Software Engineering (Handouts by – T.Sujanavan, Asst. Prof., MVSREC)

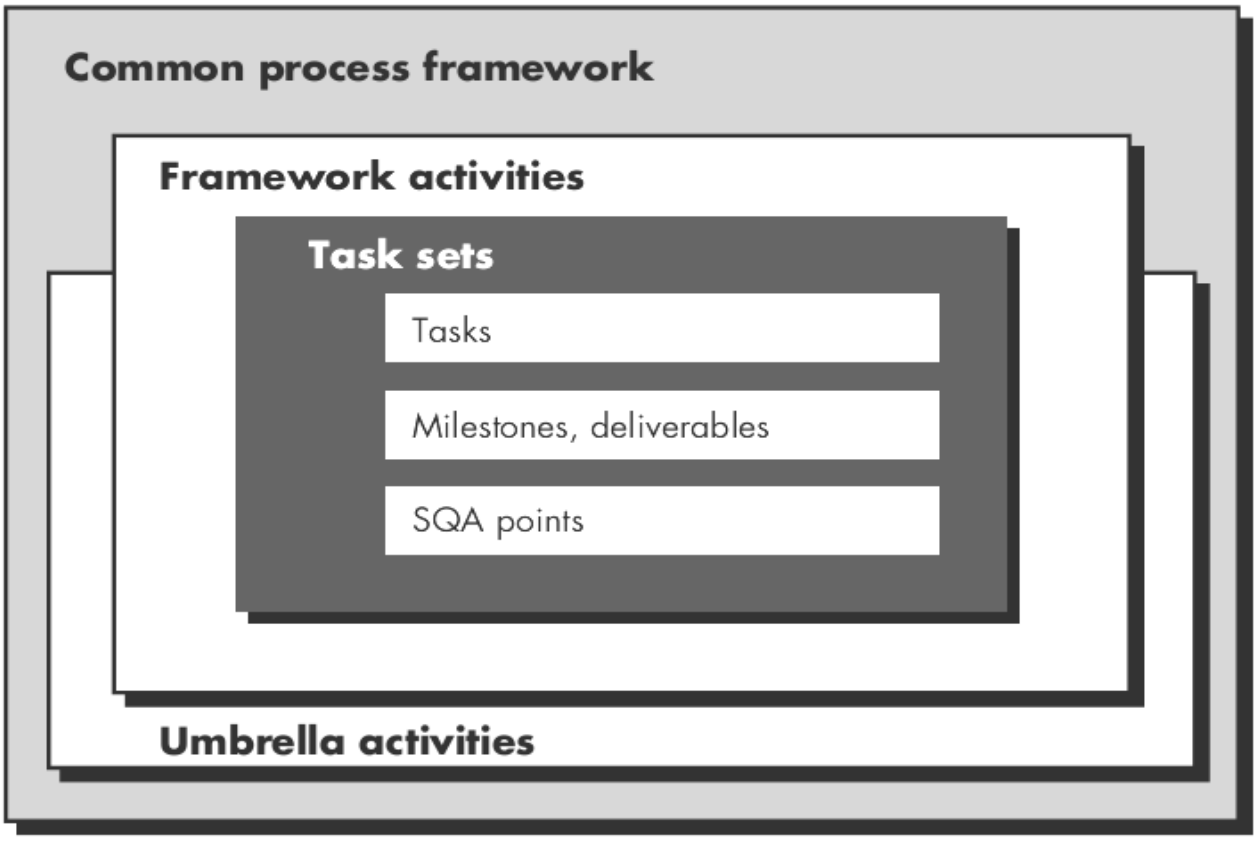
Unit 1

A generic view of process

Software engineering (SE) is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software. It is the application of engineering to software because it integrates significant mathematics, computer science and practices whose origins are in engineering. It is also defined as a systematic approach to the analysis, design, assessment, implementation, testing, maintenance and reengineering of software, that is, the application of engineering to software

- Source : Ghezzi, Carlo; [1991]. Fundamentals of Software Engineering

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. Finally, umbrella activities—such as software quality assurance, soft-ware configuration management, and measurement —overlay the process model. Umbrella activities are independent of any one framework activity and occur through-out the process.



Later, there has been a significant emphasis on "**process maturity**"

The Software Engineering Institute (SEI) has developed a comprehensive model predicated on a set of software engineering capabilities that should be present as organizations reach **different levels of process maturity**. To determine an organization's current state of process maturity, the SEI uses an assessment that results in a five point grading scheme.

The grading scheme determines compliance with a **capability maturity model (CMM)** that defines key activities required at different levels of process maturity. **Five process maturity levels** that are defined in the following manner:

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

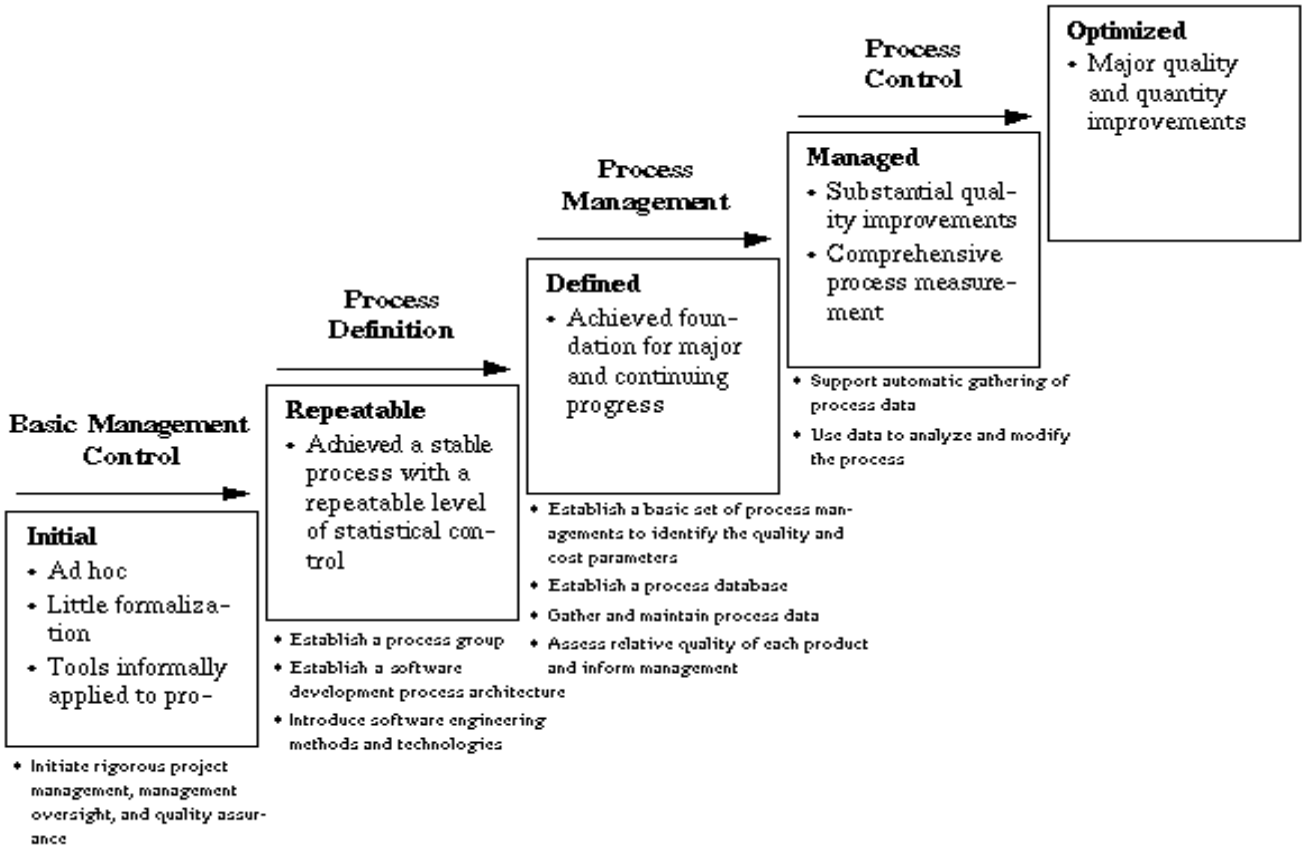
Level 2: Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

Level 3: Defined. The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.

Level 4: Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

Level 5: Optimizing. Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

- Source : Software engg. – a practitioners approach – roger.s



Levels of **Capability Maturity Model (CMM)** – *Source : SEI (Carnegie Mellon)*

Process patterns can be defined as the set of activities, actions, work tasks or work products and similar related behaviour followed in a software process lifecycle.

A process pattern describes a process-related problem that is encountered during software engineering work, identifies the environment in which the problem has been encountered, and suggests one or more proven solution to the problem.

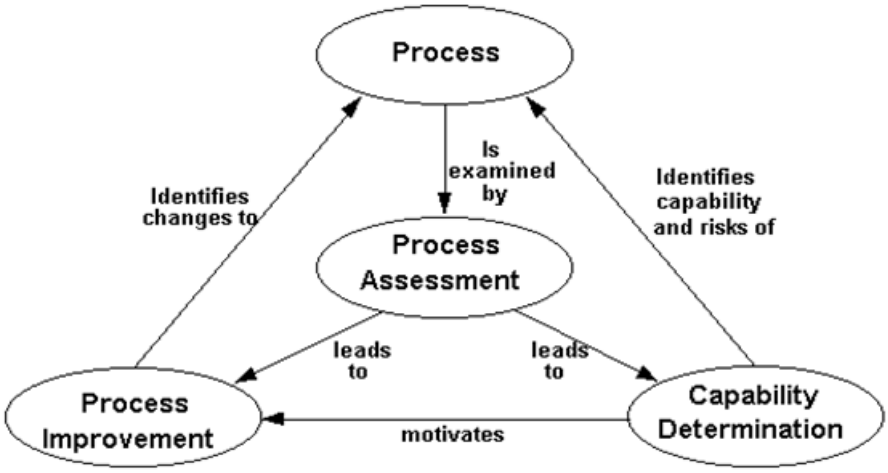
There are three types of process patterns. In order of increasing scale they are:

1. Task process patterns. This type of process pattern depicts the detailed steps to perform a specific task, such as the Technical Review and Reuse First process patterns

2. Stage process patterns. This type of process pattern depicts the steps, which are often performed iteratively, of a single project stage.

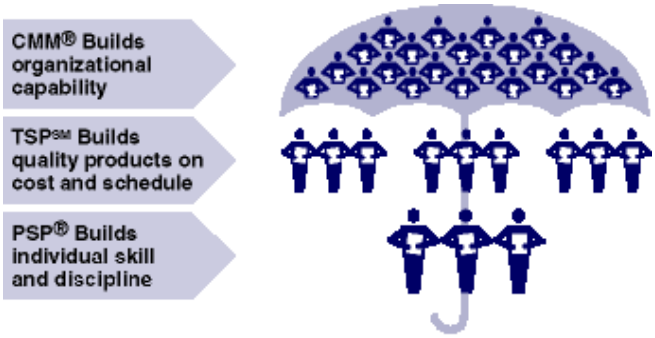
3. Phase process patterns. This type of process pattern depicts the interactions between the stage process patterns for a single project phase, such as the Initiate and Delivery phases.

- Source : Process Patterns; Scott W. Ambler Cambridge ,University Press



- Source : SPICE, Software Quality Institute

The TSP (Team Software Process) is a defined process that helps small teams of 3 to 15 engineers to run successful projects. Watts Humphrey, who founded CMM, made PSP (Personal Software Process) and TSP in order to help software engineers to do discipline work consistently.



- Source : TSP tool Dev.MSE Studio, Carnegie Mellon University

Process technology tools have been developed to help software organizations analyze their current process, organize work tasks, control and monitor progress, and manage technical quality. Process technology tools allow a software organization to build an automated model of the common process framework, task sets, and umbrella activities. The model, normally represented as a network, can then be analyzed to determine typical work flow and examine alternative process structures that might lead to reduced development time or cost. Once an acceptable process has been created, other process technology tools can be used to allocate, monitor, and even control all software engineering tasks defined as part of the process model. The process technology tool can also be used to coordinate the use of other computer-aided software engineering tools that are appropriate for a particular work task.

Product and Process – (depend on each other & express duality in nature) – If the process is weak, the end product will undoubtedly suffer, but an obsessive over-reliance on process is also dangerous. People derive as much (or more) satisfaction from the creative process as they do from the end product. An artist enjoys the brush strokes as much the framed result. A creative software professional should also derive as much satisfaction from the process as the end-product. The duality of product and process is one important element in keeping creative people engaged as the transition from programming to software engineering is finalized.

- *Source : Software engg. – a practitioners approach – roger.s*

Systems Development Life Cycle (SDLC)

Life-Cycle Phases

Initiation

Begins when a sponsor identifies a need or an opportunity. Concept Proposal is created

System Concept Development

Defines the scope or boundary of the concepts. Includes Systems Boundary Document. Cost Benefit Analysis. Risk Management Plan and Feasibility Study.

Planning

Develops a Project Management Plan and other planning documents. Provides the basis for acquiring the resources needed to achieve a soulution.

