# SOFTWARE ENGINEERING

(CSE3005)

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

Exclusive for VIT Students by Prof. Anand Motwani

# 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).

Exclusive for VIT Students by Prof. Anand Motwani



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

**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)

Exclusive for VIT Students by Prof. Anand Motwani



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

**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.

Exclusive for VIT Students by Prof. Anand Motwani



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

**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

Exclusive for VIT Students by Prof. Anand Motwani



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.

Exclusive for VIT Students by Prof. Anand Motwani



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

**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

Exclusive for VIT Students by Prof. Anand Motwani



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

**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

Exclusive for VIT Students by Prof. Anand Motwani



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

**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;

Exclusive for VIT students by Prof. Anand Motwani



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

- 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 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.

Exclusive for VIT Students Prof. Anand Motwani



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

## 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.

Exclusive for VIT Students by Prof. Anand Motwani



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

- 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

Exclusive for VIT Students by Prof. Anand Motwani



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

- 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.

Exclusive for VIT Students by Prof. Anand Motwani



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

*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.*

## Phases / Tasks of Evolutionary (Spiral) Model

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

Exclusive for VIT Students Prof. Anand Motwani



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

- 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.

## 11.6 Component

**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.

Exclusive for VIT Students by Prof. Anand Motwani



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

## The Component Assembly Model

- 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**

Exclusive for VIT Students by Prof. Anand Motwani



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

## 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**

Exclusive for VIT Students by Prof. Anand Motwani



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

**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. |

Exclusive for VIT Students by Prof. Anand Motwani



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

**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.
**<u>Business Result</u>**
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.

Exclusive for VIT Students by Prof. Anand Motwani



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

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.

Exclusive for VIT Students by Prof. Anand Motwani



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

## 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.

Exclusive for VIT Students by Prof. Anand Motwani



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

## 15.2 Product and Process Metrics

*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

Exclusive for VIT Students by Prof. Anand Motwani



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

**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

Exclusive for VIT Students by Prof. Anand Motwani



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

- 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 |

Exclusive VIT students by Prof. Anand Motwani



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

- Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.
  User Input = 50
  User Output = 40
  User Inquiries = 35
  User Files = 6
  External Interface = 4
  Solution:
  **Step-1:**
  As complexity adjustment factor is average (given in question), hence,
  scale = 3.
  F = 14 * 3 = 42
  **Step-2:**
  CAF = 0.65 + ( 0.01 * 42 ) = 1.07
  **Step-3:**
  As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.
  UFP = (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628
  **Step-4:**
  Function Point = 628 * 1.07 = 671.96
  This is the required answer.

- **Counting Function Point (FP):**

- **Step-1:**
  F = 14 * scale
  Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:
  0 - No Influence
  1 - Incidental
  2 - Moderate
  3 - Average
  4 - Significant
  5 - Essential
- **Step-2:** Calculate Complexity Adjustment Factor (CAF).
  CAF = 0.65 + ( 0.01 * F )
- **Step-3:** Calculate Unadjusted Function Point (UFP).

  TABLE (Required)

| FUNCTION UNITS | LOW | AVG | HIGH |
|---|---|---|---|
| EI | 3 | 4 | 6 |
| EO | 4 | 5 | 7 |

| FUNCTION UNITS | LOW | AVG | HIGH |
|---|---|---|---|
| EQ | 3 | 4 | 6 |
| ILF | 7 | 10 | 15 |
| EIF | 5 | 7 | 10 |

Multiply each individual function point to corresponding values in TABLE.

