SOFTWARE ENGINEERING

(CSE3005)

7/18/2019
Prof. Anand Motwani
Faculty, SCSE
VIT Bhopal University

Unit I
SOFTWARE PROCESS and MODELS

The Nature Of Software, Software Engineering, Software Process - Software Myths - Process Models – Generic –Perspective – Specialized – The Unified Process – Personal And Team Software Process - Agile Development – Agile Process- Extreme Programming - Software Engineering Knowledge – Core Principles.

Aims and Objectives

- To introduce the basic concepts of software development Process, models and methods.
- To appraise the need for a professional approach to software system development;

1. Introduction

Computer software has become a driving force. It is the engine that drives business decision making. It serves as the basis for modern scientific investigation and engineering problem solving. It is a key factor that differentiates modern products and services. . It is embedded in systems of all kinds: transportation, medical, telecommunications, military, industrial processes, entertainment, office products... Software is virtually inescapable in a modern world. And as we move into the 21st century, it will become the driver for new advances in everything from elementary education to genetic engineering.

Software's impact on our society and culture continues to be profound. As its importance grows, the software community continually attempts to develop technologies that will make it easier, faster, and less expensive to build high-quality computer programs. Some of these technologies are targeted at a specific application domain (e.g., Web-site design and implementation); others focus on a technology domain (e.g., object-oriented systems); and still others are broad-based (e.g., operating systems such as LINUX).

2. What is software? Write it's roles.

Software

Not only the computer program (s) but also

1) Instructions (programs) that when executed provide desired function and performance

2) Data structures that enable the programs to adequately manipulate information

3) Documents that describe the operation and use of the programs

So, Software is a set of items or objects that form a "configuration" that includes programs, data structures, documents and data etc.

Or

Alternate definition: Software is the collection of computer programs, procedures, Rules and associate with documentation and data. Product that engineers design & build

(1) Programs that execute within a computer of any size & architecture.

(2) Documents

(3) Data (Text, Video etc.)

Software is everywhere: PC, phones, electronic devices (e.g. TV, washing machine, . . .), computing centre, Web server, . . .

Software's Dual Roles

- Software is a product
  - Delivers computing potential Delivers potential
  - Produces, manages, acquires, modifies, displays, or transmits information
- Software is a vehicle for delivering a product Software product
  - Supports or directly provides system functionality Supports functionality
  - Controls other programs (e.g., an operating system)
  - Effects communications (e.g., networking software)
  - Helps build other software (e.g., software tools)

3. Define Software Engineering?

Software Engineering is a systematic approach to development, operation, maintenance and retirement of software.

Or

Software Engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures and associated with documentation.

Or

Software engineering (SE) is the application of systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.

Or

SE is an engineering discipline concerned with all aspects of software production from early specifications through to maintenance.

Or

SE is the profession that creates and maintains software applications by applying technologies and practices from computer science, project management, engineering, application domain, and other fields.

Goal of the Software Engineering is to produce high quality software at low cost.

4. Write Characteristics of Software. OR Write about nature of Software.

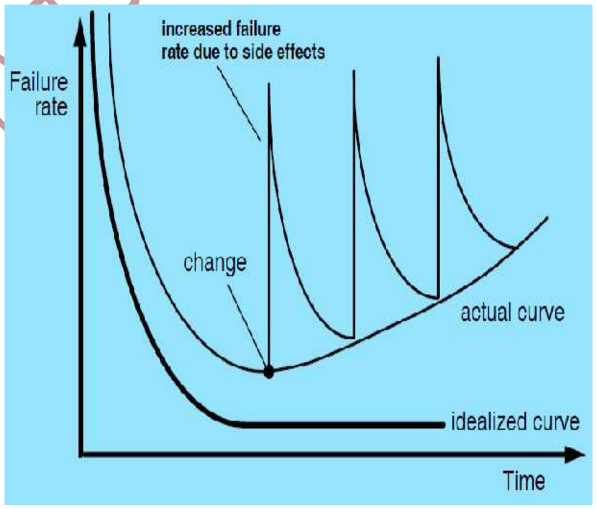
The economies of ALL developed nations are dependent on software. More and more systems are software controlled Software engineering is concerned with theories, methods and tools for professional software development. Expenditure on software represents a significant fraction of GNP in all developed countries.

- **Software is engineered:** Software is engineered or developed, not manufactured in the traditional sense. Some general approaches to develop software.
    - Develop and use good engineering practices for building software.
    - Make heavy use of reusable software components.
    - Use modern languages that support good software development practices, e.g. Java.
    - Use 4th generation languages.

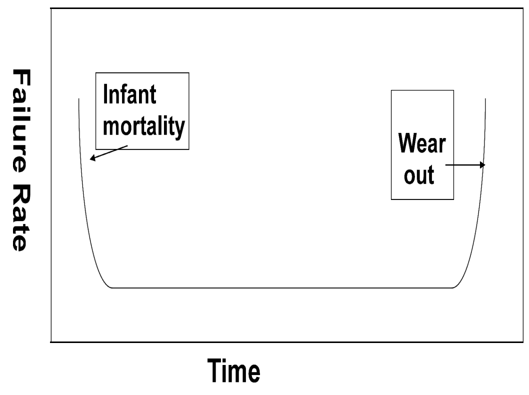
But, almost everything is a two-edged sword while considering long term tool maintenance.

