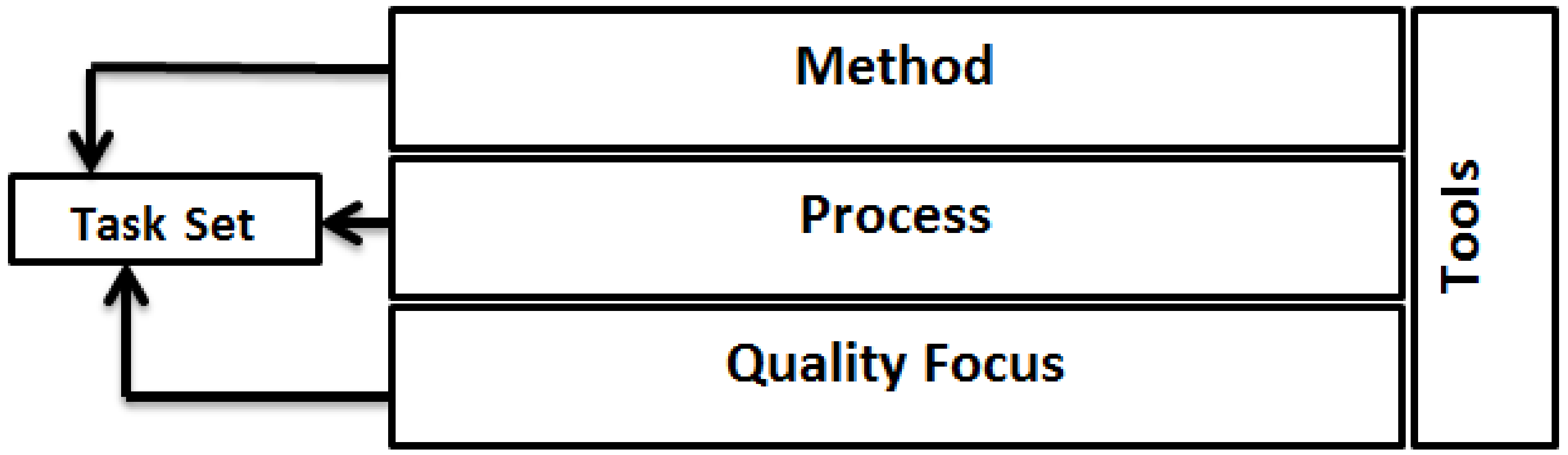
Chapter 2and 3
Software Engineering and Software Process Structure

Software Engineering: A Practitioner’s Approach
by Roger S. Pressman and Bruce R. MAXIM

Chapter Overview

- What? A software process - a series of predictable steps that leads to a timely, high-quality product.
- Who? Managers, software engineers, and customers.
- Why? Provides stability, control, and organization to an otherwise chaotic activity.
- Steps? A handful of activities are common to all software processes, details vary.
- Work product? Programs, documents, and data.
- Correct process? Assessment, quality deliverable.