- **Step-4:** Calculate Function Point.
  FP = UFP * CAF

| Measurement parameter | Count | Simple | Average | Complex | Count Total |
|---|---|---|---|---|---|
| Inputs | **50** | 3 | 4 | 6 | 50*4 = 200 |
| Outputs | **40** | 4 | 5 | 7 | 40*5=200 |
| inquiry | **35** | 3 | 4 | 6 | 140 |
| files | **6** | 7 | 10 | 15 | 60 |
| External interfaces | **4** | 5 | 7 | 10 | 28 |
| Total | | | | | = 628 |
| | | | | | |
| | | | | | |

$FP = counttotal * ¿ + 0.01 x \sum Fi ¿¿$

FP = counttotal x CAF

FP = 628 * [0.65 + 0.01 x (14x3)]

FP = 628 x 1.07 = 672

# SOFTWARE ENGINEERING

(CSE3005)

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

Exclusive for VIT Students by Prof. Anand Motwani

# Unit II

## REQUIREMENT ENGINEERING

*Understanding Requirements – Establish Ground Work – Eliciting Requirements - Developing Use case- Negotiating Requirements- Validating Requirements - Requirements Modeling – Requirement Analysis –Scenario Based Modeling - UML Supplements – Data Modeling Concepts- Class Based Modeling - Flow Oriented Modeling – Creating Behavioral Modeling - Patterns For Modeling.*

### Aims and Objectives

- To understand of software requirements specification.
- To understand Functional and Non-Functional requirements of Software.

### 1. Introduction

**Software Development Process: A Brief Overview**

Fig. 1. Software Development Process

Prof. Anand Motwani



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

***Understanding Requirements***

***"It Depends On Common Sense"***

- *Software requirements* are documentation that completely describes the behavior that is required of the software-before the software is designed built and tested.
  - Requirements analysts (or business analysts) build software requirements specifications through *requirements elicitation*.
    - Interviews with the users, stakeholders and anyone else whose perspective needs to be taken into account during the design, development and testing of the software
    - Observation of the users at work
    - Distribution of discussion summaries to verify the data gathered in interviews

**Discussion summary**

- A requirements analyst can use a *discussion summary* to summarize information gathered during elicitation and validate it through a review.
- Notes gathered during the elicitation should fit into the discussion summary template
- The discussion summary outline can serve as a guide for a novice requirements analyst in leading interviews and meetings

**Discussion Summary outline**

- ✓ Project background
- ✓ Purpose of project
- ✓ Scope of project
- ✓ Other background information
- ✓ Perspectives
- ✓ Who will use the system?
- ✓ Who can provide input about the system?
- ✓ Project Objectives
- ✓ Known business rules
- ✓ System information and/or diagrams
- ✓ Assumptions and dependencies

Exclusive for VIT Students by Prof. Anand Motwani



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

- ✓ Design and implementation constraints
- ✓ Risks
- ✓ Known future enhancements
- ✓ References
- ✓ Open, unresolved or 'to be determined' (TBD) issues

**Requirements Analysis and Specification**

Analysis is the process of specifying *what* a system will do.

- The intention is to provide a clear understanding of what the system is about and what its underlying concepts are.

The result of analysis is a *specification document*.

**Requirements Engineering**

- Elicitation — determining what the customer requires
- Analysis & negotiation — understanding the relationships among various customer requirements and shaping those relationships to achieve a successful result
- Requirements specification — building a tangible model of requirements
- System Modeling — building a representation of requirements that can be assessed for correctness, completeness, and consistency
- Validation — reviewing the model
- Management — identify, control and track requirements and the changes that will be made to them

**Functional and Non-functional requirements**

- ***Functional requirements*** define the outward behavior required of the software project.
  - The goal of the requirement is to communicate the needed behavior in as clear and unambiguous a manner as possible.
  - The behavior in the requirement can contain lists, bullets, equations, pictures, references to external documents, and any other material that will help the reader understand what needs to be implemented.
- ***Nonfunctional requirements*** define characteristics of the software which do not change its behavior.
  - Users have implicit expectations about how well the software will work.

Exclusive for VIT Students by Prof. Anand Motwani



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

– These characteristics include how easy the software is to use, how quickly it executes, how reliable it is, and how well it behaves when unexpected conditions arise.

– The nonfunctional requirements define these aspects about the system.

  - The nonfunctional requirements are sometimes referred to as "non-behavioral requirements" or "software quality attributes"

**More Explanation on Functional and Non-functional requirements**

These can take a number of forms such as traditional, basic form: textual, according to some template or standard form. It includes precise statement of one function the system carries out

Example: "The system must validate the patron ID number against the patron-database before allowing the patron to check out a book."

