Project Management (CS604) - Assignment 3

Rustam Ji Institute of Technology, Tekanpur
Department of Computer Science & Engineering
Semester – 6
Session: 2025-26

Assignment 3: Unit 2

1. Different Layers of Software Engineering

Software Engineering is a multi-layered technology, often conceptualized as layers that build upon each other, with a quality focus as the bedrock.

1. **Tools (or Tools & Methods):** The outermost layer. It encompasses the automated or semi-automated tools that support the processes and methods. This includes Integrated Development Environments (IDEs) like VS Code, configuration management systems (e.g., Git), Continuous Integration/Continuous Deployment (CI/CD) pipelines, testing frameworks (e.g., JUnit, Selenium), and project management tools (e.g., Jira, Trello). Tools provide the infrastructure to apply methods efficiently and consistently.

2. **Methods:** The next layer. Methods define the "how-to" of software construction. They provide technical approaches for performing various software engineering activities. This includes:

   - Communication methods (e.g., user stories, use cases)
   - Requirements analysis methods (e.g., requirements workshops, prototyping)
   - Design methods (e.g., Object-Oriented Design with UML, structured design)
   - Coding standards and practices (e.g., pair programming, code reviews)
   - Testing strategies (e.g., test-driven development, black-box testing)
   - Quality assurance techniques (e.g., formal inspections)

3. **Process:** The foundation layer upon which methods are applied. The process is the "glue" that holds the technology layers together. It defines the framework of activities—a logical sequence of steps—required to deliver a software system. It defines when and how methods are applied, what artifacts are produced, and how the project is managed. Examples include the Waterfall model, Spiral model, Rational Unified Process (RUP), and various Agile frameworks (Scrum, Kanban).

4. **A Quality Focus:** This is the bedrock, or the innermost layer, that permeates all other layers. It represents the organization's commitment to continuous improvement and quality. It includes:

    - A culture of quality that values defect prevention over defect detection.
    - Process improvement initiatives (e.g., using the Capability Maturity Model Integration - CMMI).
    - The ethical and professional responsibility to deliver high-quality, reliable, and maintainable software.

2. Software Process Framework and its Activities

A **Software Process Framework** establishes the foundation for a complete software process by identifying a small number of **framework activities** that are applicable to all software projects, regardless of their size or complexity. These activities are complemented by a set of **umbrella activities** (e.g., project management, configuration management, quality assurance) that run across the entire project.

The five generic framework activities are:

1. **Communication:** The crucial first activity focused on understanding stakeholder objectives and requirements. It involves heavy interaction with customers, users, and other stakeholders to:

    - Define the project scope and objectives.
    - Identify features and functionalities.
    - Capture functional and non-functional requirements.
    - Establish the business case.

2. **Planning:** The activity that establishes a roadmap for the project. It involves:

    - Creating the project plan.
    - Defining the project scope in detail.
    - Estimating effort, cost, and risk.
    - Scheduling tasks and defining resource requirements.
    - Establishing a Work Breakdown Structure (WBS).

    It answers the questions: "What will we build? How will we build it? Who will be involved?"

3. **Modeling:** The activity that involves creating representations of the software to be built. It helps stakeholders and developers better understand the requirements and the design. This includes:

    - **Requirements Modeling:** Creating models that depict user scenarios, system functions, and data flow (e.g., use-case diagrams, user stories).

- **Design Modeling:** Creating models that describe the software architecture, components, interfaces, and data structures (e.g., class diagrams, sequence diagrams, architectural patterns).

4. **Construction:** The activity where the actual executable software is built. This is a combination of:
   - **Coding:** Writing source code based on design models, following coding standards.
   - **Testing:** Performing unit tests, integration tests, and system tests to uncover and fix defects.
5. **Deployment:** The activity where the software is delivered to the customer for evaluation and use. This includes:
   - Deployment to production environments.
   - User training and documentation.
   - Transition to a support and maintenance phase.
   - Gathering feedback for future iterations or releases.