- **Software does not wear out:** Software does not wear out in the same sense as hardware. In theory, software does not wear out at all but, as Hardware upgrades Software also upgrades.
- **Software is complex**
    - adding another button to an alarm clock costs money during manufacturing of each piece
    - the cost is manufacturer's
    - adding a butting to a program may cost something during software development (or not)

Its consequence is more complex software which may be more difficult to use and the user pays



Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University



Bathtub curve

5. Types of Software Applications

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- WebApps (Web applications)
- AI software

Software New Categories Software—

- Ubiquitous computing——wireless networks
- Netsourcing——the Web as a computing engine
- Open source——””free””source code open to the computing community (a blessing, but also a potential curse!)
- Data mining
- Grid computing
- Cognitive machines
- Software for nanotechnologies

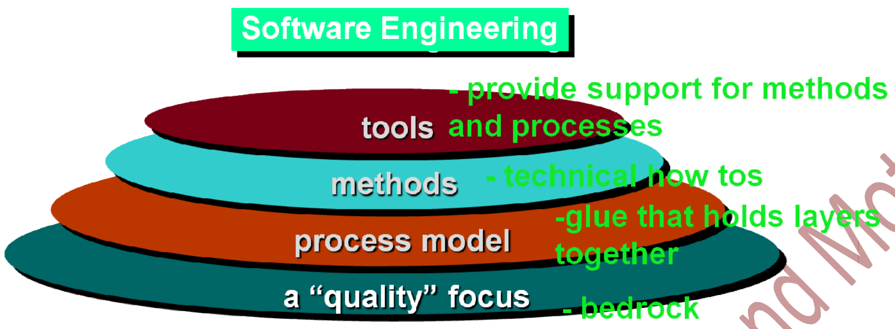
Define Legacy Software.

- Software must be adapted to meet the needs of new to computing environments or technology.
- Software must be enhanced to implement new to business requirements.
- Software must be extended to make it interoperable with other more modern systems or databases.
- Software must be re-architected to make it viable within a network environment.

6. Software Engineering is a layered technology. Explain.

Or

Define Layers of Software Engineering.



Software Process

A software process consists of a set of activities and associated results which lead to the production of a software product

Methods

- Includes standards (formal or informal)
- May include conventions, e.g., low level such as naming, variable, language construct use, etc.
- May involve design methodologies.

List the Generic Software Engineering Phases/Activities.

Tools

- Editors
- Design aids
- Compilers
- Computer Aided Software Engineering (CASE)

7. What is a software process?

A set of activities whose goal is the development or evolution of software. Another definition:

It is a series of predictable steps to build a product or system ensuring timely & high quality result.

8. Write Fundamental / Generic activities of all software processes.

Fundamental / Generic activities of all software processes are:

1. Software specification
2. Software design and implementation
3. Software validation
4. Software evolution

Or

System or information engineering (leading to requirements)

- Software project planning
- Requirements analysis
- Development
- Software design
- Coding
- Testing
- Deployment
- Maintenance

Typical activities in these (SE) phases

- Project tracking and control
- Formal reviews
- Software quality assurance
- Configuration management
    - versions, group of versions = configuration
- Documentation
- Reusability management
- Measurements
- Risk management

9. Software Myths:

Definition: Beliefs about software and the process used to build it. Myths have number of attributes that have made them insidious (i.e. dangerous).

- Misleading Attitudes - caused serious problem for managers and technical people.

Management myths

Managers in most disciplines, are often under pressure to maintain budgets, keep schedules on time, and improve quality.

Myth1: We already have a book that's full of standards and procedures for building software, won't that provide my people with everything they need to know?

Reality :

- Are software practitioners aware of existence standards?
- Does it reflect modern software engineering practice?
- Is it complete? Is it streamlined to improve time to delivery while still maintaining a focus on quality?

Myth2*:* If we get behind schedule, we can add more programmers and catch up

Reality*:* Software development is not a mechanistic process like manufacturing. Adding people to a late software project makes it later.

- People can be added but only in a planned and well-coordinated manner

Myth3*:* If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality*:* If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsource software projects

Customer Myths

Customer may be a person from inside or outside the company that has requested software under contract.

Myth*:* A general statement of objectives is sufficient to begin writing programs— we can fill in the details later.

Reality*:* A poor up-front definition is the major cause of failed software efforts. A formal and detailed description of the information domain, function, behavior, performance, interfaces, design constraints, and validation criteria is essential. These characteristics can be determined only after thorough communication between customer and developer.

Myth*:* Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality: Customer can review requirements and recommend modifications with relatively little impact on cost. When changes are requested during software design, the cost impact grows rapidly. Below mentioned *figure* for reference.