Product requirements specify the desired characteristics that a system or subsystem must possess.

☐ Requirements for critical systems are concerned with specifying constraints on the behaviour of the executing system. Such NFRs limit the freedom of the system designers and off-the-shelf-components. Some of the components can be formulated precisely and easily quantified with Performance and Reliability. Some are difficult to quantify in terms of Usability, Adaptability.

Non-functional (product) requirements play an important role for critical systems. Failure of such systems whose causes significant economic, physical or human damage to organizations or people. So here NFRs are discussed.

**Fig. 2. Non-Functional Requirements**

Exclusive for students by Prof. Anand Motwani



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

**IEEE Definition:**

"non functional requirement – in software system engineering, a software requirement that describes not what the software will do, but how the software will do it, for example, software performance requirements, software external interface requirements, design constraints, and software quality attributes. Nonfunctional requirements are difficult to test; therefore, they are usually evaluated subjectively."

There is no a clear distinction between functional and nonfunctional requirements. Whether or not a requirement is expressed as a functional or a non-functional requirement may depend: on the level of detail to be included in the requirements document and the comprehension of application domain and the desired system and also on experience of developers.

Non-functional requirements define the overall qualities or attributes of the resulting system

☐ Non-functional requirements place restrictions on the product being developed, the development process, and specify external constraints that the product must meet.

☐ Examples of NFR include safety, security, usability, reliability and performance requirements.

☐ Project management issues (costs, time, schedule) are often considered as non-functional requirements as well

Some properties of a system may be expressed either as a functional or non-functional property.

Example. The system shall ensure that data is protected from unauthorized access. Conventionally a non-functional requirement (security) because it does not specify specific system functionality.

When expressed as functional requirement: The system shall include a user authorization procedure where users must identify themselves using a login name and password. Only users who are authorized in this way may access the system data. Thus Non-functional requirements may result in new functional requirements statements.

Types of Non-functional requirements

The 'IEEE-Std 830 - 1993' lists 13 non-functional requirements to be included in a Software Requirements Document. Refer Figure 2.

**Classification of Non-functional requirements (Jacobson, 1999)**

The FURPS+ model is proposed for the Unified Process.

☐ Functional Requirements ☐ Usability ☐ Reliability ☐ Performance☐ Supportability

☐ The FURPS+ model provides additional requirements typically also included under the general label of non-functional requirements:

Exclusive for VIT Students by Prof. Anand Motwani



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

☐ Implementation requirements ☐ Interface requirements ☐ Operations requirements

☐ Packaging requirements ☐ Legal requirements

**Requirements Elicitation**

Requirements Elicitation is the process of discovering the requirements for a system by communication with customers, system users and others who have a stake in the system development. Development teams have to take the initiative.

**Requirement Sources and Elicitation Techniques**

**Elicitation Techniques**

| | |
|---|---|
| ➜ Traditional Approaches<br><br>Introspection<br><br>Interview/survey<br><br>Group elicitation | ➜ Observational approaches<br><br>Protocol analysis<br><br>Participant Observation (ethno methodology) |
| ➜ Model-based approaches<br><br>Goal-based: hierarchies of stakeholders' goals<br><br>Scenarios: characterizations of the ways in which the system is used<br><br>Use Cases: specific instances of interaction with the system | ➜ Exploratory approaches<br><br>Prototyping ("plan to throw one away") |

***Several Classifications are as Discussed in Lab and in Class.***

For more details refer Power point presentation.

**Use Cases**

- A *use case* is a description of a specific interaction that a user may have with the system. A collection of scenarios that describe the thread of usage of a system.
- Use cases are deceptively simple tools for describing the functionality of the software.

Exclusive for VIT Students by Prof. Anand Motwani



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

  - Use cases do not describe any internal workings of the software, nor do they explain how that software will be implemented.
  - They simply show how the steps that the user follows to use the software to do his work.
  - All of the ways that the users interact with the software can be described in this manner.

