Chapter 1

Software & Software Engineering

Software Engineering: A Practitioner’s Approach, 7/e by Roger S. Pressman

For non-profit educational use only

What is Software?

Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information and (3) documentation that describes the operation and use of the programs.

What is Software?

- Software is developed or engineered, it is not manufactured in the classical sense.
- Software doesn’t “wear out.”
- Although the industry is moving toward component-based construction, most software continues to be custom-built.

Wear vs. Deterioration

Software Applications

- system software
- application software
- engineering/scientific software
- embedded software
- product-line software
- WebApps (Web applications)
- AI software

Software—New Categories

- Open world computing—pervasive, distributed computing
- Ubiquitous computing—wireless networks
- Networking—the Web as a computing engine
- Open source—free source code open to the computing community (a blessing, but also a potential curse)
- Also... (see Chapter 31)
  - Data mining
  - Grid computing
  - Cognitive machines
  - Software for nanotechnologies

Legacy Software

Why must it change?

- software must be adapted to meet the needs of new computing environments or technology
- software must be enhanced to implement new business requirements
- software must be extended to make it interoperable with other more modern systems or databases
- software must be re-architected to make it viable within a network environment

Characteristics of WebApps - I

- Network intensiveness. A WebApp resides on a network and must serve the needs of a diverse community of clients.
- Concurrency. A large number of users may access the WebApp at one time.
- Unpredictable load. The number of users of the WebApp may vary by orders of magnitude from day to day.
- Performance. If a WebApp user waits too long for access, for server-side processing, for client-side formatting and display, he or she may decide to go elsewhere.
- Availability. Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a "24/7/365" basis.

Characteristics of WebApps - II

- Data driven. The primary function of many WebApps is to use hypertext to present text, graphics, audio, and video content to the end-user.
- Content sensitive. The quality and aesthetic nature of content remains an important determinant of the quality of a WebApp.
- Continuous evolution. Unlike conventional application software that evolves over a series of planned, chronologically-spaced releases, Web applications evolve continuously.
- Immediacy. Although immediate—the compelling need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time to market that can be a matter of a few days or weeks.
- Security. Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end-users who may access the application.
- Aesthetics. An undeniable part of the appeal of a WebApp is its look and feel.
Software Engineering

- Some realities:
  - a concerted effort should be made to understand the problem before a software solution is developed
  - design becomes a pivotal activity
  - software should exhibit high quality
  - software should be maintainable

- The seminal definition:
  - Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

A Process Framework

- Process framework
- Framework activities
  - work tasks
  - work products
  - milestones & deliverables
  - QA checkpoints

Framework Activities

- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
  - Construction
  - Code generation
  - Testing
  - Deployment

Adapting a Process Model

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

The Essence of Practice

- Polya suggests:
  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).

Umbrella Activities

- Software project management
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Work product preparation and production
- Reusability management
- Measurement
- Risk management

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? Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- Has a similar problem been solved? If so, are elements of the solution reusable?
- Can subproblems be defined? If so, are solutions readily apparent for the subproblems?
- Can you represent a solution in a manner that leads to effective implementation? Can a design model be created?

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? Has the design and code been reviewed, or better, have correctness proofs been applied to 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?

Hooker’s General Principles

- 1: The Reason It All Exists
- 2: KISS (Keep it Simple, Stupid)
- 3: Maintain the Vision
- 4: What You Produce, Others Will Consume
- 5: Be Open to the Future
- 6: Plan Ahead for Reuse
- 7: Think!

Software Myths

- Affect managers, customers (and other non-technical stakeholders) and practitioners
- Are believable because they often have elements of truth, but …
- Invariably lead to bad decisions, therefore …
- Insist on reality as you navigate your way through software engineering

How It all Starts

- SafeHome:
  - Every software project is precipitated by some business need—
    - the need to correct a defect in an existing application;
    - the need to extend the functions and features of an existing application, or
    - the need to create a new product, service, or system.
Identifying a Task Set

- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.
  - A list of the task to be accomplished
  - A list of the work products to be produced
  - A list of the quality assurance filters to be applied

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 [Arm96]—a consistent method for describing problem solutions within the context of the software process.