Practitioner's myths

Myth1*:* Once we write the program and get it to work, our job is done.

Reality*:* Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth2*:* Until I get the program "running" I have no way of assessing its quality.

Reality: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the *formal technical review.*

Myth3: The only deliverable work product for a successful project is the working program.

10. Define Process Models (Software Engineering paradigms). How it is chosen?

It is the development strategy that encompasses the process, methods, tools & generic phases i.e. definition, development & support & followed by software engineer or team of E's to develop software.

A simplified representation of a software process, presented from a specific perspective. Examples of process perspectives are

- Workflow perspective - sequence of activities;
- Data-flow perspective - information flow;
- Role/action perspective - who does what.

Few Generic process models are:

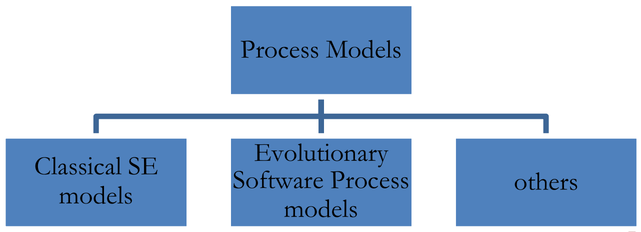
- Waterfall;
- Iterative development;

- Component-based software engineering.

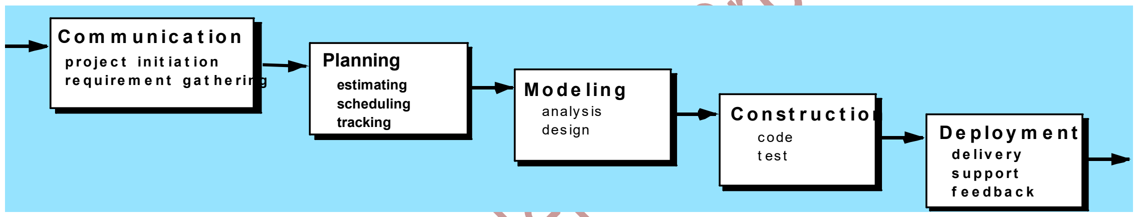
Process models chosen based on:

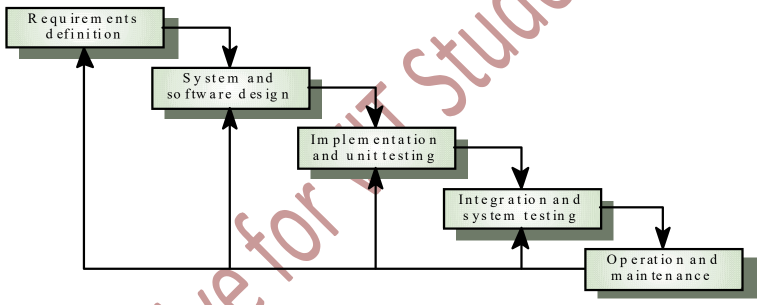
- Nature of project & application
- Methods & Tools to be used
- Controls & deliverables those are required.

11. Show classification of Process Models. Describe few models.



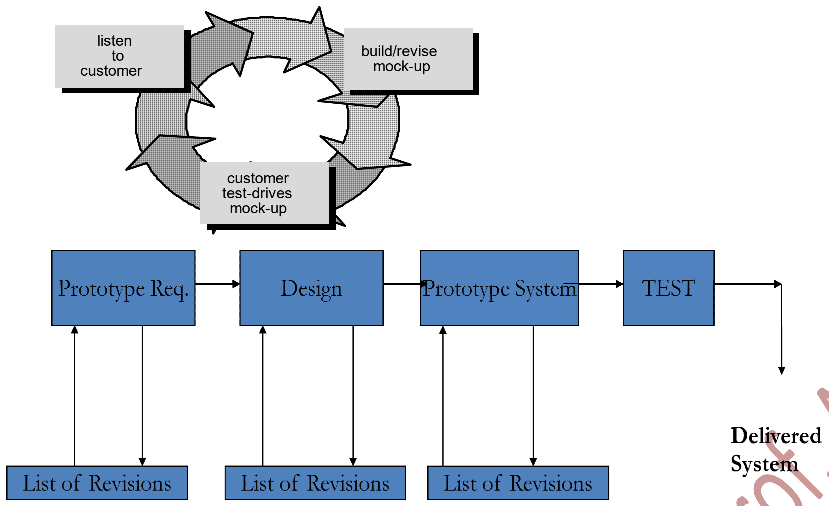
11.1 The Waterfall Model




Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance
- The drawback of the waterfall model is the difficulty of accommodating change after the process is underway
- Inflexible partitioning of the project into distinct stages
- This makes it difficult to respond to changing customer requirements
- Therefore, this model is only appropriate when the requirements are well-understood.

11.2 Prototyping Prototype Model



Advantages

- A partial product is built in initial stages, customer get chance to see it.
- New requirement easily accommodated.
- More secure, comfortable & satisfied.