3. Project Life Cycle Phases

A project life cycle is a collection of project phases that a project passes through from its start to its completion. While phases vary by process model, a generic, phase-based view is essential for understanding project governance.

1. **Conception and Initiation:** The idea is conceived. This phase involves:
   - Feasibility analysis (technical, economic, operational).
   - Defining the initial business case and high-level scope.
   - Identifying key stakeholders.
   - Securing initial approval and funding to proceed.
2. **Definition and Planning:** The project is defined in detail. This phase involves:
   - Refining and baselining the scope.
   - Gathering and documenting detailed requirements.
   - Creating the Work Breakdown Structure (WBS).
   - Developing a detailed schedule and budget.
   - Finalizing estimates.
   - Identifying, analyzing, and planning for risks.
   - Establishing the project management plan.
3. **Launch and Execution:** The actual work of building the product is performed. This phase consumes the majority of the project budget and effort. It involves:

- Executing the framework activities of modeling and construction.
- Following the project management plan.
- Managing the team and resources.
- Producing the software deliverables.

4. **Performance and Control:** This is an "umbrella" phase that runs concurrently with Execution. It involves:
   - Monitoring progress against the plan (schedule, budget, quality).
   - Managing changes to scope, schedule, and budget through a formal change control process.
   - Tracking and mitigating risks.
   - Reporting performance to stakeholders.
   - Taking corrective actions when variances are detected.
5. **Closure:** The project is formally completed. This includes:
   - Finalizing all activities and deliverables.
   - Delivering the final product and obtaining formal acceptance.
   - Releasing project resources.
   - Conducting post-project reviews and retrospectives.
   - Archiving project documentation and lessons learned.

4. Steps of the Project Planning Phase

Project planning is the most critical phase for project success. It involves a systematic sequence of steps to define the "what, how, who, and when":

1. **Establish Project Scope:** Define clear boundaries—what is included and, critically, what is *not* included. This is often documented in a Vision and Scope document. A clear scope prevents scope creep later.
2. **Create Work Breakdown Structure (WBS):** Decompose the project scope into manageable, trackable tasks and deliverables. The WBS is the foundational input for most planning and control activities. It should be a deliverable-oriented hierarchy.
3. **Estimate Effort, Cost, and Duration:** Use models (e.g., COCOMO, FP), historical data, and expert judgment to estimate the resources, time, and cost required for each WBS element.
4. **Develop Schedule:** Sequence the activities defined in the WBS, define dependencies (e.g., finish-to-start), allocate resources, and assign start and finish dates to create a project schedule. Common tools include Gantt charts and network diagrams (e.g., CPM).

5. **Identify and Plan for Risks:** Perform risk identification (brainstorming, checklists), analysis (probability and impact assessment), and develop mitigation (preventive) and contingency (reactive) plans for high-priority risks. Document these in a risk register.

6. **Develop Quality Plan:** Define the quality standards (e.g., ISO 9001, internal standards), processes for reviews, audits, and inspections, and metrics to be used for quality assurance and control.

7. **Plan Communication:** Define the communication needs of all stakeholders. Outline what information will be communicated, the frequency, the format (e.g., status reports, dashboards), and the channels (e.g., email, meetings).

8. **Staff the Project:** Identify and assign the required personnel based on the skills needed for each task. This includes defining roles, responsibilities, and reporting structures.

9. **Establish Baseline:** Formally approve the project plan. The schedule, budget, and scope are now "baselined." This baseline is the reference point against which all future performance and changes will be measured.

5. Steps of the Project Execution Phase

The Execution phase is where the plan is put into action to build the product. Its key steps are iterative and ongoing:

1. **Perform Work According to Plan:** The team executes the tasks as defined in the schedule, following the prescribed processes and methods.

2. **Implement Processes:** Adhere to the defined communication, quality, and risk management processes. This involves holding scheduled meetings, conducting formal reviews, and following coding and testing standards.

