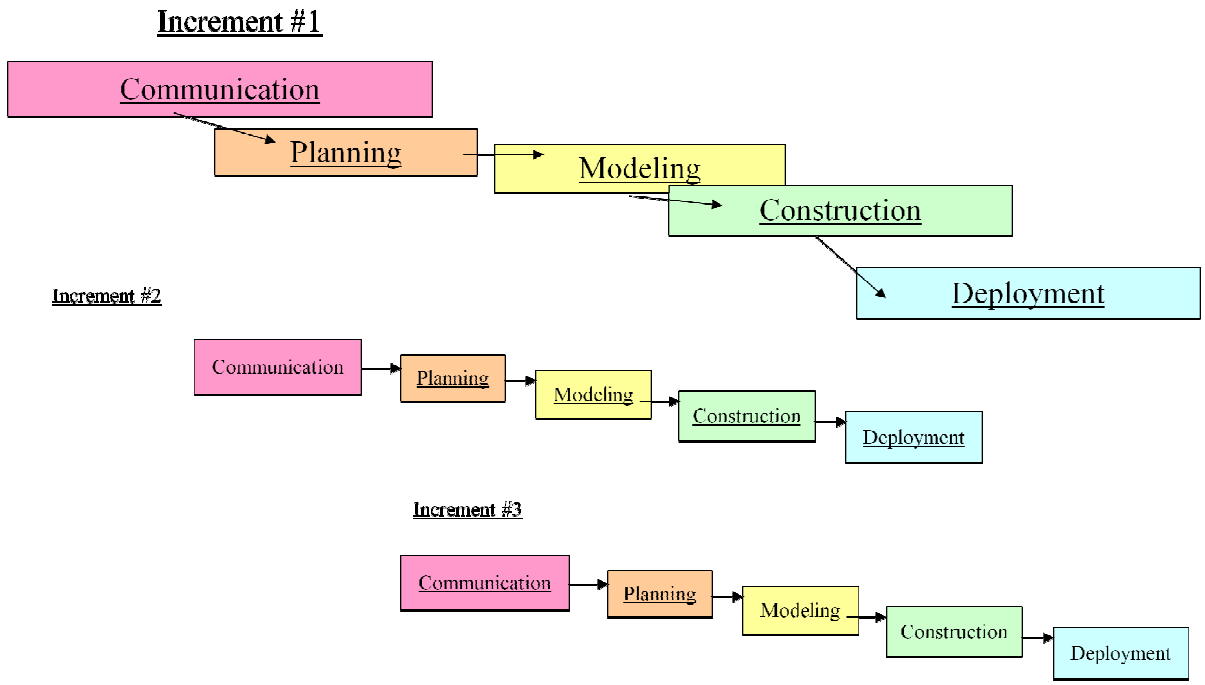


- Sometimes called the *classic life cycle*
- Suggests a systematic, sequential (or linear) approach to s/w development
- The oldest paradigm for s/w engineering
- Works best when –
  - Requirements of a problem are reasonably well understood
  - Well-defined adaptations or enhancements to an existing system must be made
  - Requirements are well-defined and reasonably stable
- Oldest software lifecycle model and best understood by upper management
- Used when requirements are well understood and risk is low
- Work flow is in a linear (i.e., sequential) fashion
- Used often with well-defined adaptations or enhancements to current software
- Doesn't support iteration, so changes can cause confusion
- Difficult for customers to state all requirements explicitly and up front
- Requires customer patience because a working version of the program doesn't occur until the final phase
- Problems can be somewhat alleviated in the model through the addition of feedback loops

The Waterfall Model with Feedback



Incremental Model



- Combines elements of the waterfall model applied in an iterative fashion
- Each linear sequence produces deliverable "increments" of the software
- The first increment is often a *core product*
- The core product is used by the customer (or undergoes detailed evaluation)
- Based on evaluation results, a plan is developed for the next increment
- Used when requirements are well understood
- Multiple independent deliveries are identified
- Work flow is in a linear (i.e., sequential) fashion within an increment and is staggered between increments
- Iterative in nature; focuses on an operational product with each increment
- Provides a needed set of functionality sooner while delivering optional components later
- Useful also when staffing is too short for a full-scale development
- The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature
- But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment
- Increments can be planned to manage technical risks

Advantages of Incremental model:

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it'd iteration.