- Each scenario is described from the point-of-view of an "actor"—a person or device that interacts with the software in some way
- Each scenario answers the following questions:
  - What are the main tasks of functions performed by the actor?
  - What system information will the actor acquire, produce or change?
  - Will the actor inform the system about environmental changes?
  - What information does the actor require of the system?
  - Does the actor wish to be informed about unexpected changes

***Software Requirement Analysis***

requirements modelling will begin with scenario-based modelling, which results in creating a use case

- identify the "customer" and work together to negotiate "product-level" requirements
- build an analysis model
  - focus on data
  - define function
  - represent behavior
- prototype areas of uncertainty
- develop a specification that will guide design
- conduct formal technical reviews.
- **Analysis Process is depicted in Figure 3.**

**Davis Principle**

- Understand the problem before you begin to create the analysis model.

Exclusive for VIT Students by Prof. Anand Motwani



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

- Develop prototypes that enable a user to understand how human-machine interaction will occur.
- Record the origin of and the reason for every requirement.
- Use multiple views of requirements.
- Prioritize requirements.
- Work to eliminate ambiguity.

**Fig. 3. Analysis Process**

Exclusive for VIT Students by Prof. Anand Motwani



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

**Fig. 4. Analysis Model**

**Analysis Principle**

I. Model the Data Domain:
- define data objects
- describe data attributes
- establish data relationships

II. Model Function
- identify functions that transform data objects
- indicate how data flow through the system
- represent producers and consumers of data

III. Model Function
- indicate different states of the system
- specify events that cause the system to change state

IV. Partition the Models
- refine each model to represent lower levels of abstraction
  - refine data objects
  - create a functional hierarchy
  - represent behavior at different levels of detail

V. Essence
- begin by focusing on the essence of the problem without regard to implementation details

**Software Requirements Specification (SRS)**

The *software requirements specification* (SRS) represents a complete description of the behavior of the software to be developed.

- The SRS includes:
  - A set of use cases that describe all of the interactions that the users will have with the software.
  - All of the functional requirements necessary to define the internal workings of the software: calculations, technical details, data manipulation and processing, and other specific functionality that shows how the use cases are to be satisfied
  - Nonfunctional requirements, which impose constraints on the design or implementation (such as performance requirements, quality standards or design constraints).

Exclusive for VIT Students by Prof. Anand Motwani



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

**Example of System Requirement Specification**

**User Requirement Definition**

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

**System Requirements Specification**

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics shall be generated.
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed, and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g., 10 mg, 20 mg) separate reports shall be created for each dose unit.
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

**Use case Modeling**

**Ivar Jackobson has been credited with inventing use cases which appeared in object oriented community somewhere in 1992.**

A Use Case describes a function (behaviour) provided by the system in terms of flowing events, including interaction with actors. It is initiated by an actor. Each use case has a name with entry and termination condition. Graphical Notation: An oval with the name of the use case

**Use Case Model:** The set of all use cases specifying the complete functionality of the system.

Exclusive for VIT Students



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

Users, Roles and which user can invoke which use case are shown in figure.

A use case model consists of use cases and use case associations

☐ A use case association is a relationship between use cases.

The important types of use case associations: Include, Extends, Generalization

♦ Include: A use case uses another use case ("functional decomposition").

♦ Extends: A use case extends another use case.

♦ Generalization: An abstract use case has different specializations.

Problem: A function in the original problem statement is too complex to be solvable immediately or the function already exists.

♦ Solution: Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into smaller use cases.

e for VIT Students by Prof. Anand Motwani



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

- <<include>> relationship

*Flow of Events*

...

3. **FieldOfficer** completes and submitts the form
4. SYSTEM receives the form, evaluates the information (by using Use Case *EvaluateInfo)* being received and sends an acknowledge

- <<extend>> relationship

*Entry Condition* — The *ConnectionDown* use case extends *any use* case in which the connection between the **FieldOfficer** and the SYSTEM can be lost

*Flow of Events*

1. The FieldOfficer is notified that the connection is broken

**Dynamic Modeling with UML**

Modeling of behavioral aspects among the participating objects is Dynamic Modeling. It shows how the behavior of a use case (or scenario) is distributed among its participating objects. In it we Tie use cases with objects. Identify new objects and classes. It helps to determine how decentralized the system is. Its purpose is to Detect and supply methods for the object model.

