Software Construction

Talha Arif Wains
Department of Computer Science
Quaid-i-Azam University, Islamabad

July 2025

1 Process Framework

A **process framework** is a plan that outlines the steps to follow in order to build software effectively.

- Planning
- Designing
- Coding
- Testing

2 The 4 P's of Software Construction

- **People** – Everyone involved (e.g., developers, users, managers)
- **Product** – The software being built
- **Process** – The steps to build the software
- **Project** – All the work needed to deliver the product

3 Umbrella Activities

Umbrella activities are ongoing support tasks like **quality checks**, **documentation**, **reviews**, and **project tracking** that happen alongside the main development work.

4 Software Scope

Software scope defines:

- The **functions** that are to be delivered to end-users
- The **data** that are input and output
- The **project constraints** that bound the system

5 Earned Value Analysis

- **The Budgeted Cost of Work Scheduled (BCWS)**: BCWS_i is the effort planned for work task i.
- **Budget at Completion (BAC)**: Total planned effort, $\text{BAC} = \sum(\text{BCWS}_k)$ for all tasks k.
- **The Budgeted Cost of Work Performed (BCWP)**: Value of work actually completed.
- **Schedule Performance Index (SPI)**: $\text{SPI} = \frac{\text{BCWP}}{\text{BCWS}}$
  If SPI less than 1 $\rightarrow$ behind schedule.
- **Schedule Variance (SV)**: $\text{SV} = \text{BCWP} - \text{BCWS}$
  Positive = ahead, Negative = behind schedule.
- **The Actual Cost of Work Performed (ACWP)**: Actual cost incurred for the work completed.
- **Cost Performance Index (CPI)**: $\text{CPI} = \frac{\text{BCWP}}{\text{ACWP}}$
  If CPI less than 1 $\rightarrow$ over budget.
- **Cost Variance (CV)**: $\text{CV} = \text{BCWP} - \text{ACWP}$
  Positive = under budget, Negative = over budget.

6 The "First Law" of System Engineering

"No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle."

7 Software Configuration

Software configuration is the complete set of **programs**, **documents**, and **data** developed and maintained during a software project.

8 Software Configuration Management (SCM)

SCM is the process of managing changes in software components (**code**, **documents**, **data**) to ensure the project stays organized, consistent, and under control.

Key Activities of SCM

- Identify change
- Control change
- Ensure change is properly implemented

- Report change to others

Note: These activities are not the same as general maintenance.

9 Baseline

A **baseline** is a checkpoint or approved version of software components (like code, documents, or models) after review, that is used in work in future.

10 SCM Repository

An **SCM repository** is a storage system where all important software components (**code**, **documents**, **data**) are saved, tracked, and managed.

11 Version Control

Version control is the process of managing changes to software files by tracking different versions over time.

Main Features

- **Project database (repository)** – Stores all items
- **Version management** – Tracks every version
- **Make facility** – Builds specific software versions
- **Issue tracking** – Records and follows up on bugs or tasks

12 FURPS+ Model in the Unified Process (UP)

In the Unified Process (UP), requirements are categorized using the **FURPS+** model:

FURPS

- **F – Functional:** What the system should do (features, security).
- **U – Usability:** Ease of use, user help, and documentation.
- **R – Reliability:** System stability, error handling, and predictability.
- **P – Performance:** Speed, accuracy, resource use, uptime.
- **S – Supportability:** Ease of maintenance, updates, and adaptability.

“+” (Extra Concerns)

- **Implementation:** Tools, platforms, constraints.
- **Interface:** Integration with other systems.
- **Operations:** system management.
- **Packaging:** How it’s delivered.
- **Legal:** Licensing and compliance.

Simplified Categories

- **Functional Requirements** → What the system does (behavior).
- **Non-Functional Requirements** → How well it does it (quality attributes like usability, performance, etc.).

13 Data Dictionary

A **data dictionary** is a list of all the data used in a system, along with details about each item such as name, type, format, and description.
Example:

- **Username** – String
- **DOB** – format(type, length, unit)
- **Balance**

14 Software Architecture

Software architecture is the overall structure of a software system, showing how parts are organized, how they work together, and how the system is built using rules and patterns.

15 Low Coupling and High Cohesion

Low Coupling: Parts of the system should depend as little on each other as possible.
High Cohesion: Each part should focus on a single task or responsibility.
These principles apply to both small components (objects) and large components (subsystems, applications).

16 The Layers Pattern (Solution to High Coupling)

To reduce complexity and improve flexibility, use layers in your system:

- **User Interface Layer** – Handles input/output, UI.

- **Application/Domain Layer** – Core business logic.
- **Technical Services Layer** – Handles things like databases, logging, or networks.

Each layer has its own responsibility and communicates only with the layer directly below it.

17 Benefits of Using Layers

- Clean separation of responsibilities.
- Easier to scale, maintain, and update.
- Developers can work on different parts independently.