Requirements Analysis

Analyses user needs and develops user requirements. Create a detailed Functional Requirements Document.

Design

Transforms detailed requirements into complete, detailed Systems Design Document Focuses on how to deliver the required functionality

Development

Converts a design into a complete information system Includes acquiring and installing systems environment; creating and testing databases preparing test case procedures; preparing test files, coding, compiling, refining programs; performing test readiness review and procurement activities.

Integration and Test

Demonstrates that developed system conforms to requirements as specified in the Functional Requirements Document. Conducted by Quality Assurance staff and users. Produces Test Analysis Reports.

Implementation

Includes implementation preparation, implementation of the system into a production environment, and resolution of problems identified in the Integration and Test Phases

Operations & Maintenance

Describes tasks to operate and maintain information systems in a production environment. includes Post-Implementation and In-Process Reviews.

Disposition

Describes end-of-system activities, emphasis is given to proper preparation of data.

The tenth phase occurs when the system is disposed of and the task performed is either eliminated or transferred to other systems.

SDLC (- Sys. DLC actually called as System Development Life cycle)

Prescriptive **Process models**

Prescriptive process models advocate an orderly approach to software engineering

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 (traditional waterfall model, waterfall model with feedback, v model)

incremental (RAD), or

evolutionary (Prototyping, Spiral, *Concurrent Dev. Model*),

specialized (Component based, formal method model, AOSD)

Software Development Lifecycle (S/w DLC) Models

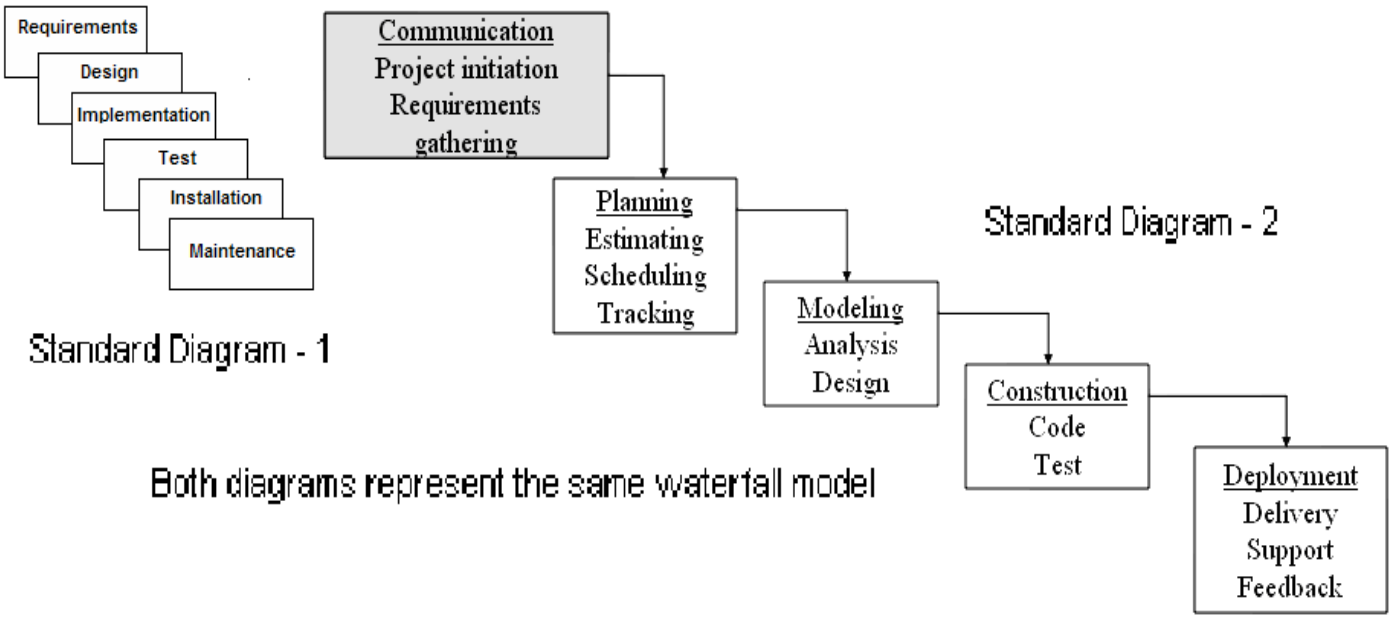
A lifecycle model, also called as process model, is an abstract description of the software development and modification process.

- It maps out the main stages in which the above process is carried out.
- Software Engineering provides certain process models for carrying out the software development activity.
- There are various and numerous methods to guide the life cycle of a software project. We can nonetheless define a few canonical models from which the wide variety of methods is local adaptations.

Activities in the Development Process Lifecycle

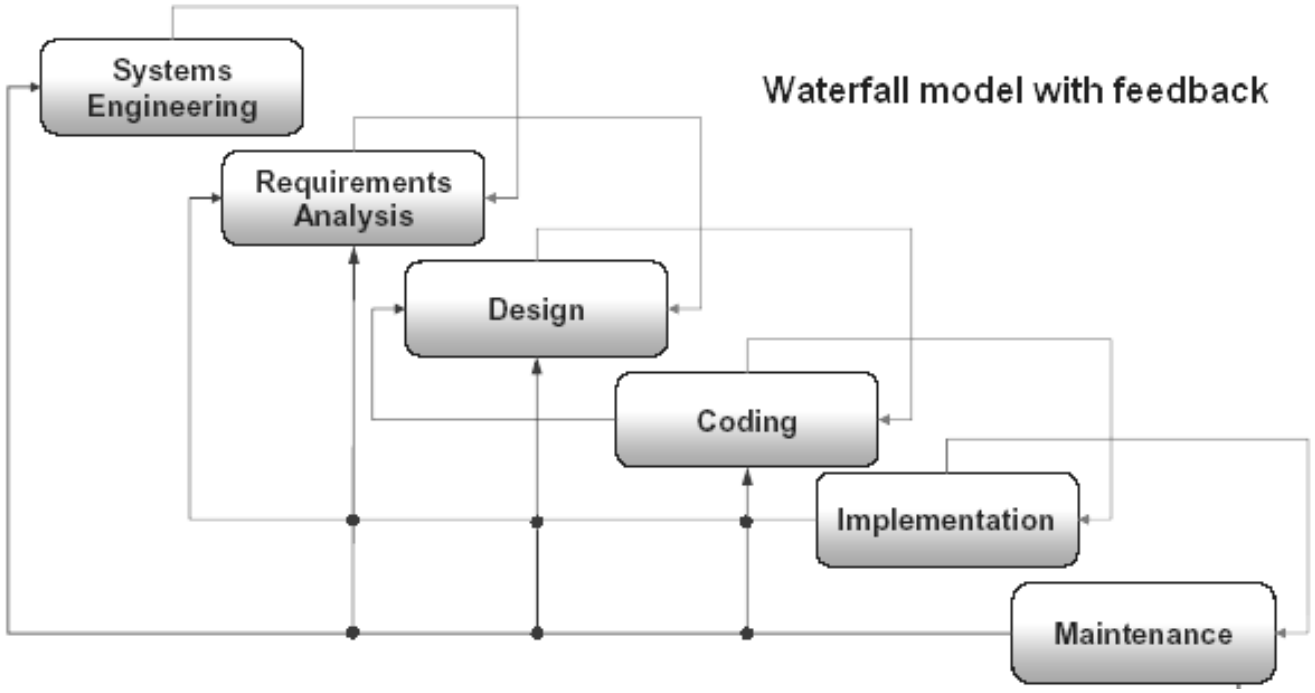
- Conceptualization & Requirements
- Planning (Estimating ,Scheduling, Tracking)
- Modeling / Design of Systems (Analysis & Specification, Architectural & Infrastructure Design, Detailed Design, etc)
- Construction / Module Development
  - Coding
  - Testing
    - Unit Testing
    - Integration Testing
    - System Testing
    - User Acceptance Testing
- Deployment
  - Installation
  - User and Technical Training
  - Maintenance

Classical / Traditional Waterfall model



Called as the Classic Lifecycle Model developed by Royce in 1970.

- Follows a documentation driven paradigm.
- Is a **time-ordered sequence of activities** called as lifecycle stages.
- Requirements of customer are well understood
- Work flows from Communication activity to deployment in Linear Fashion/ a systematic and sequential approach to software
- Generally adapted when well-defined requirements which are reasonably stable or when enhancements to the existing system must be made.
- Each framework activity starts only after the previous activity has finished.
- Iteration is not possible



- System Engineering: The software to be developed must always function as part of a larger system. Hence it is necessary to start out by establishing requirements for the larger system. A subset of these
is then assigned to the software. This stage ensures that the software will interface properly with people, hardware and other software in the system.

 Software Requirements Analysis: The requirements gathering activity now focuses specifically on the software. Here the software engineer, playing the role of analyst, understands the information domain, as well as the functional, performance and interfacing requirements. The output of this stage is the structured requirements.

 Design: After the structured requirements are documented, the actual structure of the programs that will make up the software is created. The structure may be thought of as being determined by four distinct attributes: software architecture, data structure, procedural detail and interface characterization. The design process thus translates the structured requirements into an actual representation of the software. The output of this stage is the design documents, including the program specifications.

 Coding (Build): The program specifications are translated into a machine-readable form by the process of coding. The output of this stage is the actual programs that will make up the software.

 Testing: The programs created are tested for conformance to specifications. The internal logic, as well as functional and performance aspects are tested.

 Maintenance: After software system is installed and running there are many reasons that demand a change to this system. Some of the most common reasons being, in the long run changing business needs, changing technological advances or needs, etc. The immediate being the errors made during development or bugs. Maintenance, therefore, means making changes to the existing system. Maintenance essentially consists of reapplying each of the above stages to existing software rather than new software.

Advantages

 Simplicity and
 The logical way of structuring the different activities in a software project.
 The Waterfall model is well suited to projects that has low risk in the areas of user interface and performance requirements, but high risk in budget, schedule predictability and control.

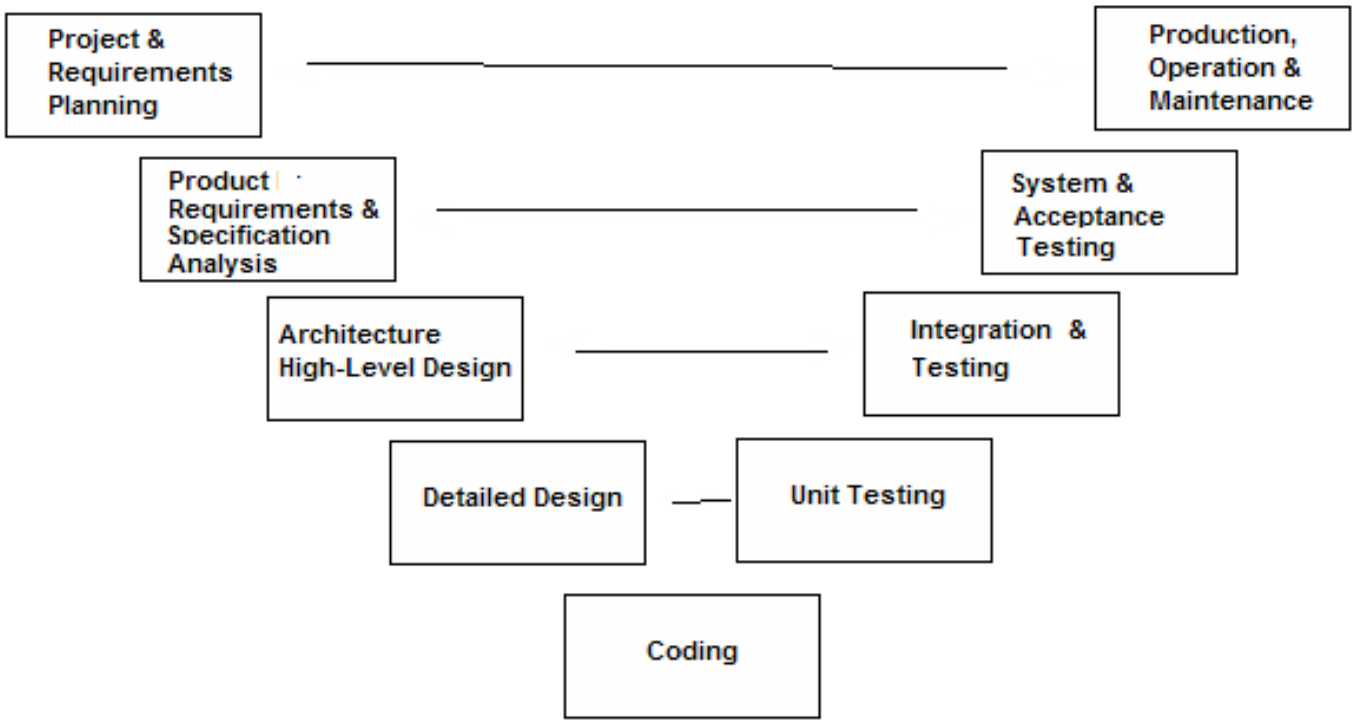
Disadvantages

 It assumes that the requirements are completely ready before the design, which is not the case in most of the development scenarios
 Does not admit iteration between stages.
 A working version of the software is not available until late in the development process and the customer is completely cut-off from the development until the acceptance testing stage.
 The major drawback of the waterfall model is that it assumes that the requirements completely evolved, elicited and specified in advance during the Requirements Analysis stage. Unfortunately, requirements keep on evolving and changing throughout the process and beyond, therefore requiring for considerable and continuous feedback and iterative consultation.