Process Pattern Types

- **Stage patterns**—defines a problem associated with a framework activity for the process.
- **Task patterns**—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice
- **Phase patterns**—define the sequence of framework activities that occur with the process, even when the overall flow of activities is iterative in nature.

Process Assessment and Improvement

- **Standard CMMS Assessment Method for Process Improvement (SCAMPI)**—provides a five step process assessment model that incorporates five phases: Initiating, diagnosing, establishing, acting and finding.
- **CMMS-Based Appraisal for Internal Process Improvement (CBAP)**—provides a systematic technique for assessing the relative maturity of a software organization, uses the SSh CMMS as the basis for the assessment [Bren97].
- **SPICE**—The SPICE (ISO/IEC15504) standard defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process, [ICOCO]
- **ISO 9001:2000 for Software**—a generic standard that applies to any organization that wants to improve the overall quality of its products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies, [Arm96]

Prescriptive Models

- Prescriptive process models advocate an orderly approach to software engineering. That leads to a few questions:
  - If prescriptive process models strive for structure and order, are they inappropriate for a software world that thrives on change?
  - Yet, if we reject traditional process models (and the order they imply) and replace them with something less structured, do we make it impossible to achieve coordination and coherence in software work?

The Waterfall Model

![The Waterfall Model Diagram](image)

The V-Model

![The V-Model Diagram](image)

The Incremental Model

![The Incremental Model Diagram](image)

Evolutionary Models: Prototyping

![Evolutionary Models: Prototyping Diagram](image)
Evolutionary Models: The Spiral

Evolutionary Models: Concurrent

Still Other Process Models
- Component based development—the process to apply when reuse is a development objective
- Formal methods—emphasizes the mathematical specification of requirements
- AOSD—provides a process and methodological approach for defining, specifying, designing, and constructing aspects
- Unified Process—a "use-case driven, architecture-centric, iterative and incremental" software process closely aligned with the Unified Modeling Language (UML)

The Unified Process (UP)

UP Phases

UP Work Products

Personal Software Process (PSP)
- Planning: The activity lists requirements and develops both size and resource estimates. In addition, or about estimates the number of defects likely to be found over the project.
- High-level design: A system requirements document is created, which describes the system requirements (Chapter 2) and the project's major components. The project's requirements are decomposed into major components, which are then decomposed into smaller components.
- Development: The component-level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Defects are identified, which are maintained for all important tasks and work results.
- Postmortem: Using the measures and metrics collected (this is a substantial amount of data that must all be analyzed statistically), the effectiveness of the process is determined. Measures and metrics also provide guidance for improving the process to improve its effectiveness.

Team Software Process (TSP)
- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPT) of three to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected.
- The Capability Maturity Model (CMM), a measure of the effectiveness of a software process, is discussed in Chapter 10.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industry-grade team skills.
Software Process

- Process is distinct from product – products are outcomes of executing a process on a project
- SW Engg. focuses on process
- Premise: Proper processes will help achieve project objectives of high QP

Component Software Processes

- Two major processes
  - Development – focuses on development and quality steps needed to engineer the software
  - Project management – focuses on planning and controlling the development process
- Development process is the heart of software process; other processes revolve around it
- These are executed by different people
  - developers execute engg. Process
  - project manager executes the mgmt proces

Component Processes...

- Other processes
  - Configuration management process: manages the evolution of artifacts
  - Change management process: how changes are incorporated
  - Process management process: management of processes themselves
  - Inspection process: How inspections are conducted on artifacts

Process Specification

- Process is generally a set of phases
- Each phase performs a well defined task and generally produces an output
- Intermediate outputs – work products
- At top level, typically few phases in a process
- How to perform a particular phase – methodologies have been proposed

ETVX Specification

- ETVX approach to specify a step
  - Entry criteria: what conditions must be satisfied for initiating this phase
  - Task: what is to be done in this phase
  - Verification: the checks done on the outputs of this phase
  - Exit criteria: when can this phase be considered done successfully
- A phase also produces info for mgmt

ETVX approach

- Provide high Q&P
  - Support testability as testing is the most expensive task; testing can consume 30 to 50% of total development effort
  - Support maintainability as maintenance can be more expensive than development; over life up to 80% of total cost
  - Remove defects early, as cost of removing defects increases with latency

High Q&P: Early Defect Removal...