Disadvantages of Incremental model:

- Needs good planning and design.

- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than **waterfall** model.

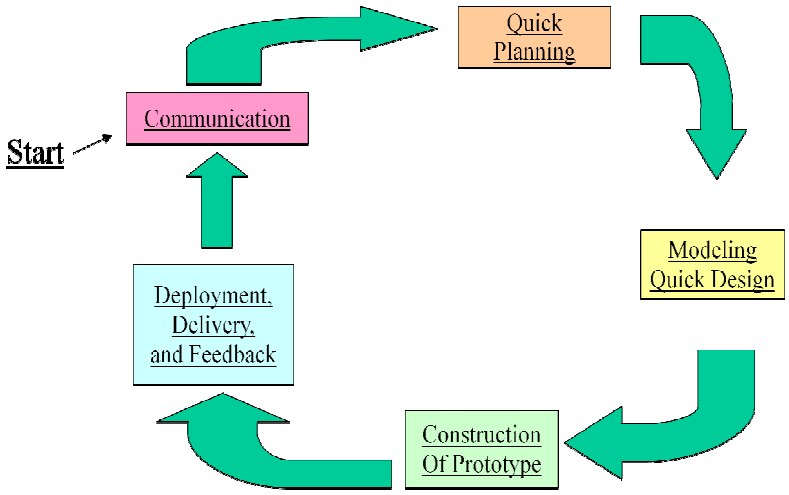
Evolutionary Process Models

Software, like all complex systems, evolves over a period of time

Business and product requirements often change as development proceeds, making a straight-line path to an end product is unrealistic

Evolutionary models are iterative

Prototyping Model



- Follows an evolutionary and iterative approach
- Used when requirements are not well understood
- Serves as a mechanism for identifying software requirements
- Customers may press for immediate delivery of working but inefficient products
- Focuses on those aspects of the software that are visible to the customer/user
- The developer often makes implementation compromises in order to get a prototype working quickly
- Feedback is used to refine the prototype
- Potential Problems
- The customer sees a "working version" of the software, wants to stop all development and then buy the prototype after a "few fixes" are made
- Developers often make implementation compromises to get the software running quickly (e.g., language choice, user interface, operating system choice, inefficient algorithms)
- Lesson learned
    - Define the rules up front on the final disposition of the prototype before it is built
    - In most circumstances, plan to discard the prototype and engineer the actual production software with a goal toward quality