3. **Manage Teams:** Foster a collaborative environment, resolve conflicts, provide coaching and leadership, and ensure the team has the necessary resources and a conducive working environment.

4. **Create Deliverables:** Produce the work products defined in the project plan, including requirements documents, design models, source code, test cases, user manuals, and other artifacts.

5. **Perform Quality Assurance (QA):** Execute the quality plan by conducting process audits, formal reviews (e.g., design reviews), and inspections to ensure processes are being followed and quality standards are met.

6. **Manage Changes:** Process change requests from stakeholders. Evaluate their impact on scope, schedule, and budget. Approve or reject them through a formal change control board (CCB) process to maintain baseline integrity.

7. **Manage Risks:** Continuously monitor existing risks, implement mitigation strategies, track contingency triggers, and actively identify new risks as the project unfolds. Update the risk register regularly.

8. **Manage Communication:** Distribute performance reports, hold status meetings, manage stakeholder expectations, and ensure information flows efficiently to all relevant parties.

6. Activities of Monitoring and Control Project Work

Monitoring and Control is the parallel process to Execution, ensuring the project stays on track. It is a feedback loop that enables corrective action.

1. **Measure Performance:** Collect data on actual effort, cost, schedule progress, and quality metrics (e.g., number of defects found, test cases passed).

2. **Variance Analysis:** Compare actual performance against the baseline plan. Key tools include:

   - **Earned Value Management (EVM):** Calculates:
     - **Schedule Variance (SV):** EV - PV (Earned Value - Planned Value)
     - **Cost Variance (CV):** EV - AC (Earned Value - Actual Cost)
     - **Schedule Performance Index (SPI):** EV / PV (values ¡1 indicate behind schedule)
     - **Cost Performance Index (CPI):** EV / AC (values ¡1 indicate over budget)
   - **Milestone Tracking:** Monitoring the timely completion of key project milestones.

3. **Track Risks:** Re-evaluate risk probability and impact, monitor trigger conditions (risk symptoms), and track the effectiveness of mitigation plans.

4. **Manage Change Control:** Review, approve, or reject all change requests against the baselines. Ensure that approved changes are reflected in updated plans and communicated.

5. **Report Performance:** Communicate project status to stakeholders through regular reports (e.g., weekly status reports). Reports should highlight accomplishments, variances, key issues, risks, and forecasts for completion dates and costs.

6. **Initiate Corrective Actions:** When variances exceed acceptable thresholds, take action to bring the project back on track. This could involve re-allocating resources, adjusting the schedule, reducing scope (via change control), or refining estimates.

7. **Manage Issues:** Identify, track, and resolve problems that arise and are not covered by the risk management plan. Maintain an issue log with owners and target resolution dates.

7. The Five Different Artifacts Sets

In a software development process, especially in iterative and phase-based models like the Rational Unified Process (RUP), artifacts are grouped into five sets. These represent the different perspectives on the project and the evolving system.

1. **Management Set:** Artifacts used to manage the project itself. They capture the project's plan, status, and risks, providing the control framework.
    - *Examples:* Project Plan, Risk List, Work Breakdown Structure (WBS), Schedule, Iteration Plan, Status Reports.
2. **Requirements Set:** Artifacts that capture what the system must do from the user's and stakeholder's perspective. They form the contract between developers and stakeholders.
    - *Examples:* Vision Document, Use-Case Model, Supplementary Specifications (Non-functional requirements), User Stories.
3. **Design Set:** Artifacts that describe how the system will be built to meet the requirements. They detail the architecture, components, and their interactions.
    - *Examples:* Design Model (UML diagrams: class, sequence, state, activity), Software Architecture Document (SAD), Database Model, Interface Specifications.
4. **Implementation Set:** Artifacts that are the actual executable code and related files. These are the tangible outputs of the construction activity.
    - *Examples:* Source code files, header files, configuration files, build scripts (e.g., Maven, Gradle), and the compiled or packaged executable application.
5. **Deployment Set:** Artifacts used to deliver, operate, and maintain the system in its target environment. These ensure the system can be successfully transitioned to the user.
    - *Examples:* User manuals, online help systems, deployment scripts (e.g., Terraform, Ansible), installation guides, release notes, training materials.