- Cost of a defect increases with latency
  - I.e. fixing a req defect in operation can cost a 100 times the cost of fixing it in requirements itself
- Hence, for high Q&P, the process must support early defect removal
- That is why there is a V in ETVX, and quality control tasks in the sw process
Early Defect Removal...

Desired Properties...
- Predictability and repeatability
  - Process should repeat its performance when used on different projects
  - I.e. outcome of using a process should be predictable
  - Without predictability, cannot estimate, or say anything about quality or productivity
  - With predictability, past performance can be used to predict future performance

Predictability...
- Predictable process is said to be under statistical control
  - Repeatedly using the process produces similar results
  - Results – properties of interest like quality, productivity...
  - To consistently develop sw with high Q&P, process must be in control

Predictability...

Support Change
- Software changes for various reasons
- Requirements change is a key reason
- Requirement changes cannot be wished away or treated as “bad”
- They must be accommodated in the process for sw development

Summary
- Process – method for doing something
- Process typically has stages, each stage focusing on an identifiable task
- Stages have methodologies
- Software process is the methods for developing software
- Best to view it as comprising of multiple processes

Summary
- Goal is to produce software with high quality and productivity
- Process is the means
- Development process is central process
- Mgmt process is for controlling dev
- Other supporting processes
- Sw process should have high Q&P, predictability, and support for change

Development Process and Process Models

Software Project
- Project – to build a sw system within cost and schedule and with high quality which satisfies the customer
- Project goals – high Q and high P
- Suitable process needed to reach goals
- For a project, the process to be followed is specified during planning
Development Process

- A set of phases and each phase being a sequence of steps
- Sequence of steps for a phase - methodologies for that phase.
- Why have phases
  - To employ divide and conquer
  - Each phase handles a different part of the problem
  - Helps in continuous validation

Requirement Analysis

- To understand state the problem precisely
- Forms the basis of agreement between user and developer
- Specifies "what", not "how".
- Not an easy task, as needs often understood.
- Requirement specifications of even medium systems can be many hundreds of pages
- Output is the sw req spec (SRS) document

Design

- A major step in moving from problem domain to solution domain; three main tasks
  - Architecture design - components and connectors that should be there in the system
  - High level design - modules and data structures needed to implement the architecture
  - Detailed design - logic of modules
- Most methodologies focus on architecture or high level design
- Outputs are arch/des/logic design docs

Coding

- Converts design into code in specific language
- Goal: Implement the design with simple and easy to understand code.
  - Code should be simple and readable.
- The coding phase affects both testing and maintenance. Well written code can reduce the testing and maintenance effort.
- Output is code

Testing

- Defects are introduced in each phase
- They have to be found and removed to achieve high quality
- Testing plays this important role
- Goal: Identify most of defects
- Is a very expensive task; has to be properly planned and executed.
- Outputs are Test plans/results, and the final tested (hopefully reliable) code

Effort Distribution

**Distribution of effort:**
- Req. - 10-20%
- Design - 10-20%
- Coding - 20-30%
- Testing - 30-50%
- Coding is not the most expensive.

Distribution of effort...

- How programmers spend their time
  - Writing programs - 13%
  - Reading programs and manuals - 16%
  - Job communication - 32%
  - Others - 39%
- Programmers spend more time in reading programs than in writing them.
- Writing programs is a small part of their lives.

Defects

- Distribution of error occurrences by phase is
  - Req. - 20%
  - Design - 30%
  - Coding - 50%
- Defects can be injected at any of the major phases.
- Cost of latency: Cost of defect removal increases exponentially with latency time.
Defects...

- Cheapest way to detect and remove defects close to where it is injected.
- Hence must check for defects after every phase.