The Spiral Model

Couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model

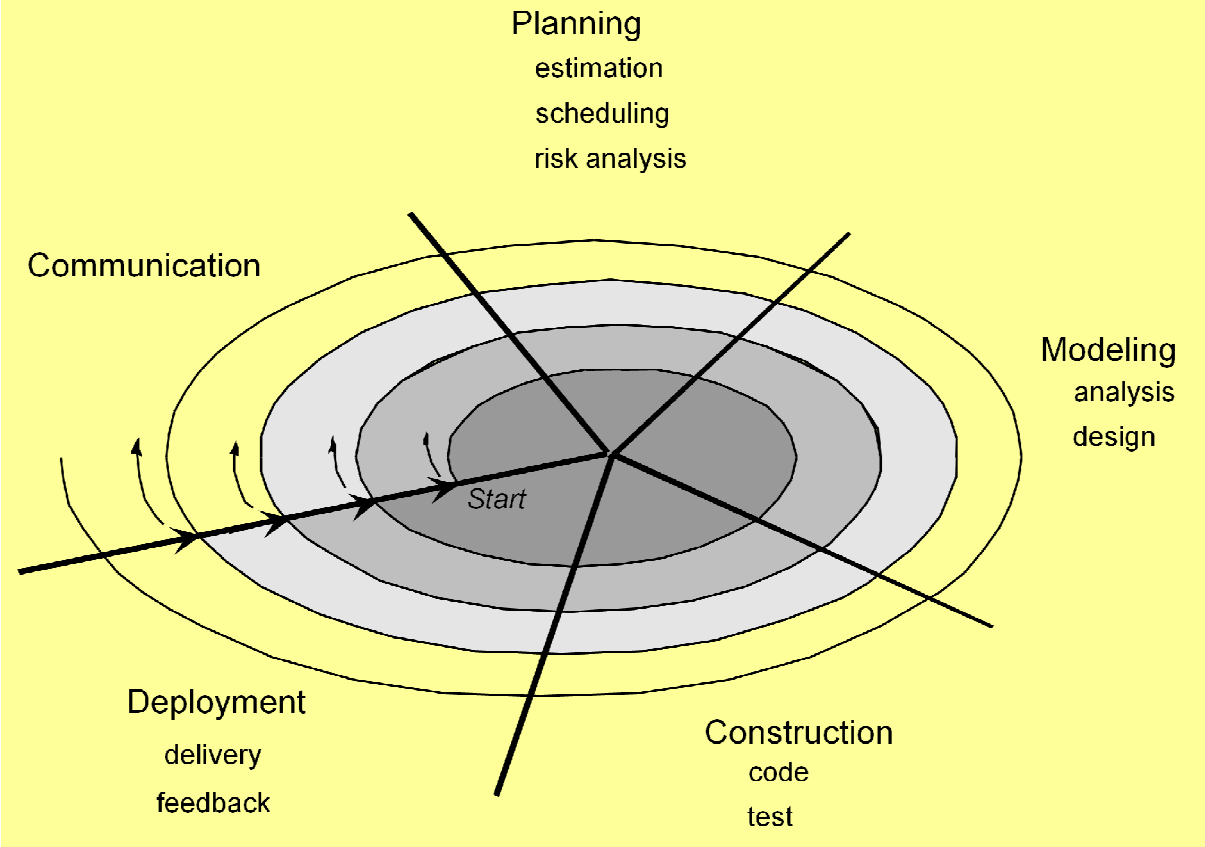
It provides the potential for rapid development of increasingly more complete versions of the software

It is a *risk-driven process model* generator

It has two main distinguishing features

Cyclic approach

Incrementally growing a system's degree of definition and implementation while decreasing its degree of risk
A set of *anchor point milestones*
A combination of work products and conditions that are attained along the path of the spiral



- Invented by Dr. Barry Boehm in 1988
- Follows an evolutionary approach
- Used when requirements are not well understood and risks are high
- Inner spirals focus on identifying software requirements and project risks; may also incorporate prototyping
- Outer spirals take on a classical waterfall approach after requirements have been defined, but permit iterative growth of the software
- Operates as a risk-driven model…a go/no-go decision occurs after each complete spiral in order to react to risk determinations
- Requires considerable expertise in risk assessment
- Serves as a realistic model for large-scale software development

Drawbacks

It may be difficult to convince customers (particularly in contract situations) that the evolutionary approach is controllable.
It demands considerable risk assessment expertise and relies on this expertise for success. If a major risk is not uncovered and managed, problems will undoubtedly occur.
Weaknesses of Evolutionary Process Models

1) Prototyping poses a problem to project planning because of the uncertain number of iterations required to construct the product
2) Evolutionary software processes do not establish the maximum speed of the evolution
    - If too fast, the process will fall into chaos
    - If too slow, productivity could be affected