Disadvantages

- User not know the difference between prototype and actual well engineered system.
- User expects the early delivery.
- If not managed properly iterative process of prototype can continue for long time.
- Poor documentation.

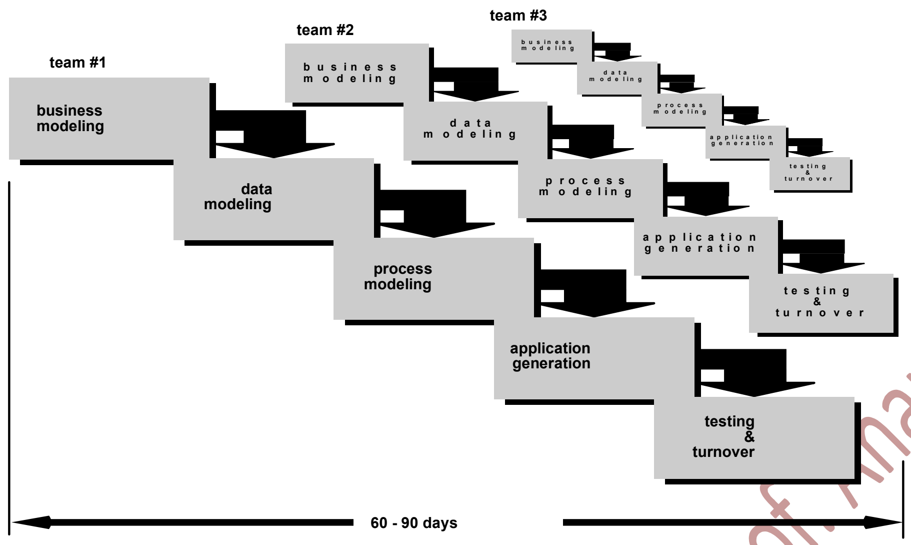
11.3 Rapid Application Development (RAD)

- Proposed by IBM (1980s')
  - High speed adaptation of Linear sequential model
  - Rapid development achieved by using component based construction
  - Increase involvement of customer
  - Rapid prototype delevired to customer (working model)

Advantages

- Customer satisfaction.
- use of tools results into reduced SDLC.

- Reusable components, reduces testing time.
- Feedback available
- Reduced costs



Disadvantages

- For large projects, it required skilled professionals and hence not suitable when technical risks are high (heavy use of new tech.).

- Absence of reusable components, i.e. if high performance good tuning of interfaces required.

11.4 The Incremental Model

- Combines elements of linear seq. model applied repetitively with the iterative philosophy of prototyping.

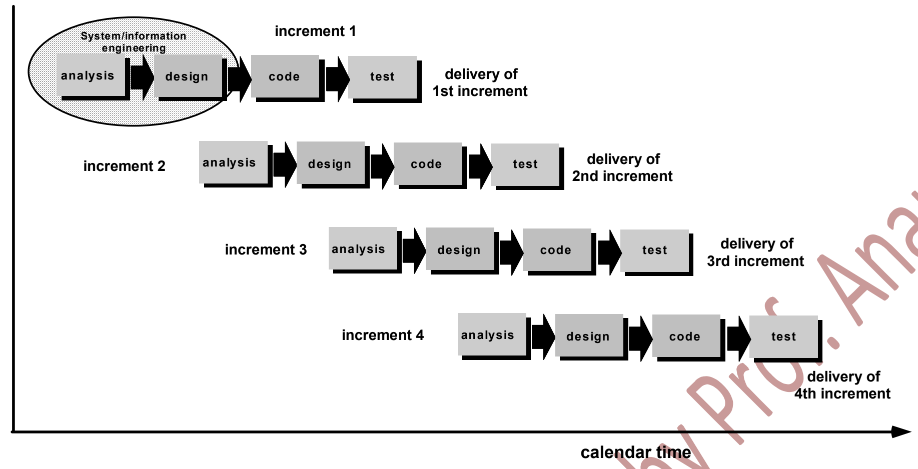
Incremental development advantages

- Customer value can be delivered with each increment so system functionality is available earlier
- Early increments act as a prototype to help elicit requirements for later increments
- Lower risk of overall project failure
- The highest priority system services tend to receive the most testing
- Total cost distributed
- Limited no. of persons
- Development activities for the next release & use of early version done simultaneously
- Testing easy

- Low risk of failures

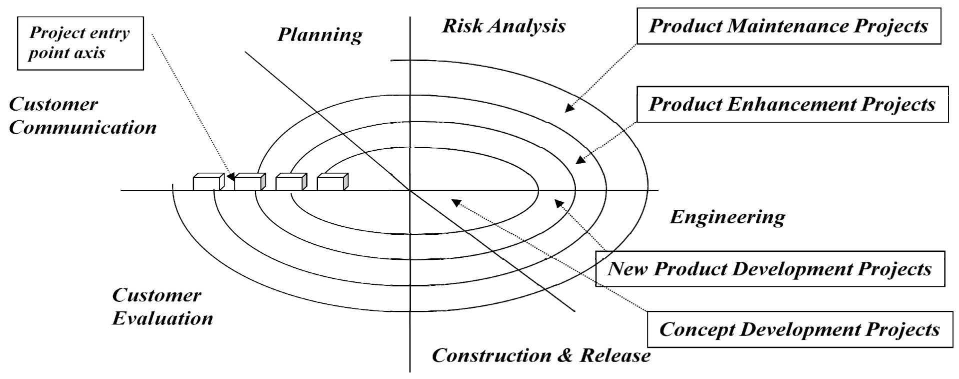
Incremental development Disadvantages

- High Development cost
- Project planning schedule to distribute work
- Testing of modules increases cost.