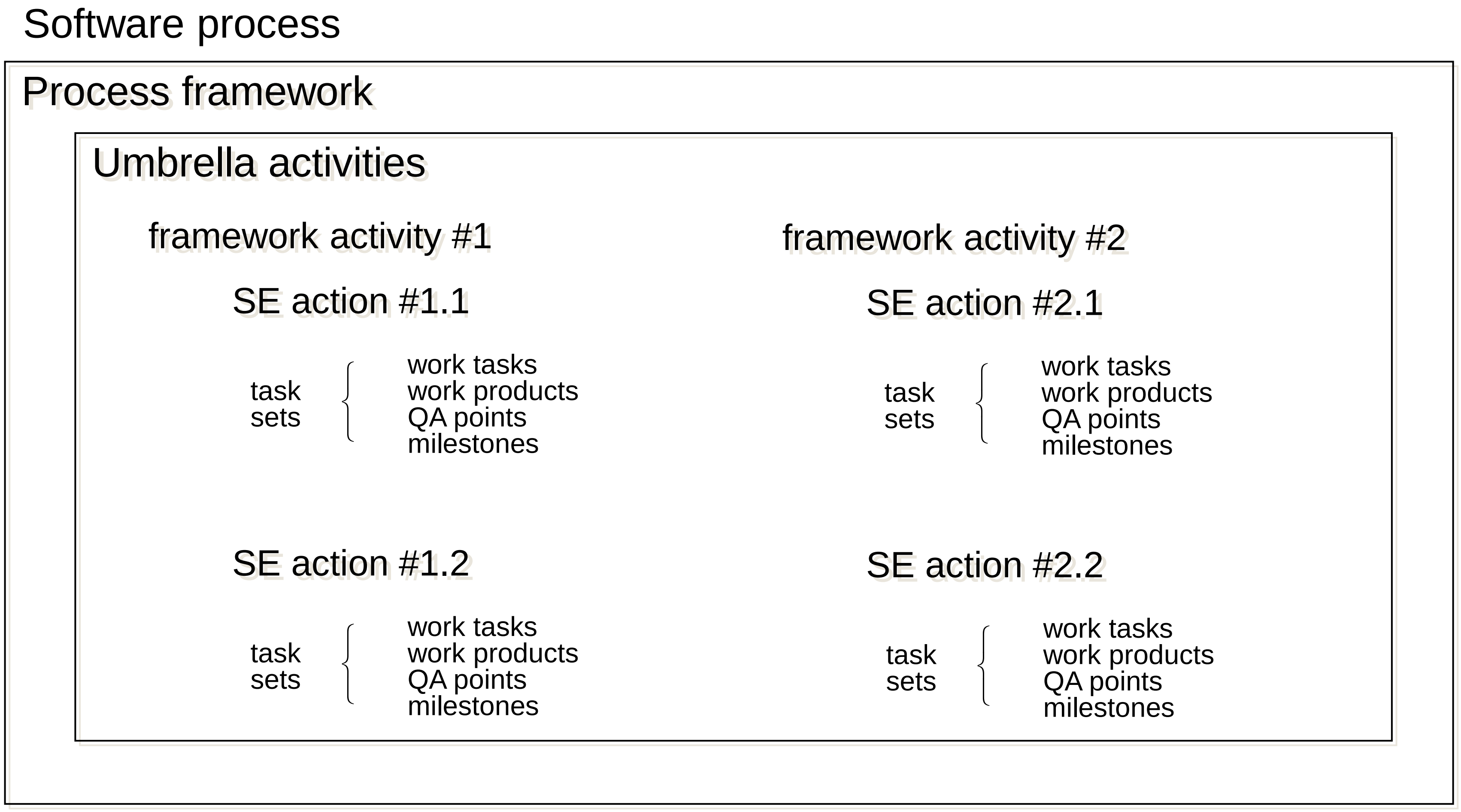
Software Engineering Framework



- Any Engineering approach must be founded on organizational commitment to quality. That means the software development organization must have special focus on quality while performing the software engineering activities. Based on this commitment to quality by the organization, a software engineering framework is proposed that is shown in figure.
- **Quality Focus:** As we have said earlier, the given framework is based on the organizational commitment to quality. The quality focus demands that processes be defined for rational and timely development of software. And quality should be emphasized while executing these processes.

- **Processes:** The processes are set of key process areas (KPAs) for effectively manage and deliver quality software in a cost effective manner. The processes define the tasks to be performed and the order in which they are to be performed. Every task has some deliverables and every deliverable should be delivered at a particular milestone.
- **Methods:** Methods provide the technical "how-to's" to carryout these tasks. There could be more than one technique to perform a task and different techniques could be used in different situations.
- **Tools:** Tools provide automated or semi-automated support for software processes, methods, and quality control.

A Process Framework



The software Process

- A *process* is a collection of activities, actions, and tasks that are performed when some work product is to be created.
- An *activity* strives to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.
- An *action* (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural design model)
- A *task* focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

Framework Activities

- A *process framework* establishes the foundation for a complete software engineering process by identifying a small number of *framework activities* that are applicable to all software projects, regardless of their size or complexity. In addition, the
- A generic process framework for software engineering encompasses five activities:
- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment

- **Communication.** Before any technical work can commence, it is critically important to communicate and collaborate with the customer (and other stakeholders) The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.
- **Planning.** defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.
- **Modeling.** creating models to better understand software requirements and the design that will achieve those requirements.
- **Construction.** This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.
- **Deployment.** The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

Umbrella Activities

process framework encompasses a set of *umbrella activities* that are applicable across the entire software process.

- Software project tracking and control
- Risk Management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

- **Risk management**—assesses risks that may affect the outcome of the project or the quality of the product.
- **Software quality assurance**—defines and conducts the activities required to ensure software quality.
- **Technical reviews**—assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.
- **Measurement**—defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.

- **Software configuration management**—manages the effects of change throughout the software process.
- **Reusability management**—defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
- **Work product preparation and production**—encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

The Process Model: Adaptability

- The framework activities will <u>always</u> be applied on <u>every</u> project … BUT
- The tasks (and degree of rigor) for each activity will vary based on:
  - the type of project
  - characteristics of the project
  - common sense judgment; concurrence of the project team

A process adopted for one project might be significantly different than a process adopted for another project. Among the differences are

- Overall flow of activities, actions, and tasks and the interdependencies among them
- Degree to which actions and tasks are defined within each framework activity
- Degree to which work products are identified and required
- Manner in which quality assurance activities are applied
- Manner in which project tracking and control activities are applied
- Overall degree of detail and rigor with which the process is described
- Degree to which the customer and other stakeholders are involved with the project
- Level of autonomy given to the software team
- Degree to which team organization and roles are prescribed

How does a framework activity change as the nature of the project changes?