♦ How do we do this?

☐ Start with use case or scenario

☐ Model interaction between objects: (three types of objects i.e. Entity, boundary and control) and draw sequence diagram.

☐ Model dynamic behavior of a single object: State chart diagram.

**Sequence Diagram** can be read out of use cases. It is Dynamic behavior of a set of objects arranged in time sequence and good for real-time specifications and complex scenarios. It tells the Information of the life time of objects and Relationships between objects.

♦ State Chart Diagram describes the dynamic behavior of a single object. A state machine that describes the response of an object of a given class to the receipt of outside stimuli (Events).

Exclusive for VIT Students by Prof. Anand Motwani

**2.5 System and Software Requirement Specifications**

The goal of analysis is to discover problems, incompleteness and inconsistencies in the elicited requirements. These are then fed back to the stakeholders to resolve them through the negotiation process. Analysis is interleaved with elicitation as problems are discovered when the



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

requirements are elicited. A problem checklist may be used to support analysis. Each requirement may be assessed against the checklist.

**Software (System) Requirement Analysis (Summary)**

1. What are the functions? — **Functional Modeling**
   - Create scenarios and use case diagrams
     - Talk to client, observe, get historical records, do thought experiments
2. What is the structure of the system? Create class diagrams — **Object Modeling**
   - Identify objects.
   - What are the associations between them? What is their multiplicity?
   - What are the attributes of the objects?
   - What operations are defined on the objects?
3. What is its behavior? — **Dynamic Modeling**
   - Create sequence diagrams
     - Identify senders and receivers
     - Show sequence of events exchanged between objects. Identify event dependencies and event concurrency.
   - Create state diagrams
     - Only for the dynamically interesting objects.

The above activities results in the Requirement Specification document.

**Example Use Case:**

The **Document Management System (DMS)** use case diagram example below shows the actors and use cases of the system. In particular, there are include and extend relationships among use cases.

Must refer to learn more about writing effective use-cases:
https://www.visual-paradigm.com/tutorials/writingeffectiveusecase.jsp



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

Exclusive for students by Prof. Anand Motwani

**Requirement Validation**

Requirements validation is a critical step in the development process, usually during requirements engineering or requirements analysis and also at delivery (client acceptance test) of software.

♦ **Requirements validation criteria:**

- ✓ Complete: All possible scenarios, in which the system can be used, are described, including exceptional behavior by the user or the system.
- ✓ Consistent: There are no two functional or nonfunctional requirements that contradict each other.
- ✓ Unambiguous: Requirements can not be interpreted in mutually exclusive ways
- ✓ Correct: The requirements represent the client's view
- ✓ More Requirements validation criteria
- ✓ Realistic: Requirements can be implemented and delivered
- ✓ Verifiable: Requirements can be checked. It needs an exact description of the requirements

**Problem with requirements validation:** Requirements change very fast during requirements elicitation.

♦ Tool support for managing requirements:

- Store requirements in a shared repository
- Provide multi-user access
- Automatically create a system specification document from the repository
- Allow for change management
- Provide traceability throughout the project lifecycle

| **Analysis vs. Validation Differentiation** | |
|---|---|
| Analysis (Verification) works with raw requirements as elicited from the system stakeholders | Validation works with a final draft of the requirements document i.e. with negotiated and agreed requirements |
| Usually incomplete and expressed in an informal and unstructured way | All known incompleteness and inconsistency are removed |
| Incorporation of stakeholder is important | Validation process is more concerned with the way in which the requirements are described |
| This tests "Have we got the right requirements" | This tests "Have we got the requirements right" |

Exclusive for VIT Students by Prof. Anand Motwani



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

**Validation Inputs and Outputs**

---

**Notes on following topics will be provided into Part – II of this UNIT.**

Refer PPT.

- *Data Modeling Concepts - Class Based Modeling - Flow Oriented Modeling – Creating Behavioral Modeling - Patterns For Modeling.*

Exclusive for VIT Students by Prof. Anand Motwani



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