When to use the Waterfall Model

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.

V shaped SDLC model



- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

- Project and Requirements Planning – allocate resources
- Product Requirements and Specification Analysis – complete specification of the software system
- Architecture or High-Level Design – defines how software functions fulfill the design
- Detailed Design – develop algorithms for each architectural component
- Production, operation and maintenance – provide for enhancement and corrections
- System and acceptance testing – check the entire software system in its environment
- Integration and Testing – check that modules interconnect correctly
- Unit testing – check that each module acts as expected
- Coding – transform algorithms into software

V-Shaped Strengths

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

V-Shaped Weaknesses

- Does not easily handle concurrent events
- Does not handle iterations or phases

- Does not easily handle dynamic changes in requirements
- Does not contain risk analysis activities

When to use the V-Shaped Model

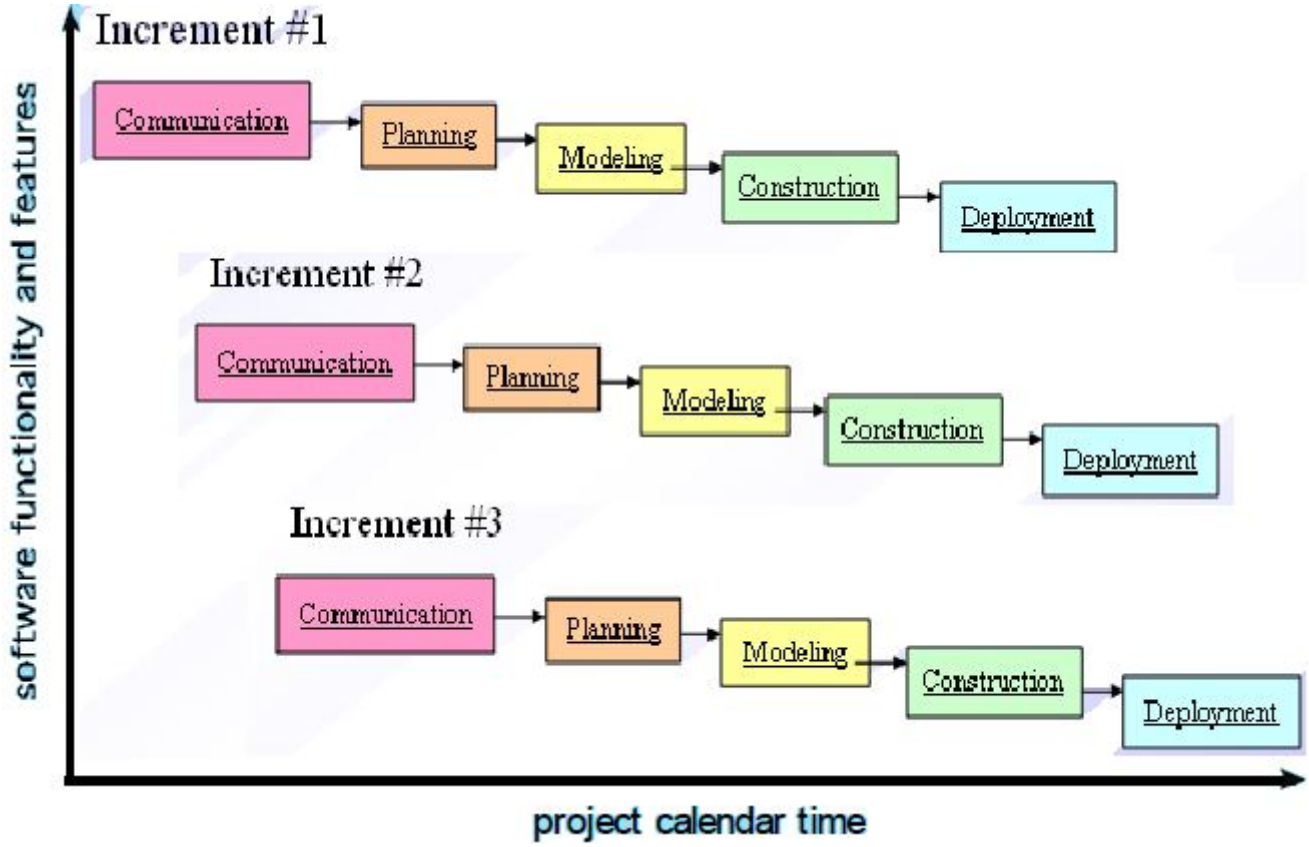
- Excellent choice for systems requiring high reliability – hospital patient control applications
- All requirements are known up-front
- When it can be modified to handle changing requirements beyond analysis phase
- Solution and technology are known

**** Iterative Process ****

 System requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems.
 Iteration can be applied to any of the generic process models.

 There may be situations where Software requirements are not well-defined, but the overall scope of the development follows a purely linear process.

The Incremental Model



Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
User requirements are prioritised and the highest priority requirements are included in early increments.
Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve

- Construct a partial implementation of a total system
- Then slowly add increased functionality
- The incremental model prioritizes requirements of the system and then implements them in groups.

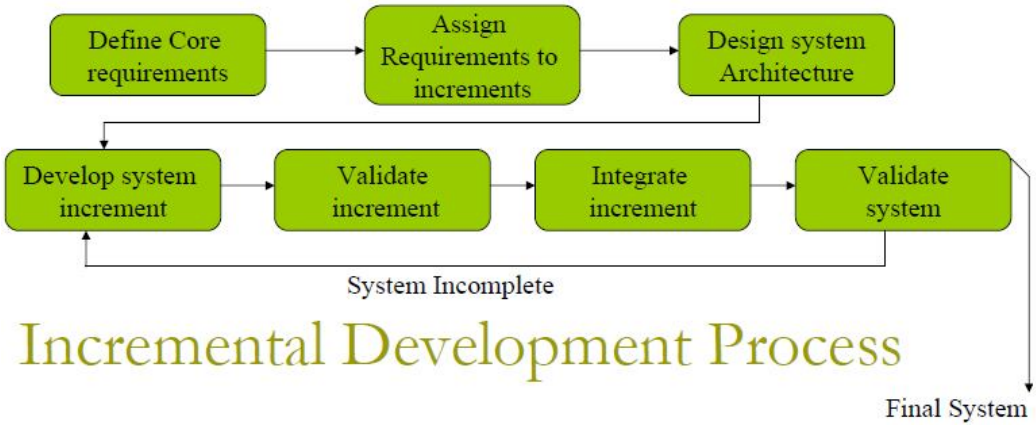
- Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.

Initial software requirements are reasonably well-defined.
Combines elements of Waterfall model applied in an iterative fashion. i,e, Applies linear sequence in a ”staggered fashion”.
It is Iterative in nature
Focuses on the delivery of operational product with each increment.
Early increments are “stripped down” versions of the final product.
Eg: Word doc/ ppt
The first increment is always the “CORE PRODUCT” i,e, basic requirements
The core product is evaluated / used by customer .
Based on the feedback , a plan is developed for the next increment.
Plan addresses the modification to the core product & delivery of additional features and functionality.
The process is repeated following the delivery of each increment, until the complete product is released.

When to use Incremental Model?

Initial software requirements are reasonably well-defined.
When there is a compelling need to provide a limited set of software functionality to users quickly and then refine and expand on that functionality in later releases.
When staffing is unavailable for a complete implementation by business deadline. Early increments can be implemented by few people & If core product is well received, additional staff can be added to implement the next increment.

- Risk, funding, schedule, program complexity, or need for early realization of benefits.
- Most of the requirements are known up-front but are expected to evolve over time
- A need to get basic functionality to the market early
- On projects which have lengthy development schedules
- On a project with new technology



Incremental Development Process

Incremental Model Strengths

- Develop high-risk or major functions first
- Each release delivers an operational product
- Customer can respond to each build
- Uses “divide and conquer” breakdown of tasks
- Lowers initial delivery cost

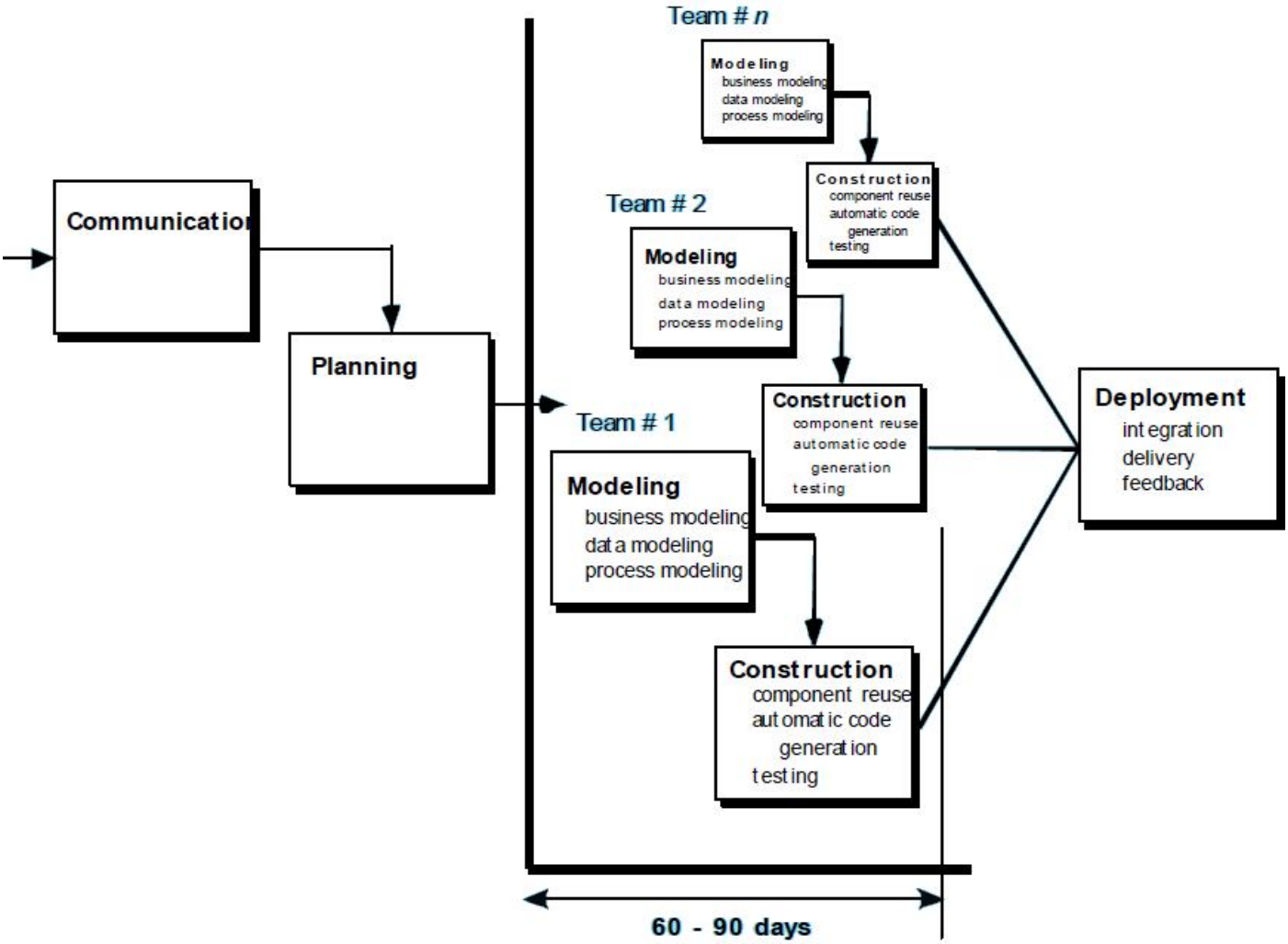
- Initial product delivery is faster
- Customers get important functionality early
- Risk of changing requirements is reduced

Incremental Model Weaknesses

- Requires good planning and design
- Requires early definition of a complete and fully functional system to allow for the definition of increments
- Well-defined module interfaces are required (some will be developed long before others)
- Total cost of the complete system is not lower

Advantages
The customer do not have to wait until the entire system is delivered.
Customers can use the early increments as a form of prototype and gain experience which informs the requirements for later system development.
Lower risk of overall project failure.
The highest priority services are delivered first and the later increments are integrated with them.

RAD (Rapid Application Development Model)



Focuses on a SHORT DEVELOPMENT CYCLE.
Requirements are well-understood and project scope is constrained.