- For a small software project requested by one person (at a remote location) with simple, straightforward requirements, the communication activity might encompass little more than a phone call with the appropriate stakeholder. Therefore, the only necessary action is *phone conversation*, and the work tasks (the *task set*) that this action encompasses are:

1. Make contact with stakeholder via telephone.
2. Discuss requirements and take notes.
3. Organize notes into a brief written statement of requirements.
4. E-mail to stakeholder for review and approval.

Identifying a Task Set:

- Each software engineering action (e.g., *elicitation*, an action associated with the communication activity) can be represented by a number of different *task sets*—each a collection of software engineering work tasks, related work products, quality assurance points, and project milestones.
- You should choose a task set that best accommodates the needs of the project and the characteristics of your team. This implies that a software engineering action can be adapted to the specific needs of the software project and the characteristics of the project team.

Process Patterns:

- A *process pattern* describes a process-related problem that is encountered during software engineering work, identifies the environment in which the problem has been encountered, and suggests one or more proven solutions to the problem.
- Stated in more general terms, a process pattern provides you with a template -a consistent method for describing problem solutions within the context of the software process.
- By combining patterns, a software team can solve problems and construct a process that best meets the needs of a project.

- Patterns can be defined at any level of abstraction. In some cases, a pattern might be used to describe a problem (and solution) associated with a complete process model (e.g., prototyping).
- In other situations, patterns can be used to describe a problem (and solution) associated with a framework activity (e.g., **planning**) or an action within a framework activity (e.g., project estimating).

Template for describing a process pattern:

- **Pattern Name.** The pattern is given a meaningful name describing it within the context of the software process (e.g.,Technical Reviews).
- **Forces.** The environment in which the pattern is encountered and the issues that make the problem visible and may affect its solution.
- **Type.** The pattern type is specified. There are three types:

1. ***Stage pattern***—defines a problem associated with a framework activity for the process. Since a framework activity encompasses multiple actions and work tasks, a stage pattern incorporates multiple task patterns (see the following) that are relevant to the stage (framework activity). An example of a stage pattern might be **EstablishingCommunication.** This pattern would incorporate the task pattern **RequirementsGathering** and others.
2. ***Task pattern***—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice (e.g., **RequirementsGathering** is a task pattern).
3. *Phase pattern*—define the sequence of framework activities that occurs within the process, even when the overall flow of activities is iterative in nature. An example of a phase pattern might be **SpiralModel** or **Prototyping**

Initial context. Describes the conditions under which the pattern applies. Prior to the initiation of the pattern:

(1) What organizational or team-related activities have already occurred?

(2) What is the entry state for the process?

(3) What software engineering information or project information already exists?

- For example, the **Planning** pattern (a stage pattern) requires that

1. customers and software engineers have established a collaborative communication;
2. successful completion of a number of task patterns [specified] for the **Communication** pattern has occurred; and
3. the project scope, basic business requirements, and project constraints are known.

- **Problem.** The specific problem to be solved by the pattern.
- **Solution.** Describes how to implement the pattern successfully. It also describes how software engineering information or project information that is available before the initiation of the pattern is transformed as a consequence of the successful execution of the pattern.
- **Resulting Context.** Describes the conditions that will result once the pattern has been successfully implemented. Upon completion of the pattern:

(1) What organizational or team-related activities must have occurred?

(2) What is the exit state for the process?

(3) What software engineering information or project information has been developed?

Process patterns provide an effective mechanism for addressing problems associated with any software process. The patterns enable you to develop a hierarchical process description that begins at a high level of abstraction

SOFTWARE ENGINEERING PRACTICE

The Essence of Practice:

1. *Understand the problem* (communication and analysis).
2. *Plan a solution* (modeling and software design).
3. *Carry out the plan* (code generation).
4. *Examine the result for accuracy* (testing and quality assurance).

Understand the problem

- *Who has a stake in the solution to the problem?* That is, who are the stakeholders?
- *What are the unknowns?* What data, functions, and features are required to properly solve the problem?
- *Can the problem be compartmentalized?* Is it possible to represent smaller problems that may be easier to understand?
- *Can the problem be represented graphically?* Can an analysis model be created?

Plan the solution.

- *Have you seen similar problems before?*
- *Has a similar problem been solved?*
- *Can sub problems be defined?*
- *Can you represent a solution in a manner that leads to effective implementation?*

Carry out the plan.

- *Does the solution conform to the plan?* Is source code traceable to the design model?
- *Is each component part of the solution provably correct?*
- Have the design and code been reviewed, or better, have correctness proofs been applied to the algorithm?

Examine the result

- *Is it possible to test each component part of the solution?* Has a reasonable
- testing strategy been implemented?
- *Does the solution produce results that conform to the data, functions, and features that are required?* Has the software been validated against all stakeholder requirements?