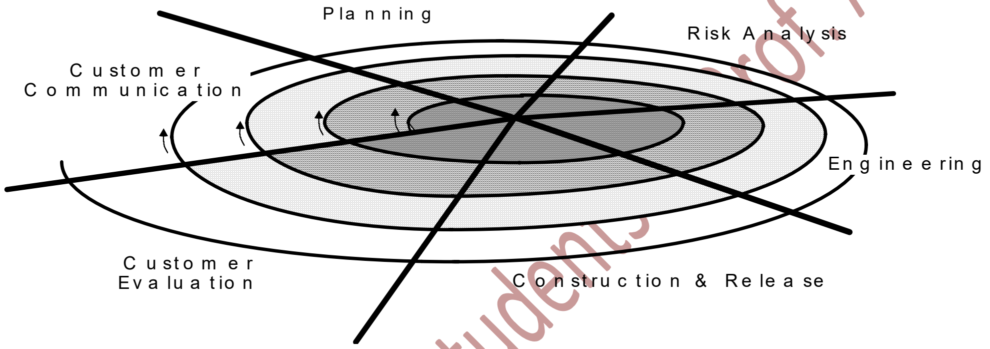
11.5 An Evolutionary (Spiral) Model

- Proposed by Boehm (1988)
- Combines iterative nature of prototyping with the controlled and systematic approach of Linear Sequential model.
- It provides the potential for rapid development of Incremental versions of software.
- It has 3-6 task regions.
- It explicitly embraces prototyping and an *iterative* approach to software development.
    - Start by developing a small prototype.
    - Followed by a mini-waterfall process, primarily to gather requirements.
    - Then, the first prototype is reviewed.
    - In subsequent loops, the project team performs further requirements, design, implementation and review.
    - The first thing to do before embarking on each new loop is risk analysis.
    - Maintenance is simply a type of on-going development.



The spiral model
-- a realistic approach to the development of large scale systems and software.
- may be difficult to convince customers that the evolutionary approach is controllable.
- a relatively new model, and has not used as widely as the linear sequential or prototyping paradigms.

<u>Phases / Tasks of Evolutionary (Spiral) Model</u>



1. Customer communication 2. Planning 3. Risk Analysis 4. Engineering
5. Construction & Release 6. Customer Evaluation

Each task region is populated by task set.

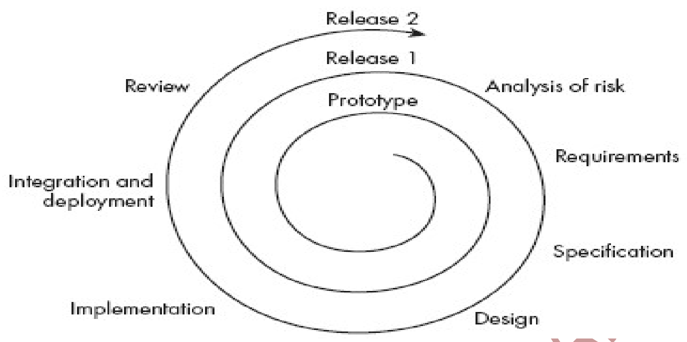
Spiral development

- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.

Spiral model sectors

- Objective setting
    - Specific objectives for the phase are identified.
- Risk assessment and reduction

    - Risks are assessed and activities put in place to reduce the key risks.
- Development and validation
    - A development model for the system is chosen which can be any of the generic models.
- Planning
    - The project is reviewed and the next phase of the spiral is planned.



Spiral model Advantages

It resolves all possible risks, so get redefined Product. It involves Use of techniques: reuse, prototyping, & component based development which is having its own advantages.

Disadvantages

It is Complex so requires (i) Risk management expertise and (ii) Good management skills.