High-speed adaptation of the waterfall model , in which Rapid development is achieved by using a component based construction approach where each major function is completed in 60 to 90 days.
Maps into generic framework activities.
Communication : works to clearly understand the business problem.
Planning: imp. As multiple teams work in parallel on different system functions.
Modeling: encompasses Business modeling, data modeling and process modeling and establishes design representations.
Construction
Deployment

- Requirements planning phase (a workshop utilizing structured discussion of business problems)
- User description phase – automated tools capture information from users
- Construction phase – productivity tools, such as code generators, screen generators, etc. inside a time-box. ("Do until done")
- Cutover phase -- installation of the system, user acceptance testing and user training

RAD Strengths

- Reduced cycle time and improved productivity with fewer people means lower costs
- Time-box approach mitigates cost and schedule risk
- Customer involved throughout the complete cycle minimizes risk of not achieving customer satisfaction and business needs
- Focus moves from documentation to code (WYSIWYG).
- Uses modeling concepts to capture information about business, data, and processes.

RAD Weaknesses

- Accelerated development process must give quick responses to the user
- Risk of never achieving closure
- Hard to use with legacy systems
- Requires a system that can be modularized
- Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.

When to use RAD

- Reasonably well-known requirements
- User involved throughout the life cycle
- Project can be time-boxed
- Functionality delivered in increments
- High performance not required
- Low technical risks
- System can be modularized

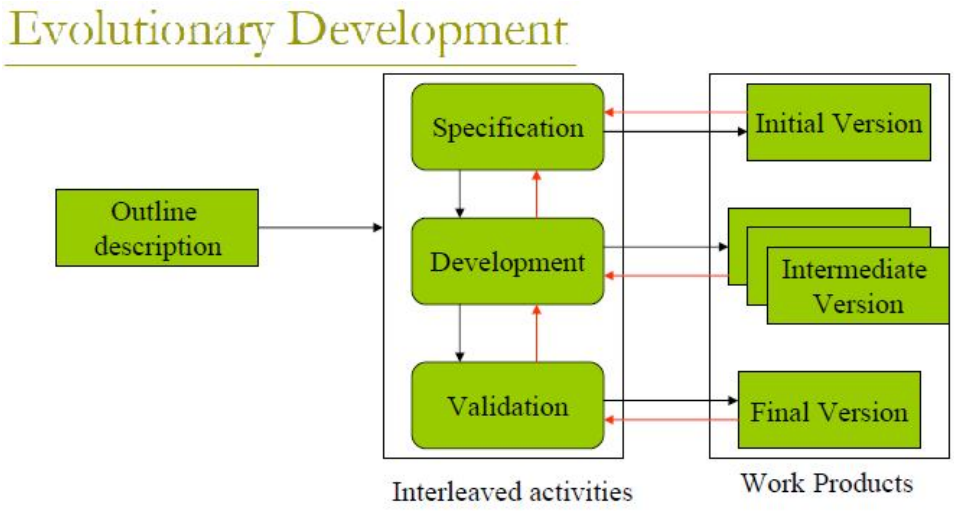
Drawbacks:
 For large, but scalable projects, RAD requires sufficient HR to create the right number of RAD teams that will be working in parallel.

 If developers and customers are not committed to Rapid-Fire- Activities necessary to complete the system, RAD projects fail.
 If the system cannot be properly modularized, building the components necessary for RAD will be problematic.
 If high performance is an issue, and performance is to be achieved through tuning the interfaces to the subsystem components, then RAD approach may not work.
 RAD may not be appropriate when technical risks are high

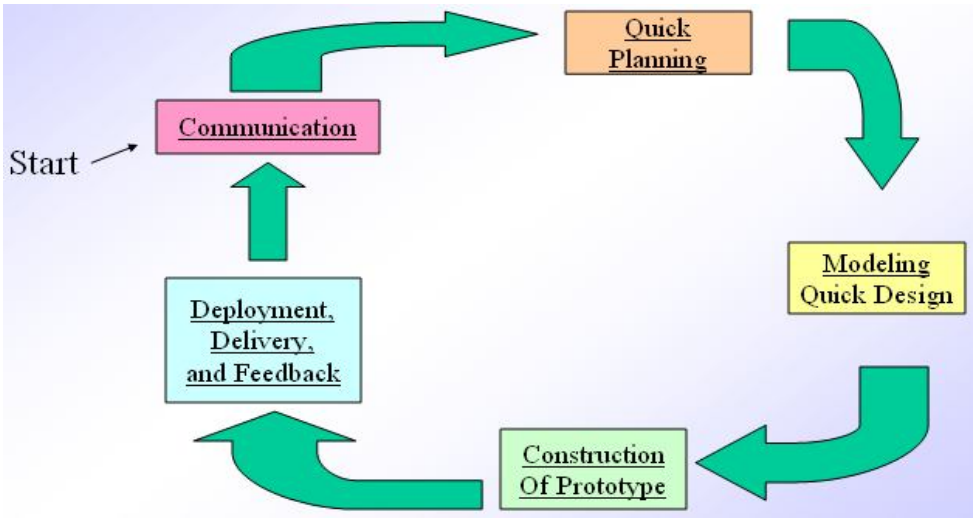
Evolutionary model

Changes or evolution in technology from the past made the s/w enggr realize that there should be techniques to cover such aspects which gave rise to ev. Models

Evolutionary Development Models
 Software like all complex systems , evolves over a period of time.
 Business and product requirement often change as development proceeds, making a straight line path to an end product unrealistic.
 The set of core product / system requirements is well understood, but the details of the product/ system extensions have yet to be defined.
 Evolutionary models are iterative.
 They are characterized in a manner that software engineers to develop increasingly more complete versions of the software



Structured Evolutionary **Prototyping**

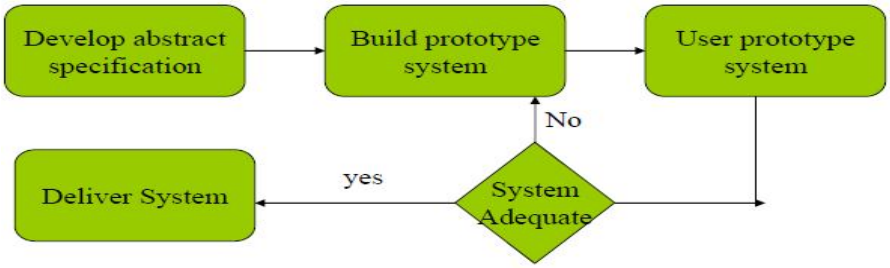


- Follows an evolutionary and iterative approach
- Used when requirements are not well understood
- Serves as a mechanism for identifying software requirements
- Focuses on those aspects of the software that are visible to the customer/user
- Feedback is used to refine the prototype

In what situation is this model useful?
Requirements are very seldom fully known at the beginning of a project, the Prototyping model therefore addresses this fact.
 How is it used?
It starts with first building a simplified version of the system, then seek feedback from the different stakeholders of the project, then come up with a second, better system. This cycle is repeated until all the stakeholders are completely satisfied with all the aspects of the system.
 Variations on the theme of prototyping can be exploited further to inscrease the success rate of the project. For instance, a development team can consider approaches like:
 Creating just a user interface without background data processing or validation logic.
 Coding only one or a few important functionality of the system or subsystems.

Prototyping Types

 Throwaway Prototyping Model
Useful in "proof of concept" or situations where requirements and user's needs are unclear or poorly specified. The approach is to construct a QUICK AND DIRTY partial implementation of the system during or before the requirements phase.
 Exploratory Prototyping Model
Useful in projects that have low risk in such areas as losing budget, schedule predictability and control, large-system integration problems, or coping with information sclerosis, but high risk in user interface design.



Prototyping model consists of the following steps:
 Requirements: Elicitation of requirements at the time.
 Design: Integrate the initial layer of requirements into a new or existing design to form a prototype.
 Prototype Creation or Modification: Design is used to create a prototype of the system. This may mean creating a prototype or modifying the already created one (in the earlier cycle).
 Assessment: The prototype is presented to the customer for review and assessment. Comments and suggestions are collected from the customer and recorded.
 Prototype Refinement: Information collected from the customer is use to refine the prototype.
 Systems Implementation: In most cases, the system is rewritten once requirements are understood.

Disadvantages -
 Prototyping can lead to false expectations. This is a situation created by the customer thinking the system is ready to roll out. The customer does not see the work that has to be done internally such as database normalization and the like.
 In addition, this model may lead to Poor System Design, especially when we have to "grow“ the software. At some point, in may become obvious that our choice of data structures for the early iterations were not general enough, and needs to be changed. Such effects are not as rare as we would like them to be.
 More importantly this model makes it more difficult to schedule the actual implementation. Situations where Market Timing is important, the risk of using this model is very high.
 The process is not visible for measuring progress.
 Systems are often poorly structured, due to continual changes.
 Special tools and techniques may be required.

Structured Evolutionary Prototyping Strengths

- Customers can “see” the system requirements as they are being gathered
- Developers learn from customers
- A more accurate end product
- Unexpected requirements accommodated
- Allows for flexible design and development
- Steady, visible signs of progress produced
- Interaction with the prototype stimulates awareness of additional needed functionality

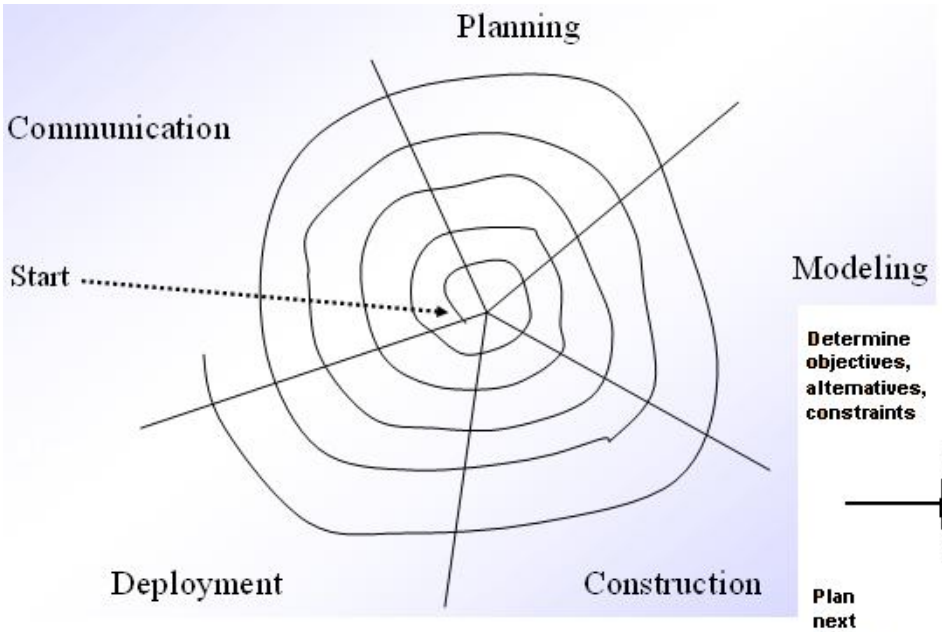
Structured Evolutionary Prototyping Weaknesses

- Tendency to abandon structured program development for “code-and-fix” development
- Bad reputation for “quick-and-dirty” methods
- Overall maintainability may be overlooked
- The customer may want the prototype delivered.
- Process may continue forever (scope creep)

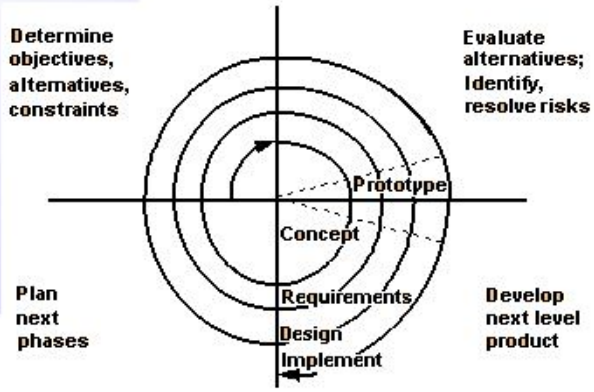
When to use Structured Evolutionary Prototyping

- Requirements are unstable or have to be clarified
- As the requirements clarification stage of a waterfall model
- Develop user interfaces
- Short-lived demonstrations
- New, original development
- With the analysis and design portions of object-oriented development.