3) Software processes should focus first on flexibility and extensibility, and second on high quality
    - We should prioritize the speed of the development over zero defects
    - Extending the development in order to reach higher quality could result in late delivery

Specialized Process Models

Take on many of the characteristics of one or more of the conventional models
Tend to be applied when a narrowly defined software engineering approach is chosen
Examples:
Component-Based Development
Consists of the following process steps
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
Relies on a robust component library
Capitalizes on software reuse, which leads to documented savings in project cost and time

The Formal Methods Model

- Encompasses a set of activities that leads to formal mathematical specification of computer software
- Enables a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation
- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily through mathematical analysis
- Offers the promise of defect-free software
- Used often when building safety-critical systems

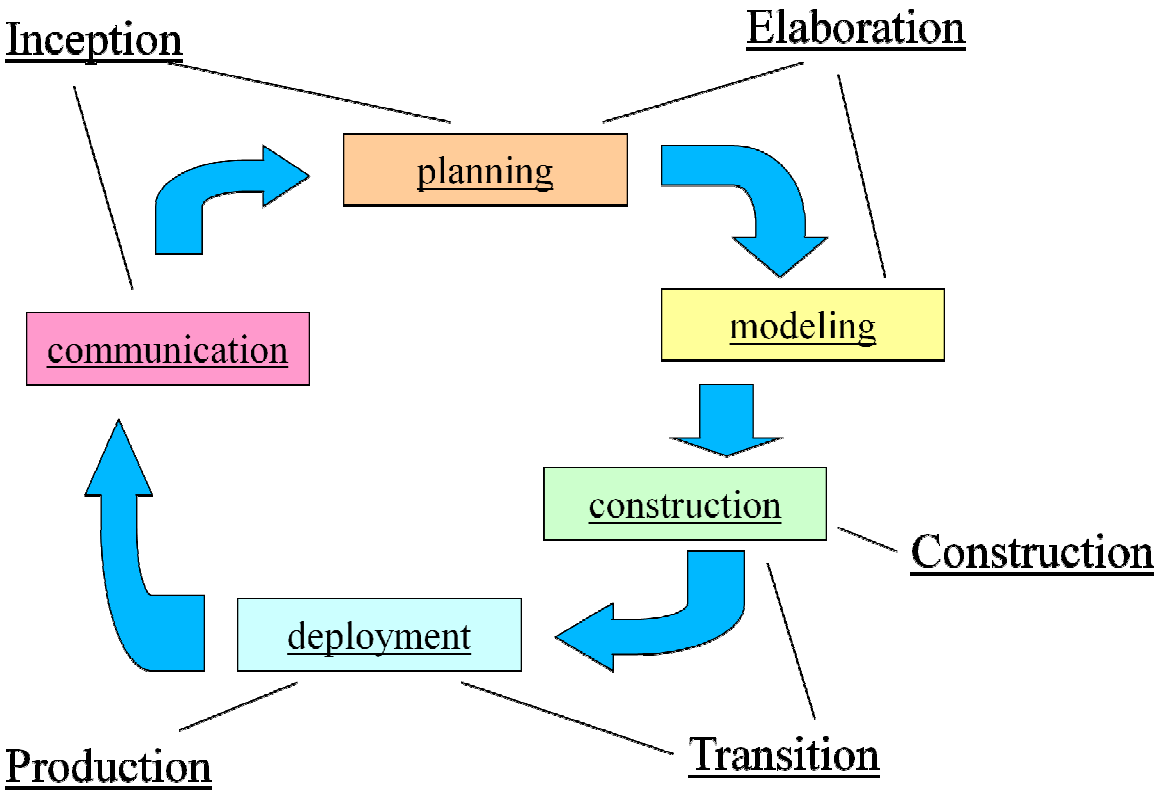
Challenges:

- Development of formal methods is currently quite time-consuming and expensive
- Because few software developers have the necessary background to apply formal methods, extensive training is required
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers

Aspect-Oriented Software Development

Certain “concerns” – customer required properties or areas of technical interest – span the entire s/w architecture
Example “concerns”
Security
Fault Tolerance
Task synchronization
Memory Management
When concerns cut across multiple system functions, features, and information, they are often referred to as *crosscutting concerns*

Phases of the Unified Process



Booch, Jacobson, and Rumbaugh later developed the unified process, which is a framework for object-oriented software engineering using UML
Draws on the best features and characteristics of conventional software process models
Emphasizes the important role of software architecture
Consists of a process flow that is iterative and incremental, thereby providing an evolutionary feel
Consists of five phases: inception, elaboration, construction, transition, and production
Inception Phase

- Encompasses both customer communication and planning activities of the generic process
- Business requirements for the software are identified
- A rough architecture for the system is proposed
- A plan is created for an incremental, iterative development
- Fundamental business requirements are described through preliminary use cases
  - A use case describes a sequence of actions that are performed by a user

Construction Phase

- Encompasses the construction activity of the generic process
- Uses the architectural model from the elaboration phase as input
- Develops or acquires the software components that make each use-case operational
- Analysis and design models from the previous phase are completed to reflect the final version of the increment
- Use cases are used to derive a set of acceptance tests that are executed prior to the next phase

Transition Phase

- Encompasses the last part of the construction activity and the first part of the deployment activity of the generic process
- Software is given to end users for beta testing and user feedback reports on defects and necessary changes
- The software teams create necessary support documentation (user manuals, trouble-shooting guides, installation procedures)

- At the conclusion of this phase, the software increment becomes a usable software release

Production Phase

- Encompasses the last part of the deployment activity of the generic process
- On-going use of the software is monitored
- Support for the operating environment (infrastructure) is provided
- Defect reports and requests for changes are submitted and evaluated

Personal and team process models

What Is Personal Software Process (PSP)?

- The Personal Software Process (PSP) shows engineers how to
  - manage the quality of their projects
  - make commitments they can meet
  - improve estimating and planning
  - reduce defects in their products

PSP emphasizes the need to record and analyze the types of errors you make, so you can develop strategies to eliminate them.

Personal Software Process

- Because personnel costs constitute 70 percent of the cost of software development, the skills and work habits of engineers largely determine the results of the software development process.
- Based on practices found in the CMMI, the PSP can be used by engineers as a guide to a disciplined and structured approach to developing software. The PSP is a prerequisite for an organization planning to introduce the TSP.

What Is Team Software Process (TSP)?

- The Team Software Process (TSP), along with the Personal Software Process, helps the high-performance engineer to