8. Artifacts Evolution of the Life Cycle

In an iterative lifecycle (e.g., Spiral, RUP, Agile), artifacts do not remain static; they evolve in completeness, detail, and stability across phases. This evolution is critical for managing risk and uncertainty.

- **Inception Phase:**
    - **Goal:** Establish feasibility, scope, and a common vision.
    - **Artifact State:** "Initial" or "Conceptual." Artifacts are at a very high level.

  - *Requirements:* A few key use cases identified; the Vision document is drafted.
  - *Design:* A preliminary, high-level architecture sketch.
  - *Management:* A high-level project plan with initial risk assessment.

- **Elaboration Phase:**
  - **Goal:** Establish a stable architectural baseline and mitigate major risks.
  - **Artifact State:** "Refined," "Baseline," and "Executable."
  - *Requirements:* Most use cases are detailed; the requirements model is baselined.
  - *Design:* The core architecture is detailed, implemented in a prototype, tested, and baselined.
  - *Management:* A detailed project plan and schedule are created.

- **Construction Phase:**
  - **Goal:** Build the system through a series of iterations to deliver a complete product.
  - **Artifact State:** "Iteratively Completed" and "Growing."
  - *Requirements:* Remaining use cases and non-functional requirements are finalized.
  - *Design:* All design elements are completed and detailed.
  - *Implementation:* Code for all features is written, tested, integrated, and refined.

- **Transition Phase:**
  - **Goal:** Deploy the system to the end-user.
  - **Artifact State:** "Finalized" and "Delivered."
  - *Requirements:* The final requirements baseline is locked.
  - *Deployment:* User manuals, training materials, and deployment scripts are completed.
  - *Implementation:* The final, fully tested, and validated executable is produced and released.

9. Software Architecture View vs. Software Architecture Description

These two concepts are foundational to software architecture documentation. A view is a perspective; a description is the complete artifact.

Table 1: Software Architecture View vs. Architecture Description

Feature	Software Architecture View	Software Architecture Description
Definition	A representation of a set of system elements and the relationships associated with them from a particular stakeholder's perspective (e.g., end-user, developer, project manager, systems engineer). It is a subset of the architecture.	The complete collection of architectural views and any other documentation that describes the system's architecture. It is a formal document or collection of models that serves as the authoritative source for the architecture.
Purpose	To address the specific concerns of particular stakeholder groups. Each view provides a lens for focusing on a specific set of quality attributes. For example: • **Logical View:** Addresses functional requirements. • **Process View:** Addresses performance and scalability. • **Physical View:** Addresses deployment and hardware concerns. • **Development View:** Addresses software organization and structure.	To provide a complete, unambiguous, and consistent record of the architecture. It serves as: • A communication medium between all stakeholders. • A blueprint for the development and testing teams. • The basis for analysis and evaluation. • Documentation for future maintenance.
Analogy	Like different blueprints for a house: the electrical plan is one view (for the electrician), the plumbing plan is another (for the plumber), the structural framing is another. Each shows different elements and relationships relevant to a specific craft.	The complete set of all blueprints (architectural, electrical, plumbing, structural) plus a specification guide that explains how they fit together and the rationale behind key decisions. This is the house's complete architectural description.
Key Frameworks	The **4+1 Architectural View Model** (Logical, Process, Development, Physical, and Scenarios) is[9] a seminal framework for defining architecture views.	An **Architecture Description Document (ADD)** or a collection of model files (e.g., in a modeling tool) is the standard deliverable containing all the views, architecture deci-