Spiral model



both diagrams are same but different books



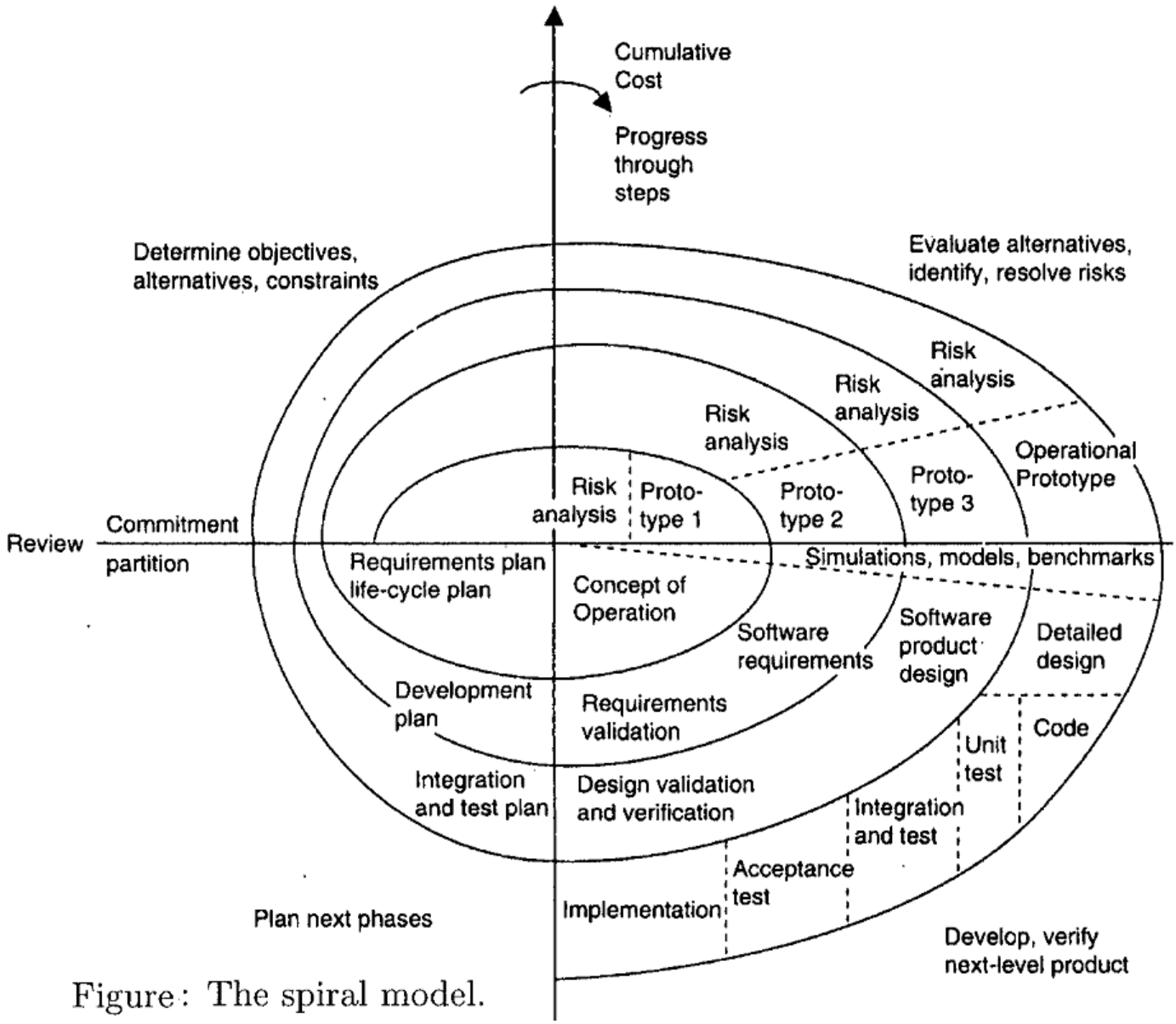


Figure: The spiral model.

- Invented by Dr. Barry Boehm in 1988 while working at TRW
- Follows an evolutionary approach
- Used when requirements are <u>not</u> 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

 Most life cycle models can be derived as special cases of the spiral model.
 It is an evolutionary model that couples with the iterative nature of prototyping, with controlled and systematic aspects of Waterfall model.
 The spiral uses a risk management approach to software development.
 It has two main distinguishing features:
 Cyclic approach for incrementally growing a system's degree of definition & implementation while decreasing the degree of risk.

 A set of anchor point milestones for ensuring feasible solution

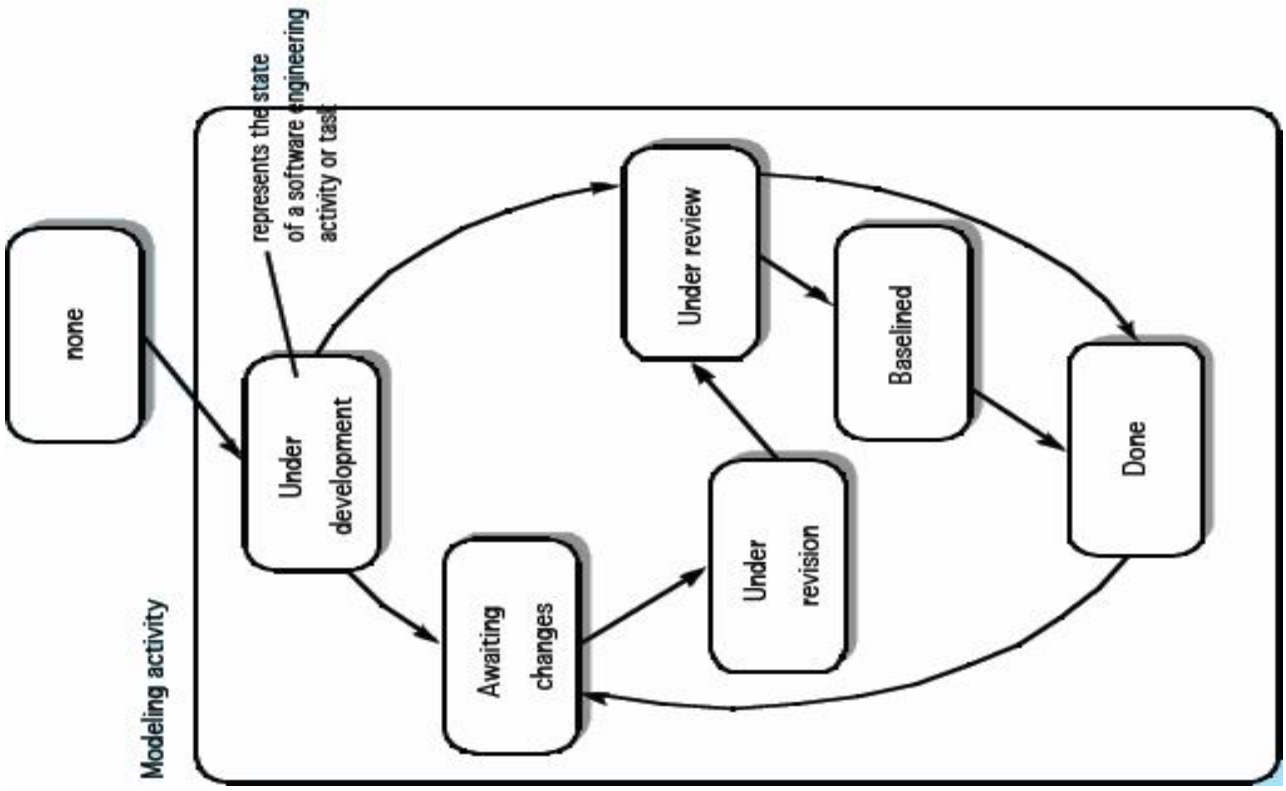
The spiral model emphasizes on the need to go back and re look at earlier stages, i.e. requirements and design, as many number of times as required till the projects stake holders are completely satisfied with all the aspects of the system.
 The spiral model is therefore iterative and can be considered as a series of short waterfall cycles, each cycle ending by producing an early prototype representing a part of the entire project.
 The advantage of this approach is that it helps in demonstrating a proof of concept early in the project life cycle, and therefore more accurately reflects the disorderly or even chaotic evolution of technology

The radial dimension represents evolution towards a complete system.
 In each iteration analyze the results of the previous iteration, determine the risk and build a prototype. With each iteration outward, progressively more complete versions of the software are built.
 During the first circuit, the objectives, alternatives and constraints are defined. Risks are identified and analyzed. If risk analysis indicates substantial uncertainty in requirements, prototyping may be done.
 The Engineering may follow the life cycle approach or a prototyping approach. The customer evaluates the work and makes suggestions for modifications. Based on these, the next stage of planning and risk analysis is done. The system thus evolves toward a complete version.

Advantages
 Defers elaboration of low risk software elements
 Incorporates prototyping as a risk reduction strategy
 Gives an early focus to reusable software
 Accommodates life-cycle evolution, growth, and requirement changes
 Incorporates software quality objectives into the product
 Focus on early error detection and design flaws
 Sets completion criteria for each project activity to answer the question: "How much is enough?"
 Uses identical approaches for development and maintenance
 Can be used for hardware-software system development

Concurrent Dev. Model

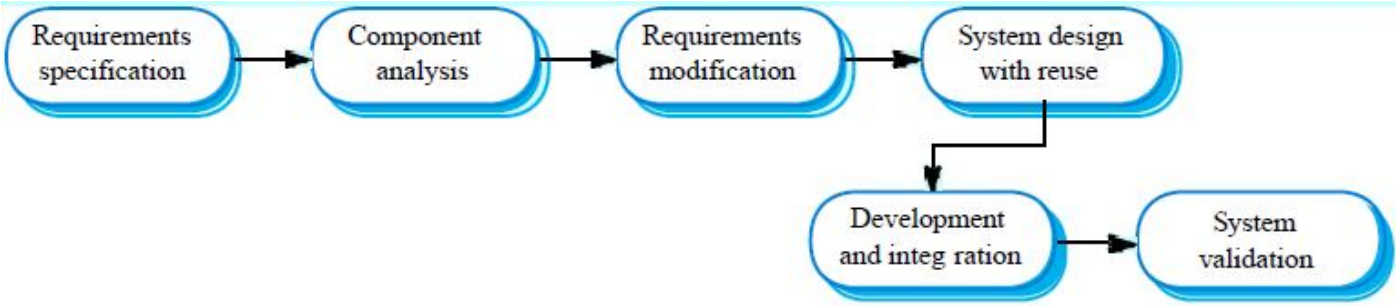
Not at all a model - Developed concurrent to know the present state of the s/w project



Component based Development model

- the process to apply when reuse is a development objective
 Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
 Process stages
 Component analysis;
 Requirements modification;
 System design with reuse;
 Development and integration.
 This approach is becoming increasingly used as component standards have emerged.

- 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 model composes applications from prepackaged or off-the shelf software components with well defined interfaces that enable the component to be integrated into the software.
 CBD incorporates many of the characteristics of the Spiral model and is evolutionary in nature, demanding an iterative approach.
 Modeling and Construction begins activities begin with the identification of Candidate components.
 Regardless of the technology that is used to build the component,

CBD incorporates the following 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

The Formal Methods Model

emphasizes the mathematical specification of requirements

- 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

AOSD

—provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*

Software development is changing. The opportunities of the Internet, computerized businesses, and computer-savvy consumers, the exponential decline in the cost of computation and communication, and the increasingly dynamic environment for longer-living systems are pressing software developers to come up with better ways to create and evolve systems. There is fomenting in software development process, system structure, programming, quality assurance, and maintenance.

Software is about building computational models of some part of the elements or information flow of the world. For all but the most trivial software systems, conquering the engineering of the system requires (perhaps recursively) dividing the system into chunks that can be (by and large) separately created and managed. The last decade of the twentieth century saw the rise (and perhaps dominance) of the objectoriented perspective on system modularization. Object-orientation focuses on selecting objects as the primary unit of modularity and associating with these objects all the system's behavior. Objects are typically elements of the domain or of the computational process.

Object-orientation is reaching its limits. Many things one cares about in creating a software system (concerns) are not neatly localized to the behavior of specific "things." Building diverse systems requires simultaneously manipulating many concerns. Examples of concerns range from non-functional notions such as security, reliability, and manageability to precise implementation techniques such as concurrency control, caching, and error recovery. Since conventional programming technologies are centered on producing a direct sequence of instructions, they require the programmer to remain cognizant of all such concerns throughout the programming process. The programmer must explicitly intermix the commands to achieve these concerns with the code for the primary application functionality. This produces tangled code and erroneous and difficult-to-maintain systems.

New technologies are emerging to allow richer specifications of programs and better modularization of these specifications. Along with these new technologies, we are also seeing novel software engineering methodologies for using them. One of the most exciting of these new technologies is aspect-oriented software development (AOSD). AOSD programming technologies (aspect-oriented programming, or AOP) provide linguistic mechanisms for separate expression of concerns, along with implementation technologies for weaving these separate concerns into working systems. Aspect-oriented software engineering (AOSE) technologies are emerging for managing the process of developing systems within this new paradigm.

THE UNIFIED PROCESS *(Developed by James Rumbaugh, Grady Booch, and Ivar Jacobson)*

—a “use-case driven, architecturecentric, iterative and incremental” software process closely aligned with the Draws the best FEATURES and CHARACTERSTICS of conventional Software process models.