Process Models

- A process model specifies a general process, usually as a set of stages
- This model will be suitable for a class of projects
- I.e. a model provides generic structure of the process that can be followed by some projects to achieve their goals

Projects Process

- If a project chooses a model, it will generally tailor it to suit the project
- This produces the spec for the projects process
- This process can then be followed in the project
- I.e. process is what is actually executed; process spec is plan about what should be executed; process model is a generic process spec
- Many models have been proposed for the development process

Typical Student Process Model

- Get problem stmt – Code – do some testing – deliver/demo
- Why this process model cannot be used for commercial projects?
  - Produces student-software, which is not what we are after
  - Cannot ensure desired quality for industrial-strength software

Common Process Models

- Waterfall – the oldest and widely used
- Prototyping
- Iterative – currently used widely
- Timeboxing

Waterfall Model

- Linear sequence of stages/phases
- Requirements – HLD – DD – Code – Test – Deploy
- A phase starts only when the previous has completed; no feedback
- The phases partition the project, each addressing a separate concern

Waterfall...

- Linear ordering implies each phase should have some output
- The output must be validated/certified
- Outputs of earlier phases: work products
- Common outputs of a waterfall: SRS, project plan, design docs, test plan and reports, final code, supporting docs

Waterfall Advantages

- Conceptually simple, cleanly divides the problem into distinct phases that can be performed independently
- Natural approach for problem solving
- Easy to administer in a contractual setup – each phase is a milestone
Waterfall disadvantages

- Assumes that requirements can be specified and frozen early
- May fix hardware and other technologies too early
- Follows the "big bang" approach – all or nothing delivery; too risky
- Very document oriented, requiring docs at the end of each phase

Waterfall Usage

- Has been used widely
- Well suited for projects where requirements can be understood easily and technology decisions are easy
- I.e. for familiar type of projects it still may be the most optimum

Prototyping

- Prototyping addresses the requirement specification limitation of waterfall
- Instead of freezing requirements only by discussions, a prototype is built to understand the requirements
- Helps alleviate the requirements risk
- A small waterfall model replaces the requirements stage

Prototyping

- Development of prototype
  - Starts with initial requirements
  - Only key features which need better understanding are included in prototype
  - No point in including those features that are well understood
  - Feedback from users taken to improve the understanding of the requirements

Prototyping

- Cost can be kept low
  - Build only features needing clarification
  - "quick and dirty" – quality not important, scripting etc can be used
  - Things like exception handling, recovery, standards are omitted
  - Cost can be a few % of the total
  - Learning in prototype building will help in building, besides improved requirements

Prototyping

- Advantages: req will be more stable, req frozen later, experience helps in the main development
- Disadvantages: Potential hit on cost and schedule
- Applicability: When req are hard to elicit and confidence in reqs is low; i.e. where reqs are not well understood

Iterative Development

- Counts the "all or nothing" drawback of the waterfall model
- Combines benefit of prototyping and waterfall
- Develop and deliver software in increments
- Each increment is complete in itself
- Can be viewed as a sequence of waterfalls
- Feedback from one iteration is used in the future iterations

Iterative Enhancement
Iterative Development

- Products almost always follow it
- Used commonly in customized development also
  - Businesses want quick response for sw
  - Cannot afford the risk of all-or-nothing
- Newer approaches like XP, Agile,... all rely on iterative development

Iterative Development

- Benefits: Get-as-you-pay, feedback for improvement,
- Drawbacks: Architecture/design may not be optimal, rework may increase, total cost may be more
- Applicability: where response time is important, risk of long projects cannot be taken, all req not known

Timeboxed Iterations

- General iterative development – fix the functionality for each iteration, then plan and execute it
- In time boxed iterations – fix the duration of iteration and adjust the functionality to fit it
- Completion time is fixed, the functionality to be delivered is flexible

Time Boxed Iterations

- This itself very useful in many situations
- Has predictable delivery times
- Overall product release and marketing can be better planned
- Makes time a non-negotiable parameter and helps focus attention on schedule
- Prevents requirements bloating
- Overall dev time is still unchanged

Timeboxing Model – Basics

- Development is done iteratively in fixed duration time boxes
- Each time box divided in fixed stages
- Each stage performs a clearly defined task that can be done independently
- Each stage approximately equal in duration
- There is a dedicated team for each stage
- When one stage team finishes, it hands over the project to the next team

