G.PULLAIAH COLLEGE OF ENGINEERING AND TECHNOLOGY, Kurnool
Department of Computer Science and Engineering
Software Engineering
TOPICS TO REMEMBER
UNIT – 1

SOFTWARE AND SOFTWARE ENGINEERING

1.1 THE NATURE OF SOFTWARE

1.1.1 Defining Software

1. Software is developed or engineered
2. Software doesn't "wear out.":
3. Although the industry is moving toward component-based construction, software continues to be custom built:

1.1.2 Software Application Domains

System software
Application softwareEngineering/scientific software
Embedded software
Product-line softwareWeb applications
Artificial intelligence software
New Challenges
Open-world computing
Net sourcing
Open source

1.1.3 Legacy Software

1.2 THE UNIQUE NATURE OF WEBAPPS

Network intensiveness
Concurrency. Unpredictable load
Performance. Availability
Data driven. Content sensitive
Continuous evolution
Immediacy
Security
Aesthetics

1.3 SOFTWARE ENGINEERING

Layered Technology

1.4 The software Process

Communication. Planning.
ModelingConstruction..
Deployment.
Software project tracking and control
Risk management
Software quality assuranceTechnical reviews
Measurement
Software configuration management
Reusability management.
Work product preparation and production

1.5 Software Engineering Practice

1.5.1 Introduction

Understand the problem.
Plan the solution
Plan the solution.
Examine the result.

1.5.2 General Principles

The First Principle: The Reason It All Exists
The Second Principle: KISS (Keep It Simple, Stupid!)
The Third Principle: Maintain the Vision
The Fourth Principle: What You Produce, Others Will Consume
The Fifth Principle: Be Open to the Future
The Sixth Principle: Plan Ahead for Reuse
The Seventh principle: Think!

1.6 Software myths

Management myths.
Customer myths
Practitioner's myths.

1.7 PROCESS MODELS

1.7 A GENERIC PROCESS MODEL:
1.7.1 Defining a Framework Activity
1.7.2 Identifying a Task Set
1.7.3 Process Patterns

1.8 PROCESS ASSESSMENT AND IMPROVEMENT

Standard CMMI Assessment Method for Process Improvement (SCAMPI)
CMM-Based Appraisal for Internal Process Improvement (CBA IPI)
SPICE (ISO/IEC15504
ISO 9001:2000 for Software

1.9 PRESCRIPTIVE PROCESS MODELS

1.9.1 The Waterfall Model
1.9.2 Incremental Process Models
1.9.3 Evolutionary Process Models
Prototyping
The Spiral Model
1.9.4 Concurrent Models

1.10 SPECIALIZED PROCESS MODELS

1.10.1 Component-Based Development
1.10.2 The Formal Methods Model
1.10.3 Aspect-Oriented Software Development

1.11 THE UNIFIED PROCESS

1.11.1 Introduction
1.11.2 Phases of the Unified Process

1.12 PERSONAL AND TEAM PROCESS MODELS

1.12.1 Introduction
1.12.2 Personal Software Process (PSP)
Planning.
High-level design
High-level design review
Development, Postmortem
1.12.3 Team Software Process (TSP)

1.13 PROCESS TECHNOLOGY

1.14 PRODUCT AND PROCESS

People

Process

Product:

AGILITY

1.15 WHAT IS AGILITY?

1.16 AGILITY AND THE COST OF CHANGE

1.17 WHAT IS AN AGILE PROCESS?

1.17.1 Agility Principles

1.17.2 The Politics of Agile Development

1.17.3 Human Factors

Competence

Common focus

Collaboration:

Decision-making ability

Fuzzy problem-solving ability

Fuzzy problem-solving ability

Self-organization

1.18 EXTREME PROGRAMMING (XP)

1.18.1 XP Values

1.18.2 XP Process

1.18.3 Industrial XP

Readiness assessment

Project community

Project chartering

Test-driven management

Retrospectives

1.18.4 The XP Debate

- Requirements volatility.
- Conflicting customer needs.
- Requirements are expressed informally.
- Lack of formal design.

1.19 OTHER AGILE PROCESS MODELS

1.19.1 Adaptive Software Development (ASD)

1.19.2 Scrum

1.19.3 Dynamic Systems Development Method (DSDM)

1.19.4 Crystal

1.19.5 Feature Driven Development (FDD):

1.19.6 Lean Software Development (LSD

1.19.7 Agile Modeling (AM):

1.19.8 Agile Unified Process (AUP)

- Modeling
- Implementation
- Testing
- Deployment
- Configuration and project management
- Environment management

UNIT – 2

2.0 UNDERSTANDING REQUIREMENTS

2.1 REQUIREMENTS ENGINEERING

a) Inception.
b)Elicitation.
c)Elaboration.
d)Negotiation.
e)Specification.