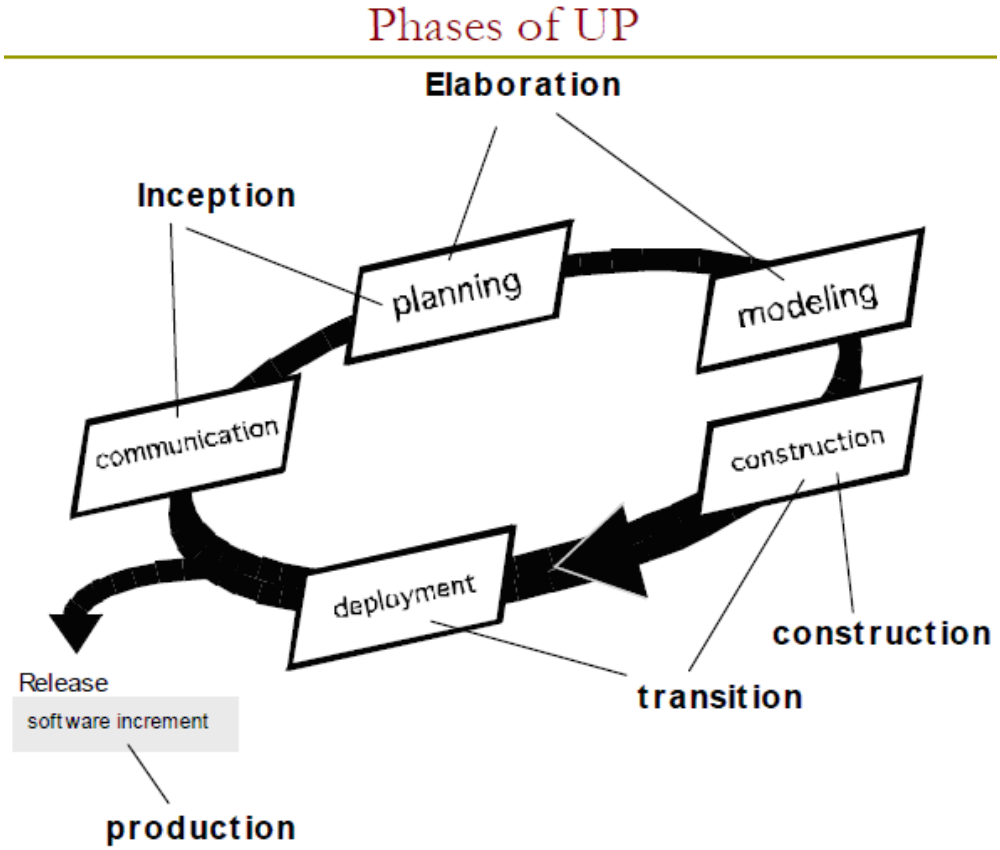
• Focuses on the importance of Customer communication & streamlined methods for describing the customer’s view of the system.

• Emphasizes the important role of software architecture and helps architect focus on the right goals, like understandability, reliance to future changes, and reuse.

• The unified process is incremental, and iterative, providing the evolutionary feel.

• Provides necessary technology to support Object-Oriented-SE practice.

•Hybrid Process model.



The INCEPTION Phase:

Encompasses both Customer Communication & Planning Activities.

Collaborates with customers, end-users, and all stake holders for identifying the fundamental Business requirements in the for of USE-CASES.

A rough Architecture for the system is proposed.

Initially, a tentative outline of major subsystems and the function and features that populate them is considered.

A Plan for Iterative, incremental nature of ensuing the project is developed.

Planning identifies Resources, Assesses major risks, defines a Schedule, decomposes the project, and establishes basis for Incremental deliverables.

. Use-Cases describes the features and functionality desired by major class of users/ sequence of actions that are performed by user as they interact with the software. Eg: ATM

Use-Case helps to identify the scope of the project and provide a basis for Project Planning.

The ELABORATION Phase:

Encompasses Customer communication and Modeling activities of Generic frame work activities.

Refines and expands preliminary Use-Cases that were developed as part of Inception phase

Expands Architectural representation to include 5 different views of software:

The Use-Case model :Use-case diagramUsers and Functionality

The Analysis model :Sequence /Activity diagram/state chart

The Design model :Class Diagram

The Implementation model :Component diagram

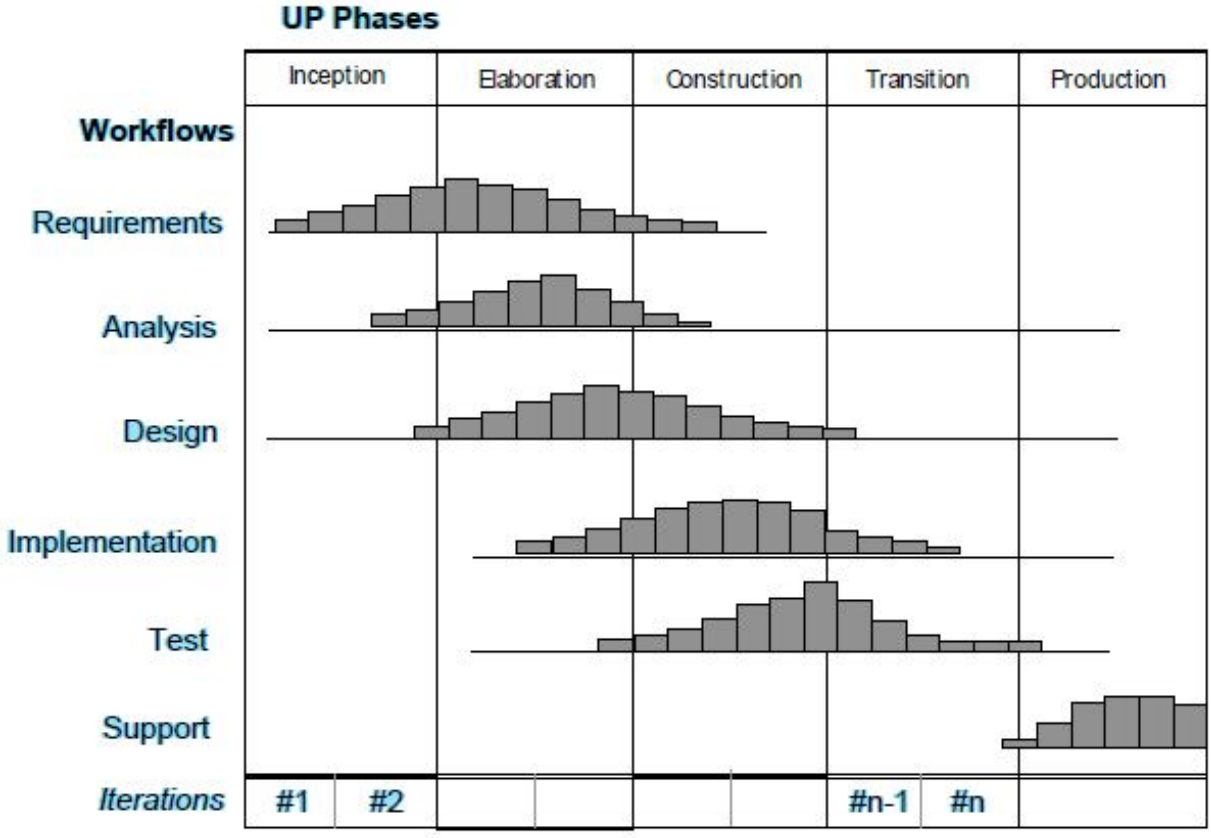
The Deployment model. :Deployment diagram

A close review is done on the Plan and any modification to the plan is made.

The CONSTRUCTION Phase:
Identical to the construction activity of generic process.
Using Architectural model as input, the construction phase develops / acquires software components that will make each use-case operational for end-users.
The Analysis and Design models started during Elaboration phase are completed to reflect the final version of s/w increment.
All reqd. functionalities and features are implemented.
Unit tests are designed & executed for each use-case.
Integration activities are conducted.

The TRANSITION Phase:
Encompasses the latter stages of generic construction activity & first part of generic Deployment activity..
Developed software is given to end-users for beta-testing.
User feedback reports DEFECTS & NECESSARY changes.
Necessary work products/ support information are created.
The output of Transition phase is USABLE SOFTWARE RELEASE.

The PRODUCTION phase:
Identical to deployment phase of generic activity.
The software is monitored.
Support for operating environment is provided
Defect reports & requests for changes are submitted and evaluated.

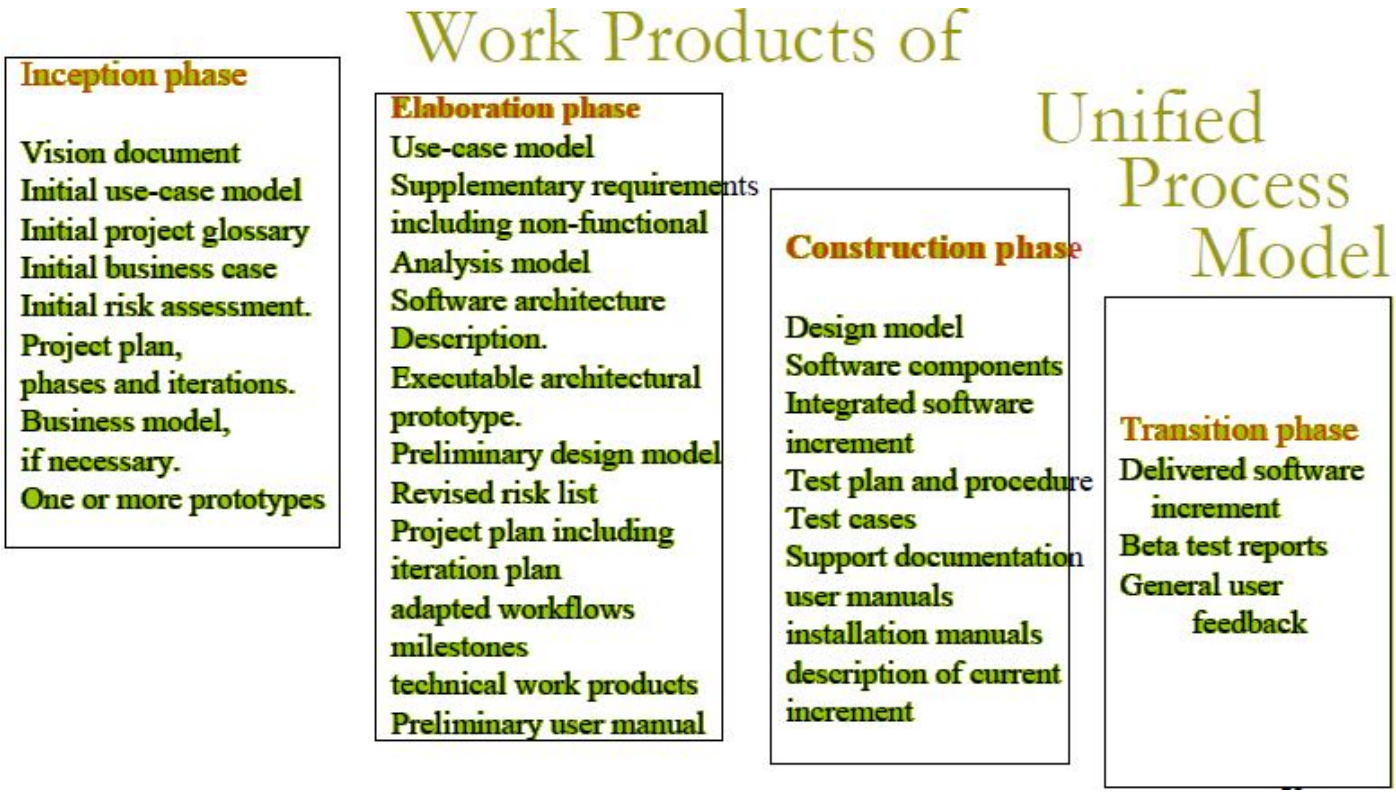


UP phases vs workflows chart

Unified Process Work Products

- Work products are produced in each of the first four phases of the unified process
- In this course, we will concentrate on the analysis model and the design model work products
- Analysis model includes
  - Scenario-based model, class-based model, and behavioral model

- Design model includes
  - Component-level design, interface design, architectural design, and data/class design