Timeboxing

- Iterative is linear sequence of iterations
- Each iteration is a mini waterfall – decide the specs, then plan the iteration
- Time boxing – fix an iteration duration, then determine the specs
- Divide iteration in a few equal stages
- Use pipelining concepts to execute iterations in parallel

Timeboxing – Taking Time Boxed Iterations Further

- What if we have multiple iterations executing in parallel
- Can reduce the average completion time by exploiting parallelism
- For parallel execution, can borrow pipelining concepts from hardware
- This leads to Timeboxing Process Model

Example

- An iteration with three stages – Analysis, Build, Deploy
  - These stages are appx equal in many situations
  - Can adjust durations by determining the boundaries suitably
  - Can adjust duration by adjusting the team size for each stage
- Have separate teams for A, B, and D
Pipelined Execution

- AT starts executing it-1
- AT finishes, hands over it-1 to BT, starts executing it-2
- AT finishes it-2, hands over to BT; BT finishes it-1, hands over to DT; AT starts it-3, BT starts it-2 (and DT, it-1)
- ...

Timeboxing Execution

- First iteration finishes at time T
- Second finishes at T+T/3; third at T+2 T/3, and so on
- In steady state, delivery every T/3 time
- If T is 3 weeks, first delivery after 3 wks, 2nd after 4 wks, 3rd after 5 wks,...
- In linear execution, delivery times will be 3 wks, 6 wks, 9 wks,...

Timeboxing execution

- Duration of each iteration still the same
- Total work done in a time box is also the same
- Productivity of a time box is same
- Yet, average cycle time or delivery time has reduced to a third

Team Size

- In linear execution of iterations, the same team performs all stages
- If each stage has a team of S, in linear execution the team size is S
- In pipelined execution, the team size is three times (one for each stage)
- I.e. the total team size in timeboxing is larger; and this reduces cycle time

Team Size

- Merely by increasing the team size we cannot reduce cycle time - Brook's law
- Timeboxing allows structured way to add manpower to reduce cycle time
- Note that we cannot change the time of an iteration – Brook's law still holds
- Work allocation different to allow larger team to function properly

Work Allocation of Teams

<table>
<thead>
<tr>
<th>Requirements Team</th>
<th>Requirements Analyst/TE</th>
<th>Requirements Analyst/TE</th>
<th>Requirements Analyst/TE</th>
<th>Requirements Analyst/TE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build/Team</td>
<td>Build/TE</td>
<td>Build/TE</td>
<td>Build/TE</td>
<td>Build/TE</td>
</tr>
<tr>
<td>Deploy/Team</td>
<td>Deploy/TE</td>
<td>Deploy/TE</td>
<td>Deploy/TE</td>
<td>Deploy/TE</td>
</tr>
</tbody>
</table>

Timeboxing

- Advantages: Shortened delivery times, other adv of iterative, distr. execution
- Disadvantages: Larger teams, proj mgmt is harder, high synchronization needed, CM is harder
- Applicability: When short delivery times v. imp.; architecture is stable; flexibility in feature grouping

Summary

- Process is a means to achieve project objectives of high QP
- Process models define generic process, which can form basis of project process
- Process typically has stages, each stage focusing on an identifiable task
- Many models for development process have been proposed
PM Process Phases
- There are three broad phases
  - Planning
  - Monitoring and control
  - Termination analysis
- Planning is a key activity that produces a plan, which forms the basis of monitoring

Planning
- Done before project begins
- Key tasks
  - Cost and schedule estimation
  - Staffing
  - Monitoring and risk input plans
  - Quality assurance plans
  - Etc.
- Will discuss planning in detail later

Monitoring and control
- Lasts for the duration of the project and covers the development process
- Monitors all key parameters like cost, schedule, risks
- Takes corrective actions when needed
- Needs information on the dev process—provided by metrics

Termination Analysis
- Termination analysis is performed when the development process is over
- Basic purpose: to analyze the performance of the process, and identify lessons learned
- Also called postmortem analysis

Relationship with Dev Process

Background
- Main goal of inspection process is to detect defects in work products
- First proposed by Fagan in 70s
- Earlier used for code phase, now used for all types of work products
- Is recognized as an industry best practice
- Data suggests that it improves both quality & productivity

Background
- Defects injected in software at any stage
- Hence must remove them at every stage
- Inspections can be done on any document including design docs and plans
- Is a good method for early phases like requirements and design
- Also useful for plans (PM plans, CM plans, testing plans, ...)