2.2 ESTABLISHING THE GROUNDWORK

2.2.1 Identifying Stakeholders
2.2.2 Recognizing Multiple Viewpoints
2.2.3 Working toward Collaboration
2.2.4 Asking the First Questions

2.3 ELICITING REQUIREMENTS

2.3.1 Collaborative Requirements Gathering:
2.3.2 Quality Function Deployment:
Normal requirements.
Expected requirements.
Exciting requirements
2.3.3 Usage Scenarios:
2.3.4 Elicitation Work Products:

2.4 DEVELOPING USE CASES

2.5 BUILDING THE REQUIREMENTS MODEL

2.5.1 Elements of the Requirements Model
Scenario-based elements.
Class-based elements
Behavioral elements.
Flow-oriented elements.
2.5.2 Analysis Patterns

2.6 NEGOTIATING REQUIREMENTS

2.7 VALIDATING REQUIREMENTS

2.8 Requirements Modeling (Scenarios, Information and Analysis Classes)

2.8 REQUIREMENTS ANALYSIS

2.8.1.1 Overall Objectives and Philosophy:
2.8.1.2 Analysis Rules of Thumb
2.8.1.3 Domain Analysis
2.8.1.4 Requirements Modeling Approaches:

2.8.2 SCENARIO-BASED MODELING

2.8.2.1 Creating a Preliminary Use Case
2.8.2.2 Refining a Preliminary Use Case
2.8.2.3 Writing a Formal Use Case

2.8.3 UML MODELS THAT SUPPLEMENT THE USE CASE

2.8.3.1 Developing an Activity Diagram:
2.8.3.2 Swimlane Diagrams

2.8.4 DATA MODELING CONCEPTS:

2.8.4.1 Data Objects
2.8.4.2 Data Attributes
2.8.4.3 Relationships

UNIT – 3

3.14.4 Classifying and Retrieving Components

UNIT – 4

4.0 User Interface Design

4.1 The Golden Rules

4.1.1 Place The User In Control

4.1.2 Reduce The User's Memory Load

4.1.3 Make The Interface Consistent

4.2 User Interface Analysis And Design

4.2.1 Interface Analysis And Design Models

4.2.2 The Process

4.3 Interface Analysis

4.3.1 User Analysis

4.3.2 Task Analysis And Modeling

4.3.3 Analysis Of Display Content

4.3.4 Analysis Of The Work Environment

4.4 Interface Design Steps

4.4.1 Applying Interface Design Steps

4.4.2 User Interface Design Patterns

4.4.3 Design Issues

4.5 Webapp Interface Design

4.5.1 Interface Design Principles And Guidelines

4.5.2 Interface Design Workflow For Webapps

4.6 Design Evaluation

4.7 Webapp Design

4.7.1 Web App Design Quality:

4.7.2 Design Goals:

4.7.3 A Design Pyramid For Webapps:

4.7.4 Webapp Interface Design

4.7.5.1 Layout Issues

4.7.5.2 Graphic Design Issues

4.8 Content Design

4,8.1 Content Objects

4.8.2 Content Design Issues

4.9 Architecture Design

4.9.1 Content Architecture

4.9.2 Webapp Architecture

4.10 Navigation Design

4.10.1 Navigation Semantics

4.10.2 Navigation Syntax

4.11 Component-Level Design

4.12 Object-Oriented Hypermedia Design Method (OOHDM)

4.12.1 Conceptual Design For OOHDM:

4.12.2 Navigational Design For OOHDM:

4.12.3 Abstract Interface Design And Implementation:

UNIT – 5

5.0 Software Testing
5.1 A Strategic Approach To Software Testing
- 5.1.1 Verification and Validation
- 5.1.2 Organizing for Software Testing
- 5.1.3 Software Testing Strategy—The Big Picture
- 5.1.4 Criteria for Completion of Testing

5.2 Strategic Issues
5.3 Test Strategies For Conventional Software
- 5.3.1 Unit Testing
- 5.3.2 Integration Testing
  - Top-down integration
  - Bottom-up integration
  - Regression testing
  - Smoke testing
  - Strategic options
  - Integration test work products

5.4 Test Strategies For Object-Oriented Software
- 5.4.1 Unit Testing in the OO Context
- 5.4.2 Integration Testing in the OO Context