AGILITY

It is the ability to change the systems state efficiently

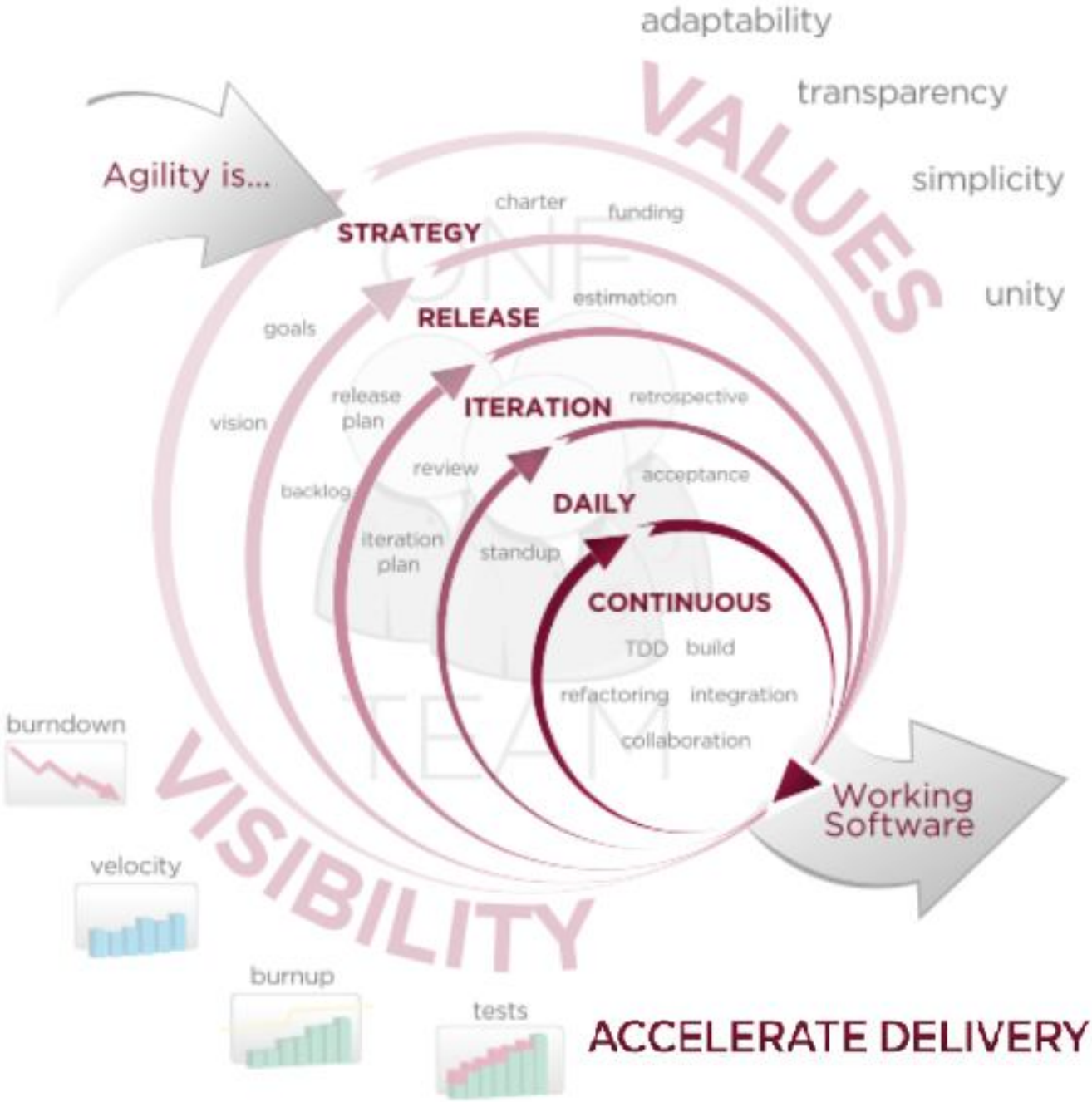
It is the ability of maintaining equilibrium of the system after sudden changes

AGILE PROCESS / Agile SDLC's

- Speed up or bypass one or more life cycle phases
- Usually less formal and reduced scope
- Used for time-critical applications
- Used in organizations that employ disciplined methods

Agile software development

It is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle.



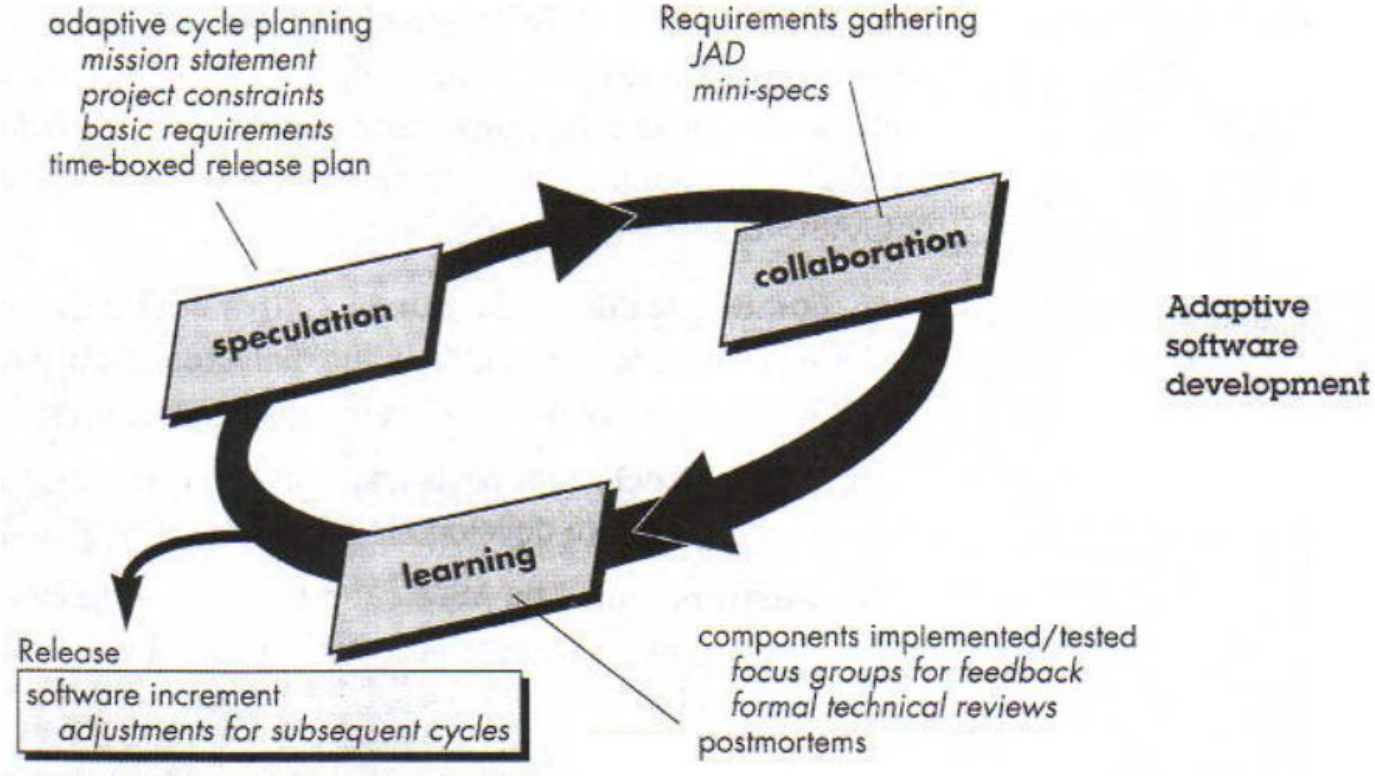
AGILE PROCESS MODELS

- Adaptive Software Development (ASD)
- Feature Driven Development (FDD)
- Crystal Clear
- Dynamic Software Development Method (DSDM)
- *Rapid Application Development (RAD)* *
- Scrum
- Extreme Programming (XP)
- Rational Unify Process (RUP)
- Agile Modeling (AM)

Adaptive SDLC

Combines RAD with software engineering best practices

- Project initiation
- Adaptive cycle planning
- Concurrent component engineering
- Quality review
- Final QA and release



Adaptive Steps-

1. Project initialization – determine intent of project
2. Determine the project time-box (estimation duration of the project)
3. Determine the optimal number of cycles and the time-box for each
4. Write an objective statement for each cycle
5. Assign primary components to each cycle
6. Develop a project task list
7. Review the success of a cycle
8. Plan the next cycle

Feature Driven Design (FDD)

Five FDD process activities

1. Develop an overall model – Produce class and sequence diagrams from chief architect meeting with domain experts and developers.

2. Build a features list – Identify all the features that support requirements. The features are functionally decomposed into Business Activities steps within Subject Areas.

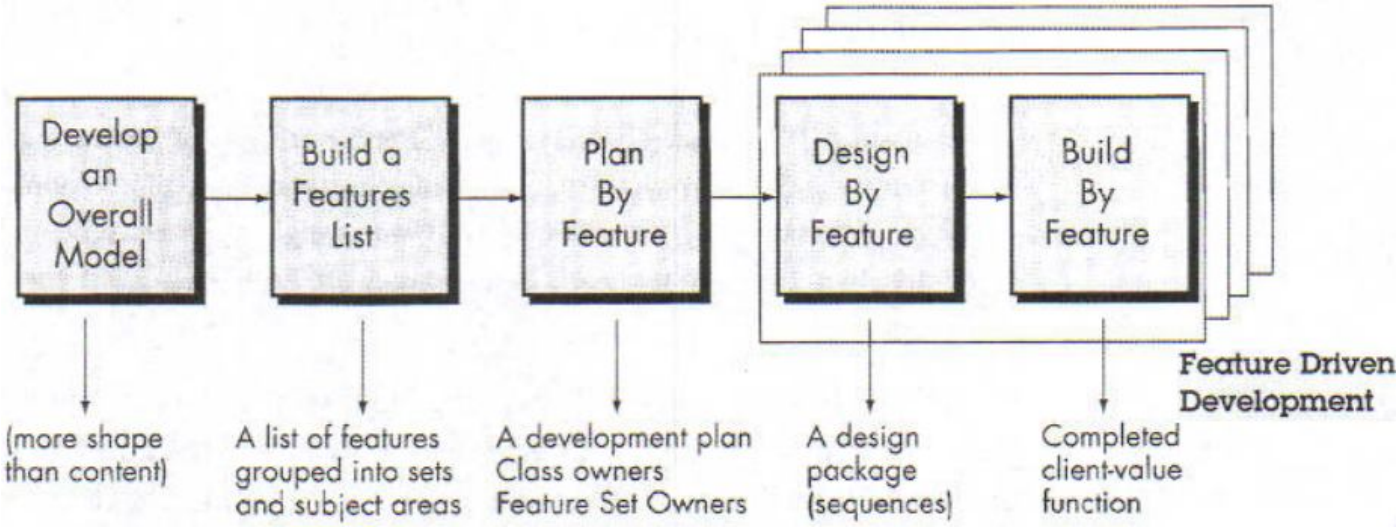
Features are functions that can be developed in two weeks and expressed in client terms with the template: <action> <result> <object>

i.e. Calculate the total of a sale

3. Plan by feature -- the development staff plans the development sequence of features

4. Design by feature -- the team produces sequence diagrams for the selected features

5. Build by feature – the team writes and tests the code



Crystal clear

Crystal Clear is a member of the Crystal family of methodologies as described by Alistair Cockburn and is considered an example of an agile or lightweight methodology.

Crystal Clear can be applied to teams of up to 6 or 8 co-located developers working on systems that are not life-critical. The Crystal family of methodologies focus on efficiency and habitability as components of project safety.[1] Crystal Clear focuses on people, not processes or artifacts.

Crystal Clear requires the following properties:

- Frequent delivery of usable code to users
- Reflective improvement
- Osmotic communication preferably by being co-located

Crystal Clear additionally includes these optional properties:

- Personal safety
- Focus
- Easy access to expert users
- Automated tests, configuration management, and frequent integration

Dynamic Systems Development Method (DSDM)

Applies a framework for RAD and short time frames

Paradigm is the 80/20 rule

– majority of the requirements can be delivered in a relatively short amount of time.

DSDM Principles-

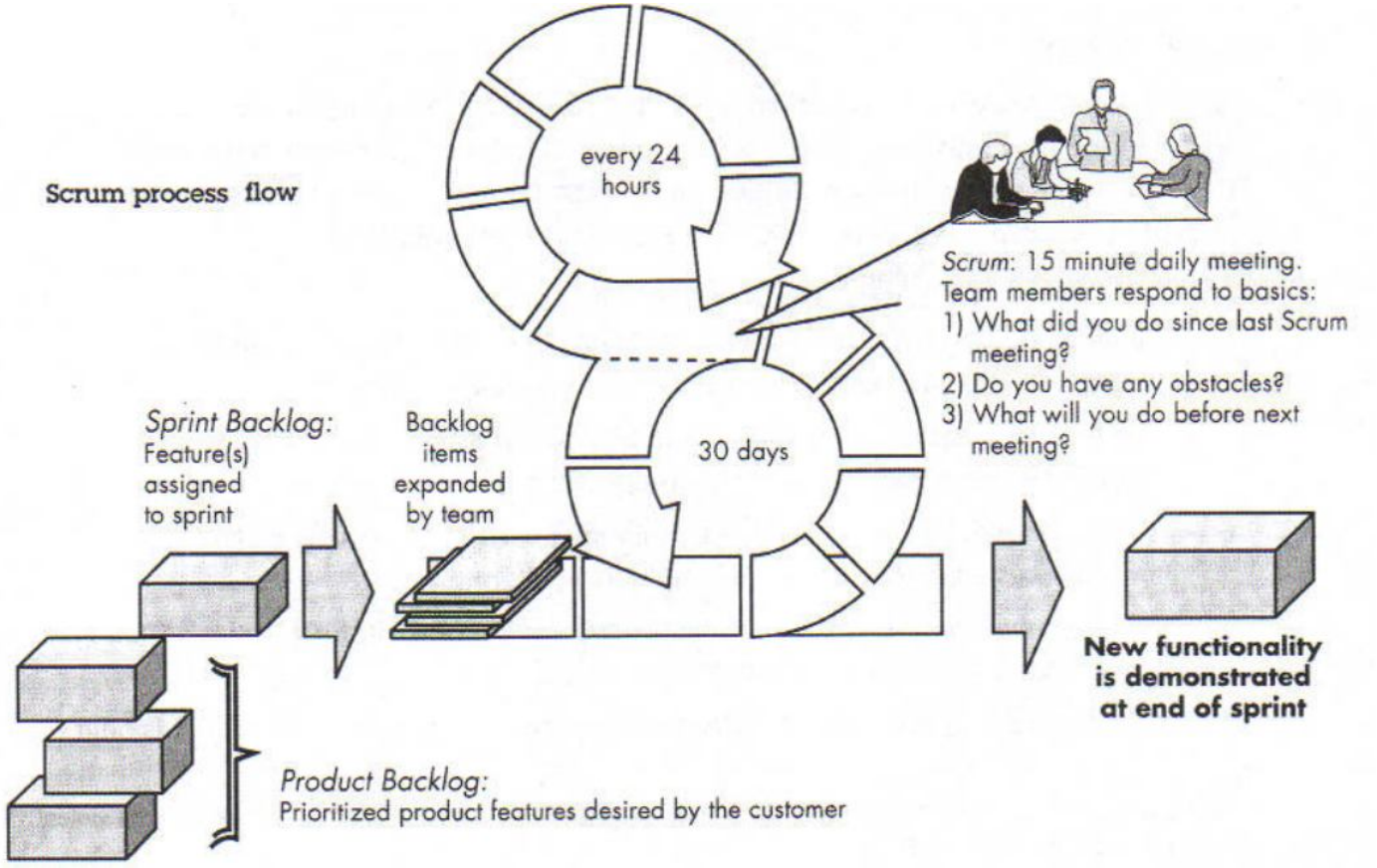
1. Active user involvement imperative (Ambassador users)
2. DSDM teams empowered to make decisions
3. Focus on frequent product delivery
4. Product acceptance is fitness for business purpose
5. Iterative and incremental development – to converge on a solution
6. Requirements initially agreed at a high level
7. All changes made during development are reversible
8. Testing is integrated throughout the life cycle
9. Collaborative and co-operative approach among all stakeholders essential