Some Characteristics
- Conducted by group of technical people for technical people (i.e., review done by peers)
- Is a structured process with defined roles for the participants
- The focus is on identifying problems, not resolving them
- Review data is recorded and used for monitoring the effectiveness

The Inspection Process
**A Review Process**

- Planning
  - Select the group review team – three to five people group is best
  - Identify the moderator – has the main responsibility for the inspection
  - Prepare package for distribution – work product for review plus supporting docs
  - Package should be complete for review

- Self-Review Log

  | Project name: | Work product name and ID: |
  | Reviewer Name | Effort spent (hours) |
  | Defect list | No | Location | Description | Criticality |

- Group Review Meeting
  - Purpose – define the final defect list
  - Entry criteria – each member has done a proper self-review (logs are reviewed)
  - Group review meeting
    - A reviewer goes over the product line by line
    - At any line, all issues are raised
    - Discussion follows to identify if a defect
    - Decision recorded (by the scribe)

- Group Review Meeting...
  - At the end of the meeting
    - Scribe records and presents the list of defects/issues
    - If few defects, the work product is accepted; else it might be asked for another review
    - Group does not propose solutions – though some suggestions may be recorded
    - A summary of the inspections is prepared – useful for evaluating effectiveness

**Planning**

- Select the group review team – three to five people group is best
- Identify the moderator – has the main responsibility for the inspection
- Prepare package for distribution – work product for review plus supporting docs
- Package should be complete for review

**Summary Report Example**

<table>
<thead>
<tr>
<th>Project</th>
<th>XXXX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Product Type</td>
<td>Project plan</td>
</tr>
<tr>
<td>Size of work product</td>
<td>14 pages</td>
</tr>
<tr>
<td>Review team</td>
<td>P1, P2, P3</td>
</tr>
<tr>
<td>Effort (person hours)</td>
<td>Total 20</td>
</tr>
<tr>
<td>Preparation</td>
<td>10 (total)</td>
</tr>
<tr>
<td>Group meeting</td>
<td>10</td>
</tr>
</tbody>
</table>

**Summary Contd.**

- Defects
  - No of critical defects | 0 |
  - No of major defects | 3 |
  - No of minor defects | 16 |
  - Total | 19 |
- Review status | Accepted |
- Reco for next phase | Nil |
- Comments | Nice plan |
Rework and Follow Up
- Defects in the defects list are fixed later by the author.
- Once fixed, the author gets it OKed by the moderator, or goes for another review.
- Once all defects/issues are satisfactorily addressed, review is completed and collected data is submitted.

Roles and Responsibilities
- Main roles: Moderator, reader, scribe, author, reviewer.
- Moderator – overall responsibility.
  - "Moderator cannot be a reader."
- Author – who is the creator of the work product.
  - "Author cannot be Moderator or reader."
- Reviewer – to identify defects.
- Reader – not there in some processes, reads line by line to keep focus.
- Scribe – records the issues raised.
- "Moderator, reader, author can be reviewer and scribe."

Guidelines for Work Products

<table>
<thead>
<tr>
<th>Code</th>
<th>Code implements design</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code is complete and correct</td>
</tr>
<tr>
<td></td>
<td>Defects in code</td>
</tr>
<tr>
<td></td>
<td>Other quality issues</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test cases</th>
<th>Set of test cases test all SRS conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test cases</td>
<td>are executable</td>
</tr>
<tr>
<td></td>
<td>Are perf and load tests there</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proj Mgmt Plan</th>
<th>Plan is complete and specifies all components of the plan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Is implementable</td>
</tr>
<tr>
<td></td>
<td>Omissions and ambiguities</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Designer</th>
<th>Tester</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Req Spec</th>
<th>Meet customer needs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Are implementable</td>
</tr>
<tr>
<td></td>
<td>Omissions, inconsistencies, ambiguities</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HLD</th>
<th>Design implements req</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Design is implementable</td>
</tr>
<tr>
<td></td>
<td>Omissions, quality of design</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Participants</th>
<th>Customer Designer Tester, Dev Analyst</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Requirement Manager, Requirement Eng</td>
</tr>
</tbody>
</table>