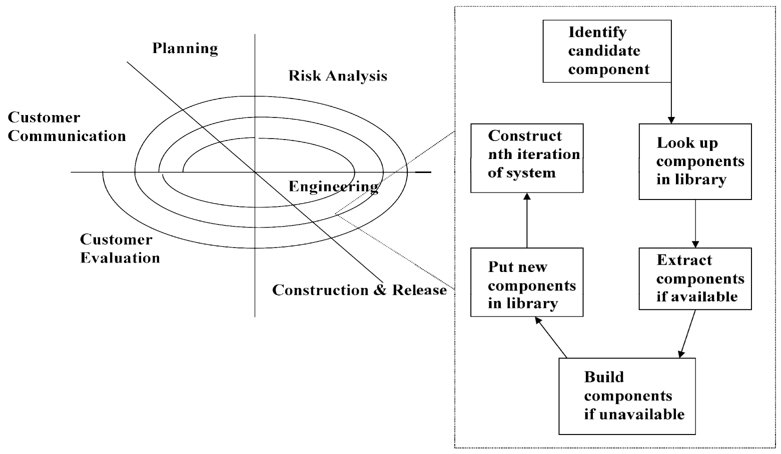
<u>11.6 Component</u>

What is a component?

1. A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces. (Philippe Krutchen, Rational Software).
2. A runtime software component is a dynamically bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at runtime. (Gartner Group)

3. A reusable part of software, which is independently developed, and can be brought together with other components to build larger units. It may be adapted but may not be modified. A component can be, for example, a compiled code without a program source, or a part of a model and/or design.

The Component Assembly Model


Planning
Risk Analysis
Customer Communication
Engineering
Customer Evaluation
Construction & Release
Identify candidate component
Look up components in library
Extract components if available
Build components if unavailable
Put new components in library
Construct nth iteration of system

- Object technologies provide the technical framework for a component-based process model for software engineering.
- The object oriented paradigm emphasizes the creation of classes that encapsulate both data and the algorithm that are used to manipulate the data.
- If properly designed and implemented, object oriented classes are reusable across different applications and computer based system architectures.
- Component Assembly Model leads to software reusability. The integration/assembly of the already existing software components accelerates the development process.
- Nowadays many component libraries are available on the Internet. If the right components are chosen, the integration aspect is made much simpler.
- **Object Technologies -- the technical framework for a component-based process model for software engineering.**
- **The component assembly model incorporates many of the characteristics of the spiral model. It is an evolutionary in nature and demands an iterative approach to the creation of software. So leads to software reuse, and reusability.**
- **Its Advantages includes software reuse, cost reduction and reduction in development cycle time**

12. Rational Unified Process

- ► Introduction
- ► Phases
- ► Core Workflows
- ► Best Practices
- ► Tools

Introduction:

The Rational Unified Process™ (RUP) is a Web-enabled software engineering process that enhances team productivity and delivers software best practices to all team members. This e-coach makes process practical by providing prescriptive guidelines, templates and examples for all critical e-development activities. RUP is a customizable framework, which can easily be adapted to work the way you work. It is tightly integrated with Rational tools, allowing development teams to gain the full benefits of the Unified Modeling Language™ (UML), software automation, and other industry best practices.

In short

- A modern process model derived from the work on the UML and associated process.
- It is a good example of hybrid Process model.
- It brings together elements from all of the generic process model, supports iteration, and illustrates good practice in specification & design.
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practice perspective that suggests good practices.

► **Team-Unifying Approach**:
The RUP unifies a software team by providing a common view of the development process and a shared vision of a common goal.

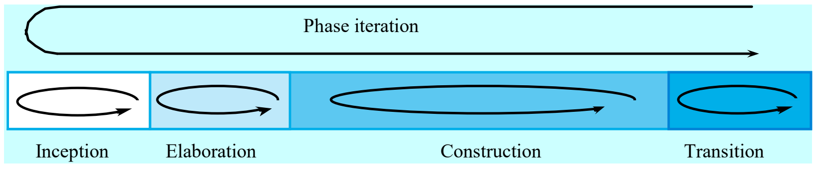
► **Increased Team Productivity**
  - Knowledge base of all processes.
  - View of how to develop software
  - Modeling language
  - Rational provides many tools

The Rational Unified Process has four phases:

- **Inception** - Define the scope of project. Establish the business case for the system, i.e. identify all external entities (people & systems) that will interact with the system & define these interactions. Then use this information to assess the contribution that the system makes to the business. If this contribution is minor, then the project may be cancelled after this phase.
-
- **Elaboration** - Plan project, specify features, baseline architecture. Develop an understanding of the problem domain and the system architecture. Develop the project plan & identify key project risks. **After this phase, you should have:** Requirement model for system, Architectural description and Development plan.
- **Construction** - Build the product. System design, programming and testing. Parts of the system are developed in parallel & integrated during this phase. **On**

completion of this phase, you should have: working software system and associated documents ready for delivery to users

- **Transition** - Transition the product into end user community i.e. Deploy the system in its operating environment.



The static view of RUP focuses on activities that take place during the core process. These are called workflows.

6 core workflows

- Business modeling
- Requirements
- Analysis & Design
- Implementation
- Testing
- Deployment

3 core supporting workflows

- Configuration & Change management
- Project management
- Environment

Workflow	**Description**
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Test	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow managed changes to the system.
Project management	This supporting workflow manages the system development.
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

RUP Good Practices

- Six fundamental best practices are recommended:
  - Develop software iteratively
  - Manage requirements
  - Use component-based architectures
  - Visually model software
  - Verify software quality
  - Control changes to software

13. Agile Development

Definition of Agile:

An iterative and incremental (evolutionary) approach performed in a highly collaborative manner with just the right amount of ceremony to produce high quality software in a cost effective and timely manner which meets the changing needs of its stakeholders.