5.5 Test Strategies For Webapps
5.6 Validation Testing
- 5.6.1 Validation-Test Criteria
- 5.6.2 Configuration Review
- 5.6.3 Alpha and Beta Testing

5.7 SYSTEM TESTING
- 5.7.1 Recovery Testing
- 5.7.2 Security Testing
- 5.7.3 Stress Testing
- 5.7.4 Performance Testing
- 5.7.5 Deployment Testing

5.8 THE ART OF DEBUGGING
- 5.8.1 The Debugging Process
- 5.8.2 Psychological Considerations
- 5.8.3 Debugging Strategies - Automated debugging
- 5.8.4 Correcting the Error

5.9 Testing Conventional Applications
- 5.9.1 Software Testing Fundamentals
- 5.9.2 Internal And External Views Of Testing
- 5.9.3 White-Box Testing
- 5.9.4 Basis Path Testing
  - 5.9.4.1 Flow Graph Notation
  - 5.9.4.2 Independent Program Paths
  - 5.9.4.3 Deriving Test Cases
    1. Using the design as foundation, draw a corresponding flow graph
    2. Determine the cyclomatic complexity of the resultant flow graph.
    3. Determine a basis set of linearly independent paths
    4. Prepare test cases that will force execution of each path in the basis set
  - 5.9.4.4 Graph Matrices

5.9.5 CONTROL STRUCTURE TESTING

5.9.5.1 Condition Testing

5.9.5.2 Data Flow Testing

5.9.5.3 Loop Testing:

Nested loops

Concatenated loops

Unstructured loops

5.9.6 BLACK-BOX TESTING

5.9.6.1 Graph-Based Testing Methods

Transaction flow modeling

Finite state modeling

Data flow modeling

Timing modeling

5.9.6.2 Equivalence Partitioning

5.9.6.3 Boundary Value Analysis

5.9.6.4 Orthogonal Array Testing

Detect and isolate all single mode faults

Detect all double mode faults

Multimode faults

5.9.7 MODEL-BASED TESTING

1. Analyze an existing behavioral model for the software or create one
2. Traverse the behavioral model and specify the inputs that will force the software to make the transition from state to state.
3. Review the behavioral model and note the expected outputs as the software makes the transition from state to state
4. Execute the test cases
5. Compare actual and expected results and take corrective action as required

5.10 TESTING FOR SPECIALIZED ENVIRONMENTS, ARCHITECTURES, APPLICATIONS

5.10.1 Testing GUIs

5.10.2 Testing of Client-Server Architectures

- Application function tests
- Server tests
- Database tests
- Transaction tests
- Network communication tests

5.10.3 Testing Documentation and Help Facilities

5.10.4 Testing for Real-Time Systems

- Task testing
- Behavioral testing .
- Intertask testing
- System testing

5.11 PATTERNS FOR SOFTWARE TESTING

5.12 TESTING OBJECT ORIENTED APPLICATIONS

5.12.1.BROADENING THE VIEW OF TESTING

5.12.2 TESTING OOA AND OOD MODELS

5.12.2.1 Correctness of OOA and OOD Models

5.12.2.2 Consistency of Object-Oriented Models

1. Revisit the CRC model and the object-relationship model
2. Inspect the description of each CRC index card to determine if a delegated responsibility is part of the collaborator's definition

3. Invert the connection to ensure that each collaborator that is asked for service is receiving requests from a reasonable source
4. Using the inverted connections examined in step 3, determine whether other classes might be required or whether responsibilities are properly grouped among the classes.
5. Determine whether widely requested responsibilities might be combined into a single responsibility

5.12.3 OBJECT-ORIENTED TESTING STRATEGIES

- 5.12.3.1 Unit Testing in the OO Context
- 5.12.3.2 Integration Testing in the OO Context
- 5.12.3.3 Validation Testing in an OO Context

5.12.4 OBJECT-ORIENTED TESTING METHODS

- 5.12.4.1 The Test-Case Design Implications of OO
- 5.12.4.2 Applicability of Conventional Test-Case Design Methods
- 5.12.4.3 Fault-Based Testing
- 5.12.4.4 Test Cases and the Class Hierarchy
- 5.12.4.5 Scenario-Based Test Design
- 5.12.4.6 Testing Surface Structure and Deep

5.12.5 TESTING METHODS APPLICABLE AT THE CLASS LEVEL

- 5.12.5.1 Random Testing for OO Classes
- 5.12.5.2 Partition Testing at the Class Level

5.13 INTERCLASS TEST-CASE DESIGN

- 5.13.1 Multiple Class Testing
- 5.13.2 Tests Derived from Behavior