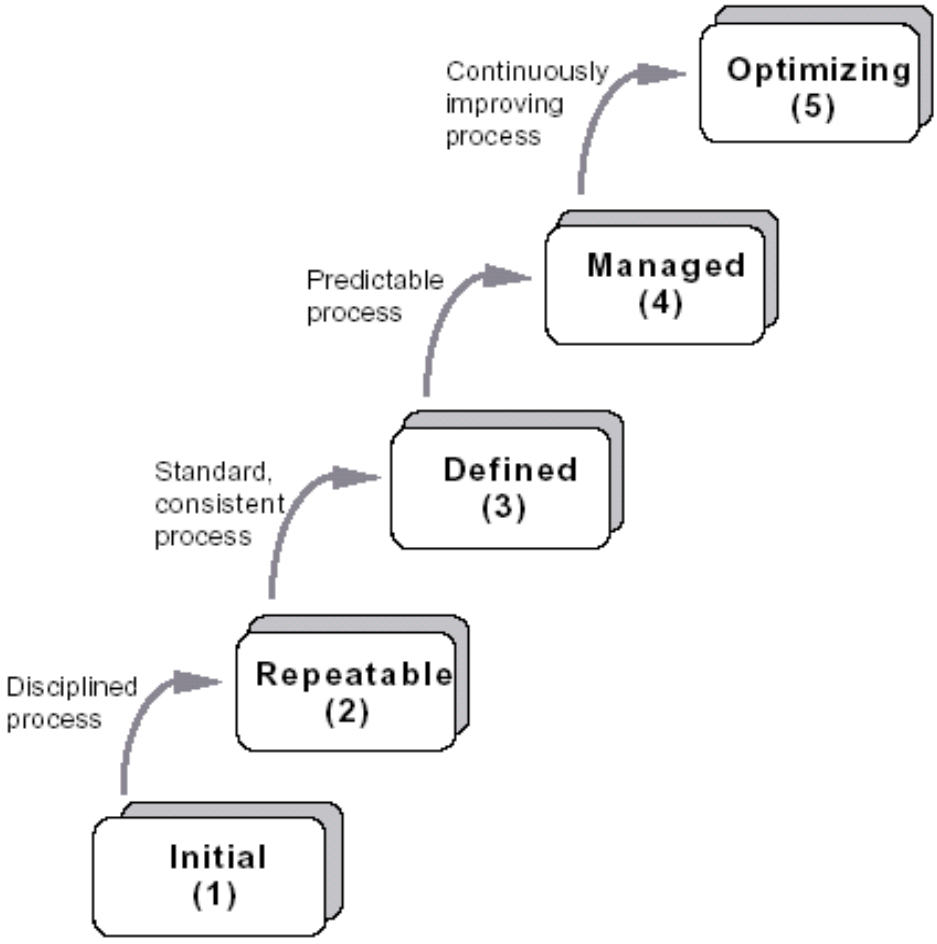
- ensure quality software products
- create secure software products
- improve process management in an organization

PSP Framework Activities	TSP Framework Activities
Planning	Launch high level design
High level Design	Implementation
High level Design Review	Integration
Development	Test
Postmortem	postmortem

Immature Software Organizations

- Software processes are generally improvised
- If a process is specified, it is not rigorously followed or enforced
- The software organization is reactionary
- Managers only focus on solving immediate (crisis) problems
- Schedules and budgets are routinely exceeded because they are not based on realistic estimates
- When hard deadlines are imposed, product functionality and quality are often compromised
- There is no basis for judging process quality or for solving product or process problems
- Activities such as reviews and testing are curtailed or eliminated when projects fall behind schedule

Five Levels of Software Process Maturity



Characteristics of Each Level

- Incomplete: not performed any process activity, or requirements management
- Initial Level / Performed (Level 1)
  - Characterized as ad hoc, and occasionally even chaotic
  - Few processes are defined, and success depends on individual effort
- Repeatable / Managed (Level 2)
  - Conforms to an organizational policy
  - Basic project management processes are established to track cost, schedule, and functionality
  - The necessary process discipline is in place to repeat earlier successes on projects with similar applications
- Defined (Level 3)
  - All L-2 criteria have been achieved
  - The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization
  - All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software
- Managed (Level 4)
  - All L-3 criteria have been achieved
  - Detailed measures of the software process and product quality are collected
  - Both the software process and products are quantitatively understood and controlled

- Optimized (Level 5)
  - All L-4 criteria have been achieved
  - The process area is adopted and optimized using quantitative or statistical means to continuously meet changing customer needs and improve the efficiency of the development process
  - Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies

Defined	*Process standardization*	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Project Management Integrated Supplier Management Risk Management Decision Analysis and Resolution Organizational Environment for Integration Integrated Teaming
Managed	*Basic project management*	Requirements Management Project Planning Project Monitoring and Control Supplier Agreement Management Measurement and Analysis Process and Product Quality Assurance Configuration Management
Performed		

Level	Focus	Process Areas
Optimizing	*Continuous process improvement*	Organizational Innovation and Deployment Causal Analysis and Resolution
Quantitatively managed	*Quantitative management*	Organizational Process Performance Quantitative Project Management