Core principles
"Fits just right" process
Continuous testing and validation
Consistent team collaboration
Rapid response to change
Ongoing customer involvement
Frequent delivery of working software

Agile Development Adopt disciplined, adaptive development approaches. Use continuous stakeholder feedback to deliver high quality and consumable code through use cases and a series of short, time-boxed iterations.
Business Result
Quality first
Teams build software
Minimize waste
Rapid feedback and response
One extended team
Integrated development tools
Adaptive planning

14. Personal and Team Software Process

W HAT I S P ERSONAL S OFTWARE P ROCESS (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 eliminate them.

PSP MODEL F RAMEWORK A CTIVITIES

- Planning:– isolates requirements and based on these develops both size & resource estimates. A defect estimate is made.
- High level Design:– external specification of all components. All issues are recorded and tracked.
- High level Design Review:- formal verification to uncover errors
- Development:- metrics are maintained for all important tasks & work results.
- Postmortem:- using measures & metrics collected effectiveness of process is determined an improved.

P ERSONAL S OFTWARE P ROCESS 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.

PSP.. The PSP can be applied to many parts of the software development process, including

- small-program development
- requirement definition
- document writing
- systems tests
- systems maintenance
- enhancement of large software systems.

W HAT I S T EAM S OFTWARE P ROCESS (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.

TSP F RAMEWORK A CTIVITIES Launch high level design Implementation Integration Test postmortem.

8 TSP.. Engineering groups use the TSP to apply integrated team concepts to the development of software-intensive systems. A launch process walks teams and their managers through
- establishing goals
- defining team roles
- assessing risks
- producing a team plan.

BENEFITS OF TSP The TSP provides a defined process framework for managing, tracking and reporting the team's progress. Using TSP, an organization can build self- directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams of 3 to 20 engineers. TSP will help your organization establish a mature and disciplined engineering practice that produces secure, reliable software.

15. Extreme Programming

Definition
Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

When Applicable:

The general characteristics where XP is appropriate were described by Don Wells on www.extremeprogramming.org:

Dynamically changing software requirements
Risks caused by fixed time projects using new technology
Small, co-located extended development team
The technology you are using allows for automated unit and functional tests
Due to XP's specificity when it comes to it's full set of software engineering practices, there are several situations where you may not want to fully practice XP. The post When is XP Not Appropriate on the C2 Wiki is probably a good place to start to find examples where you may not want to use XP.

While you can't use the entire XP framework in many situations, that shouldn't stop you from using as many of the practices as possible given your context.

Values
The five values of XP are communication, simplicity, feedback, courage, and respect

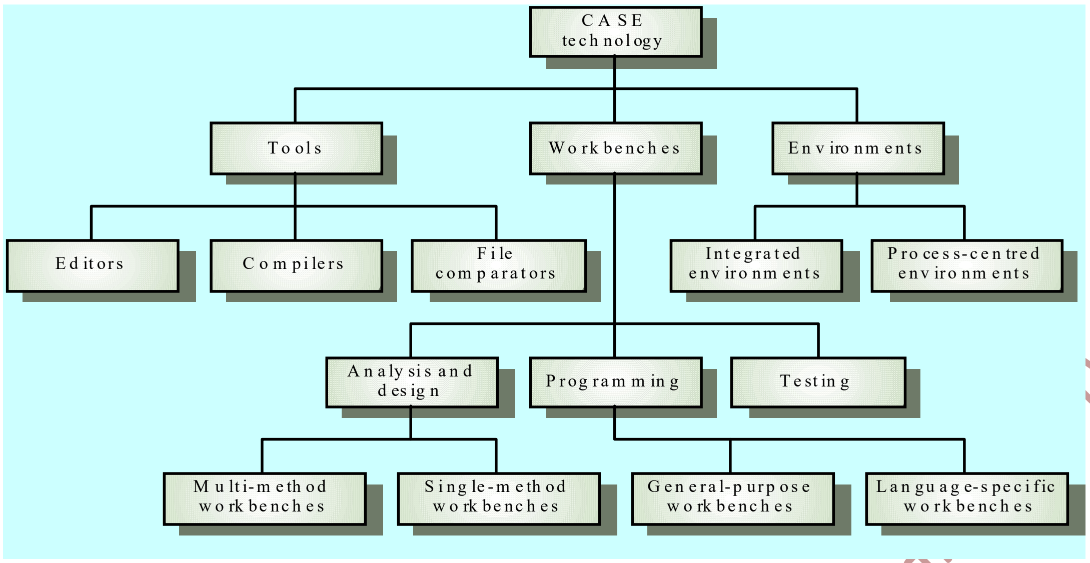
Refer the link:

https://www.agilealliance.org/glossary/xp/#q=~(infinite~false~filters~(postType~(~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'xp))~searchTerm~'~sort~false~sortDirection~'asc~page~1)

15. Software Engineering Knowledge

15.1 CASE (Computer-Aided Software Engineering)

CASE tools are software systems which are designed to support routine activities in the software process such as editing design diagrams, checking diagram consistency and keeping track of program tests which have been run.



