“Heaven’s Light is Our Guide”

Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

Assignment

Course Code: CSE 3205

Course Title: Software Engineering

<u>Submitted By-</u>	**<u>Submitted To-</u>**
Name: Sajidur Rahman Tarafder	**Farjana Parvin**
Department: CSE	**Lecturer**
Roll No: 2003154	**Department of CSE, RUET**
Section: C	
Session: 2020-21	

Assignment-1

Comparative Analysis of Software Process Models

Software process models are structured approaches that define the sequence of activities, tasks, and milestones involved in the development of software systems. These models provide a framework for planning, organizing, and controlling the process of building software, ensuring quality and efficiency. The choice of a suitable software process model depends on project requirements, team expertise, customer involvement, and the likelihood of changes during development.

Selected Software Process Models:

- Waterfall Model
- Agile Model
- Spiral Model

Waterfall Model:

Description:

The Waterfall Model is a linear and sequential approach where each phase (requirements, design, implementation, testing, deployment, and maintenance) must be completed before the next begins. It is best suited for projects with well-defined requirements and minimal expected changes.

Advantages:

- Simple and easy to understand.
- Well-suited for projects with clear, fixed requirements.
- Easy to manage due to its rigidity.

Disadvantages:

- Inflexible to changes once the process is underway.
- Poor model for complex and object-oriented projects.
- Difficult to accommodate changes after the process has started.

Real-world Application:

Best used in projects with clear, unchanging requirements, such as government systems, military software, or manufacturing process automation.

Agile Model:

Description:

The Agile Model is an iterative and incremental approach that emphasizes flexibility, customer collaboration, and rapid delivery. Development is divided into small cycles (sprints), allowing for frequent reassessment and adaptation to changing requirements.

Advantages:

- Flexible and adaptable to changing requirements.
- Promotes customer involvement and feedback.
- Faster delivery of functional software.

Disadvantages:

- Less predictable due to evolving requirements.
- Requires experienced team members.
- Documentation can be neglected.

Real-world Applications:

Ideal for web-based applications, mobile apps, start-up software, and any project with evolving requirements.

Spiral Model:

Description:

The Spiral Model combines iterative development with systematic risk analysis. Each cycle (spiral) involves planning, risk assessment, engineering, and evaluation, making it suitable for large, complex, and high-risk projects where requirements may evolve over time.

Advantages:

- Good for large, complex, and high-risk projects.
- Emphasizes risk analysis and mitigation.
- Allows for iterative refinement.

Disadvantages:

- Can be expensive and time-consuming.
- Requires expertise in risk assessment.
- Not suitable for small projects.

Real-world Applications:

Used in large-scale software projects like banking systems, aerospace, and mission-critical applications.

Comparison Table:

Model	Advantages	Disadvantages	Best Suited For
Waterfall	Simple, structured	Inflexible, poor for changes	Fixed, clear requirements
Agile	Flexible, customer-focused	Less predictable, needs expertise	Evolving requirements
Spiral	Risk management, iterative	Costly, complex	Large, high-risk projects

Assignment-2

Cost Estimation in Software Engineering

Introduction of Cost Estimation Models:

Cost estimation in software engineering is the process of predicting the amount of effort, time, and resources required to develop a software system. Accurate cost estimation is crucial for project planning, budgeting, and successful delivery. Several models and techniques have been developed to assist in estimating software costs. The most common cost estimation models include:

COCOMO (Constructive Cost Model):

COCOMO is a widely used algorithmic model that estimates the effort, time, and cost required for software development based on the size of the project (measured in thousands of lines of code, KLOC) and other factors such as complexity, team experience, and tools used. It provides different levels of estimation (Basic, Intermediate, and Detailed) to suit various project needs.

Function Point Analysis (FPA):

Function Point Analysis estimates the size and complexity of a software project by quantifying its functional components, such as inputs, outputs, user interactions, files, and interfaces. The function points are then used to estimate the required effort and cost, making this model independent of programming language and technology.

Sample Project Description:

- A web-based library management system.
- Estimated 2,000 delivered source instructions (DSI).
- Development team size: 5 developers.
- Development duration: 3 months.

COCOMO Effort Estimation:

For the Organic Model, the effort (E) and development time (D) are estimated as:

- **E = a × (KLOC)^b**
- **D = c × (E)^d**

Using typical constants for Organic Model:

- a = 2.4, b = 1.05, c = 2.5, d = 0.38

Where KLOC = 2 (since 2000 LOC)
Effort (E): E = 2.4 × (2)^1.05 ≈ 4.93 person-months
Development Time (D): D = 2.5 × (4.93)^0.38 ≈ 3.01 months

Estimated Cost Breakdown:

Assuming average developer cost = $2,000/month

Component	Amount
Effort (Person-Months)	~5
Time (Months)	~3
Total Cost (Estimated)	5 × $2,000 = $10,000

Actual Project Outcome:

Component	Actual Value
Actual Effort	6 person-months
Actual Time	3.5 months
Actual Cost	6 × \$2,000 = \$12,000

Estimated vs. Actual Cost Breakdown:

Metric	Estimated	Actual	Difference
Person-Months	5	6	+1
Time (Months)	3	3.5	+0.5
Cost (USD)	\$10,000	\$12,000	+\$2,000

Analysis:

The estimated cost was lower than the actual cost by \$2,000 due to:

- Additional feature requests mid-project.
- Slight delay in integration and testing.
- Underestimated complexity of database module.