DSDM Lifecycle-

- Feasibility study
- Business study – prioritized requirements
- Functional model iteration
  - risk analysis
  - Time-box plan
- Design and build iteration
- Implementation

Scrum

Scrum is an iterative and incremental agile software development method for managing software projects and product or application development. Scrum has not only reinforced the interest in project management but also challenged the conventional ideas about such management. Scrum focuses on project management institutions where it is difficult to plan ahead. Mechanisms of empirical process control, where feedback loops that constitute the core management technique are used as opposed to traditional command-and-control oriented management. It represents a radically new approach for planning and managing projects, bringing decision-making authority to the level of operation properties and certainties



Sprint –

A sprint is the basic unit of development in Scrum. Sprints last between one week and one month, and are a "timeboxed" (i.e. restricted to a specific duration) effort of a constant length. Each sprint is preceded by a planning meeting, where the tasks for the sprint are identified and an estimated commitment for the sprint goal is made, and followed by a review or retrospective meeting, where the progress is reviewed and lessons for the next sprint are identified.

During each sprint, the team creates finished portions of a product. The set of features that go into a sprint come from the product backlog, which is a prioritized list of requirements. Which backlog items go into the sprint (the sprint goals) is determined during the sprint planning meeting.

During this meeting, the Product Owner informs the team of the items in the product backlog that he or she wants completed (the ones with the highest priority). The team then determines how much of this they can commit to complete during the next sprint, and records this in the sprint backlog.

The sprint backlog is property of the development team, i.e. during a sprint, no one is allowed to edit the sprint backlog except for the development team. The sprint goals should not be changed during the sprint.

Development is timeboxed such that the sprint must end on time; if requirements are not completed for any reason they are left out and returned to the product backlog. After a sprint is completed, the team demonstrates how to use the software.

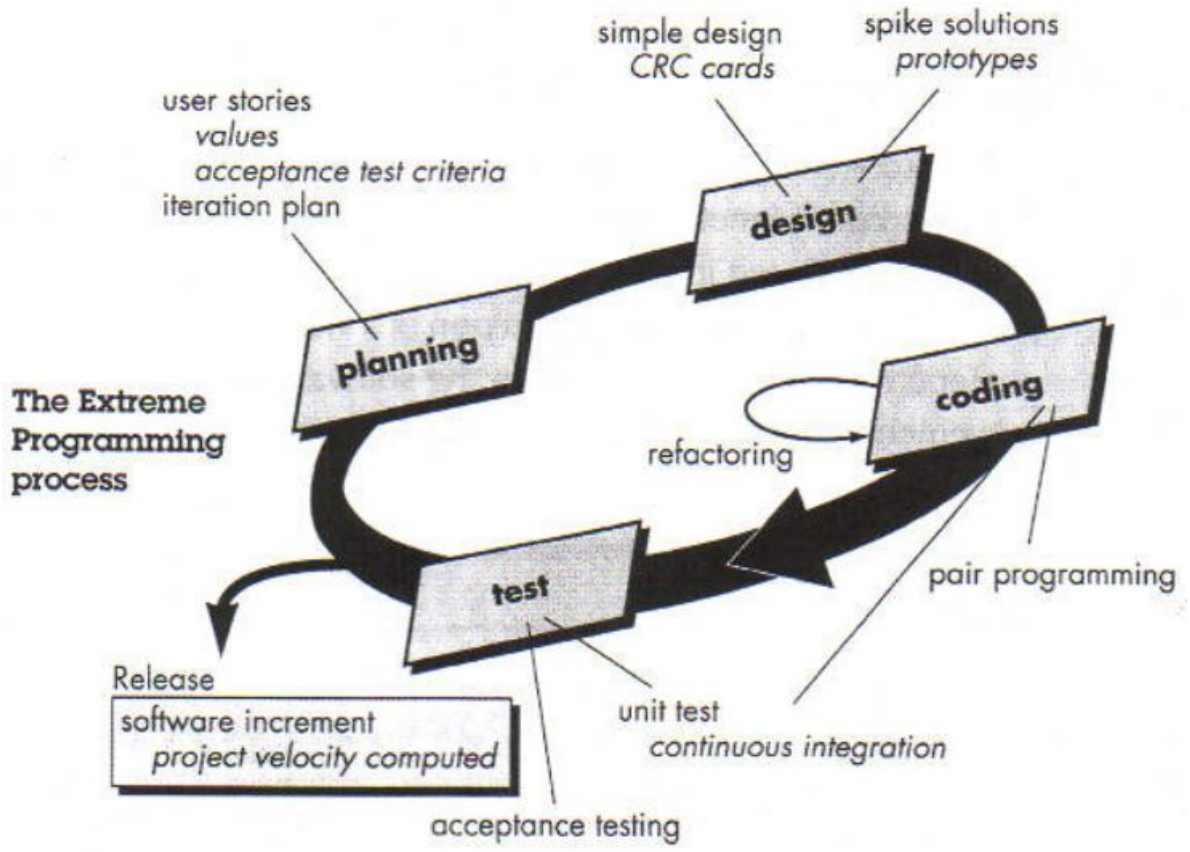
Scrum enables the creation of self-organizing teams by encouraging co-location of all team members, and verbal communication between all team members and disciplines in the project.

Extreme Programming (XP)

For small-to-medium-sized teams developing software with vague or rapidly changing requirements

Coding is the key activity throughout a software project

- Communication among teammates is done with code
- Life cycle and behavior of complex objects defined in test cases – again in code



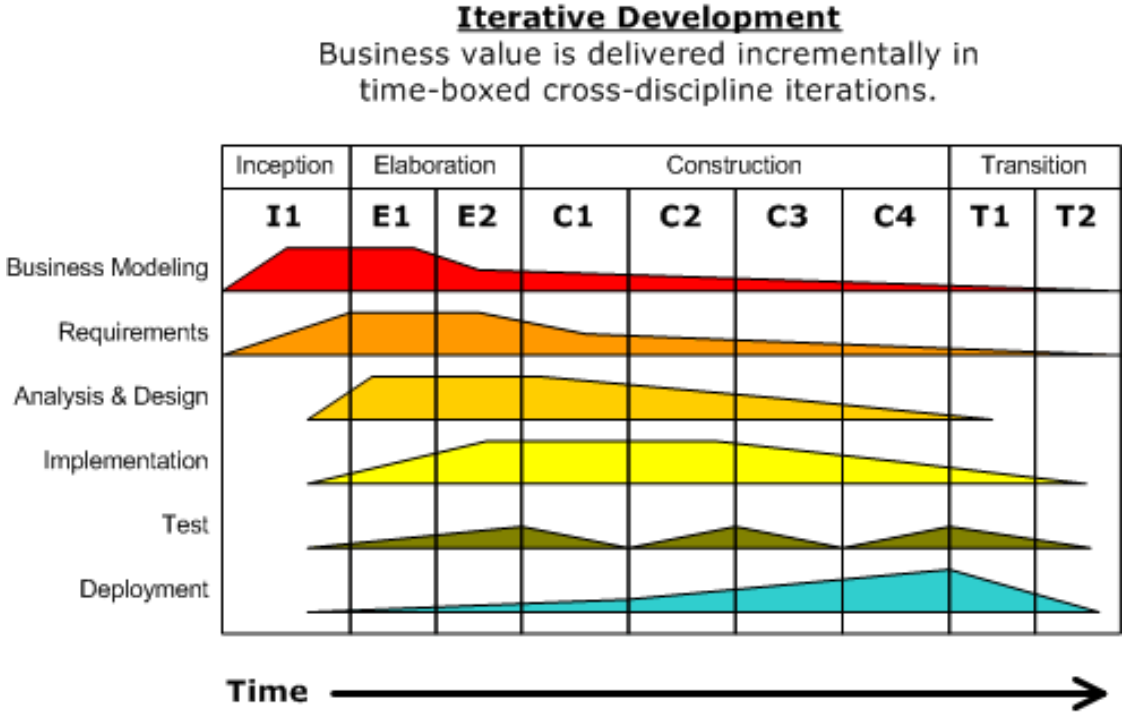
XP Practices-

1. Planning game – determine scope of the next release by combining business priorities and technical estimates
2. Small releases – put a simple system into production, then release new versions in very short cycle
3. Metaphor – all development is guided by a simple shared story of how the whole system works
4. Simple design – system is designed as simply as possible (extra complexity removed as soon as found)
5. Testing – programmers continuously write unit tests; customers write tests for features
6. Refactoring – programmers continuously restructure the system without changing its behavior to remove duplication and simplify
7. Pair-programming -- all production code is written with two programmers at one machine
8. Collective ownership – anyone can change any code anywhere in the system at any time.
9. Continuous integration – integrate and build the system many times a day – every time a task is completed.
10. 40-hour week – work no more than 40 hours a week as a rule
11. On-site customer – a user is on the team and available full-time to answer questions

12. Coding standards – programmers write all code in accordance with rules emphasizing communication through the code

Rational Unify Process (RUP)

The Rational Unified Process (RUP) is an iterative software development process framework created by the Rational Software Corporation, a division of IBM since 2003.[1] RUP is not a single concrete prescriptive process, but rather an adaptable process framework, intended to be tailored by the development organizations and software project teams that will select the elements of the process that are appropriate for their needs. RUP is a specific implementation of the Unified Process.



Six Best Practices

Six Best Practices as described in the Rational Unified Process is a paradigm in software engineering, that lists six ideas to follow when designing any software project to minimize faults and increase productivity. These practices are:[6][7]

Develop iteratively

> It is best to know all requirements in advance; however, often this is not the case. Several software development processes exist that deal with providing solution on how to minimize cost in terms of development phases.

Manage requirements

> Always keep in mind the requirements set by users.

Use components

> Breaking down an advanced project is not only suggested but in fact unavoidable. This promotes ability to test individual components before they are integrated into a larger system. Also, code reuse is a big plus and can be accomplished more easily through the use of object-oriented programming.

Model visually

Use diagrams to represent all major components, users, and their interaction. "UML", short for Unified Modeling Language, is one tool that can be used to make this task more feasible.

Verify quality

Always make testing a major part of the project at any point of time. Testing becomes heavier as the project progresses but should be a constant factor in any software product creation.

Control changes

Many projects are created by many teams, sometimes in various locations, different platforms may be used, etc. As a result it is essential to make sure that changes made to a system are synchronized and verified constantly.

Agile Modeling (AM)

Agile Modeling is a practice-based methodology for modeling and documentation of software-based systems. It is intended to be a collection of values, principles, and practices for Modeling software that can be applied on a software development project in a more flexible manner than traditional Modeling methods.[1]

Agile Modeling is a supplement to other Agile methodologies such as:

- Extreme Programming
- Agile Unified Process
- Scrum

In which it is used to replace the more standard UML, or other standard design tools.

The principles and values of Agile Modeling practices are intended to help mitigate the perceived flaws of Agile Software Development. The principle "Maximize stakeholder value" aims to inspire the developer to collaborate with the customer in providing an adequate level of documentation.

The principle "Model with others" attempts to involve project stakeholders such as the client or customer in the Modeling process to attempt to bring the model more into line with the end user requirements.

Limitations -

There is significant dependence on personal communication and customer collaboration. Agile Modeling disciplines can be difficult to apply where there are large teams (in Agile methodologies 'large' is typically considered as anything more than around 8), team members are unable to share and collaborate on models, or modeling skills are weak or lacking. However, the emergence of cloud modeling offerings that respect the tenets of Agile software development may significantly reduce or even eliminate most of these concerns.