Summary
- Purpose of reviews: to detect defects.
- Structured reviews are very effective - can detect most of the injected defects.
- For effective review, process has to be properly defined and followed.
- Data must be collected and analyzed.

Background
- A software project produces many items - programs, documents, data, manuals, ...
- All of these can be changed easily - need to keep track state of items.
- SCM: Systematically control the changes.
  - Focus on changes during normal evolution; req changes will be handled separately.
  - CM requires discipline as well as tools.

Background
- SCM often independent of dev process.
- Dev process looks at macro picture, but not on changes to individual items/files.
- As items are produced during the dev process, it is brought under SCM.
- SCM controls only the products of the development process.

SCM Process and Dev process

- Configuration Management
- Phase 1
- Phase 2
- ...
Need for CM
- To satisfy the basic project objective of delivering the product to the client
- What files should comprise the product
- What versions of these files
- How to combine these to make the product
- Just for this, versioning is needed, and state of different items has to be tracked
- There are other functions of CM also

Functionality Needed
- Give states of programs
- Give latest version of a program
- Undo a change and revert back to a specified version
- Prevent unauthorized changes
- Gather all sources, documents, and other information for the current system

CM Mechanisms
- Configuration identification and baselining
- Version control
- Access control
- These are the main mechanisms, there are others like naming conventions, directory structure, etc.

Configuration Items
- Sw consists of many items (some artifact)
- In CM some identified items are placed under CM control
- Changes to these are then tracked
- Periodically, system is built using those items and baselines are established
- Baseline = logical state of the system and all its items; is a reference point

Version and access control
- Key issues in CM
- Done primarily on source code through source code control systems, which also provide access control
- Allows older versions to be preserved and hence can undo changes
- Eg. CVS and VSS are commonly used, Clear case for large projects

Version and Access Control
- When programmer developing code – is in private area
- When code is made available to others, it goes in an access-controlled library
- For making changes to an item in library, it has to be checked out
- Changes made by checking-in the item – versioning is automatically done
- Final system is built from the library

Version/Access Control
- Generally both version and access control done through CM tools like VSS, SCCS
- Tools limit access to specified people - formal check in, check out procedures
- Automatic versioning done when a changed file is checked-in
- Check-in, check-out control may be restricted to a few people in a project

CM Process
- Defines the activities for controlling changes
- Main phases
  - CM Planning
  - Executing the CM process
  - CM audits

CM Planning
- Identify items to be placed under CM
- Define library structure for CM
- Define change control procedures
- Define access control, baselining, reconciliation, procedures
- Define release procedure
CM Audit
- During project execution CM procedures have to be followed (e.g. moving items between directories, naming, following change procedures, …)
- Process audit has to be done
- CM audit can also check if items are where they should be

Summary – CM
- CM is about managing the different items in the product, and changes in them
- Developing a CM plan at the start is the key to successful CM
- CM procedures have to be followed, audits have to be performed
- Requires discipline and tools

Requirements Change Management Process

Background
- Requirements change: At any time during the development
- Changes impact the work products and the various configuration items
- Uncontrolled changes can have a huge adverse impact on project in cost/schedule
- Changes have to be allowed, but in a controlled manner

A Change Mgmt Process
- Log the changes
- Perform impact analysis on the work products and items
- Estimate impact on effort and schedule
- Review impact with stakeholders
- Rework the work products/items

Changes
- Change often triggered by change request
- Change req log keeps a record of requests
- Impact analysis for a change request involves identifying the changes needed to diff items, and the nature of change
- The impact of changes on the project is reviewed to decide whether to go ahead
- Cumulative changes also often tracked

Process Management Process

Software Process Improvement
- To improve the process, an org must understand the current process
- Requires process be properly documented
- Properly executed on projects
- Data is collected from projects to understand the performance of process on projects
- Changes to process are best made in small increments
Chapter 3

Agile Development

Software Engineering: A Practitioner’s Approach, 7/e
by Roger S. Pressman

For non-profit educational use only

The Manifesto for Agile Software Development

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

Kent Beck et al.