<u>15.2 Product and Process Metrics</u>

Few Definitions

Measure -- Provides a quantitative indication of the extent, amount, dimensions, capacity, or size of some product or process attribute (1000 lines)

☐ *Metrics -- A quantitative measure of the degree to which a system, component, or process possesses a given attribute (40%)*

☐ *Software Metrics -- refers to a broad range of measurements for computer software.*

☐ *Indicator -- A metric or combination of metrics that provide insight into the software process, software project, or the product itself.*

- Two categories of software measurement
  - Direct measures of the
    - Software process (cost, effort, etc.)
    - Software product (lines of code produced, execution speed, defects reported over time, etc.)
  - Indirect measures of the
    - Software product (functionality, quality, complexity, efficiency, reliability, maintainability, etc.)
- Project metrics can be consolidated to create process metrics for an organization

Size-oriented Metrics

- Derived by normalizing quality and/or productivity measures by considering the size of the software produced
- Thousand lines of code (KLOC) are often chosen as the normalization value
- Metrics include
  - Errors per KLOC - Errors per person-month
  - Defects per KLOC - KLOC per person-month
  - Dollars per KLOC - Dollars per page of documentation
  - Pages of documentation per KLOC
- Size-oriented metrics are not universally accepted as the best way to measure the software process
- Opponents argue that KLOC measurements
  - Are dependent on the programming language
  - Penalize well-designed but short programs
  - Cannot easily accommodate nonprocedural languages
  - Require a level of detail that may be difficult to achieve

Function-oriented Metrics

- Function-oriented metrics use a measure of the functionality delivered by the application as a normalization value

Most widely used metric of this type is the function point:

FP = count total * [0.65 + 0.01 * sum (value adj. factors)]

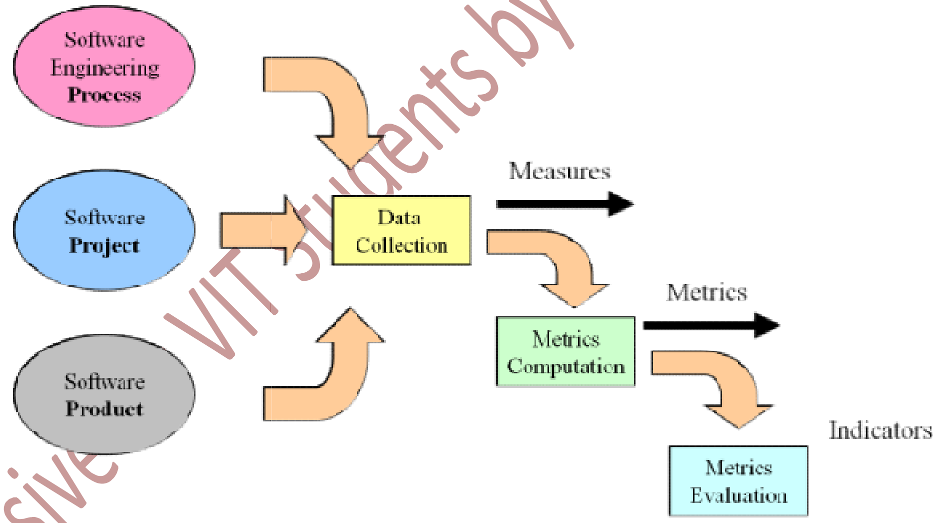
- Function point values on past projects can be used to compute, for example, the average number of lines of code per function point (e.g., 60)
- Like the KLOC measure, function point use also has proponents and opponents
- Proponents claim that
  - FP is programming language independent
  - FP is based on data that are more likely to be known in the early stages of a project, making it more attractive as an estimation approach
- Opponents claim that
  - FP requires some "sleight of hand" because the computation is based on subjective data
  - Counts of the information domain can be difficult to collect after the fact
  - FP has no direct physical meaning…it's just a number
- Relationship between LOC and FP depends upon
  - The programming language that is used to implement the software

    – The quality of the design

- FP and LOC have been found to be relatively accurate predictors of software development effort and cost
    - However, a historical baseline of information must first be established
- LOC and FP can be used to estimate object-oriented software projects
    - However, they do not provide enough granularity for the schedule and effort adjustments required in the iterations of an evolutionary or incremental process

A rough estimate tells that average LOC to one FP in various programming languages.

Language	Average	Median	Low	High
C++	66	53	29	178
Java	55	53	9	214
Visual Basic	47	42	16	158



Software Engineering Process
Software Project
Software Product
Data Collection
Measures
Metrics Computation
Metrics
Metrics Evaluation
Indicators