What is “Agility”?

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

Yielding ...

- Rapid, incremental delivery of software

Agility and the Cost of Change

An Agile Process

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple “software increments”
- Adapts as changes occur

Agility Principles - I

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
Agility Principles - II

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Human Factors

- the process molds to the needs of the people and team, not the other way around
- key traits must exist among the people on an agile team and the team itself:
  - Competence
  - Common focus
  - Collaboration
  - Decision-making ability
  - Fuzzy problem-solving ability
  - Mutual trust and respect
  - Self-organization

Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck
- XP Planning
  - Begins with the creation of “user stories”
  - Agile team assesses each story and assigns a cost
  - Stories are grouped to focus on a deliverable increment
  - A commitment is made on delivery date
  - After the first increment “project velocity” is used to help outline subsequent delivery dates for other increments

Extreme Programming (XP)

- XP Design
  - Follows the KE principle
  - Encourages the use of CRC cards (see Chapter 9)
  - For difficult design problems, suggests the creation of “spike solution” or design prototype
  - Encourages “refactoring” on the iterative refinement of the internal program design
- XP Coding
  - Recommends the construction of a unit test for a story before coding commences
  - Encourages “pair programming”
- XP Testing
  - All unit tests are executed daily
  - “Acceptance tests” are defined by the customer and executed to assess customer-visible functionality

Adaptive Software Development

- Originally proposed by Jim Highsmith
- ASD — distinguishing features
  - Mission-driven planning
  - Component-based focus
  - Uses “time-boxing” (See Chapter 24)
  - Explicit consideration of risks
  - Emphasizes collaboration for requirements gathering
  - Emphasizes “learning” throughout the process

Dynamic Systems Development Method

- Promoted by the DSDM Consortium (www.dsdm.org)
- DSDM—distinguishing features
  - Similar in most respects to XP and other ASD
  - Nine guiding principles
    - Adhere to formal development of the system
    - DSDM teams must be managed in a matrix structure
    - The focus is on frequent delivery of products
    - Manual for business purposes is the essential tool for acceptance of the system
    - Iterative and incremental development is necessary for convergence on an accurate business solution
    - Measurement of change development is essential
    - Requirements are validated at a high level
    - Analysis integrated throughout the process

Dynamic Systems Development Method

- DSDM Life Cycle (with permission of the DSDM consortium)
Scrum

- Originally proposed by Schwaber and Beedle
- Scrum—distinguishing features
  - Development work is partitioned into "packets"
  - Testing and documentation are ongoing as the product is constructed
  - Work occurs in "sprints" and is derived from a "backlog" of existing requirements
  - Meetings are very short and sometimes conducted without chairs
  - "Stories" are delivered to the customer with the time-box allocated

Crystal

- Proposed by Cockburn and Highsmith
- Crystal—distinguishing features
  - Actually a family of process models that allow "maneuverability" based on problem characteristics
  - Face-to-face communication is emphasized
  - Suggests the use of "reflection workshops" to review the work habits of the team

Feature Driven Development

- Originally proposed by Peter Coad et al.
- FDD—distinguishing features
  - Emphasis is on defining "features" in a feature class view.
  - A feature class view is defined and "plan by feature" is conducted
  - Design and construction merge in FDD

Agile Modeling

- Originally proposed by Scott Ambler
- Suggests a set of agile modeling principles
  - Model with a purpose
  - Use multiple models
  - Travel light
  - Content is more important than representation
  - Know the models and the tools you use to create them
  - Adapt locally

Chapter 4

- Principles that Guide Practice

**Software Engineering Knowledge**

- You often hear people say that software development knowledge has a 3-year half-life: half of what you need to know today will be obsolete within 3 years. In the domain of technology-related knowledge, that's probably about right. But there is another kind of software development knowledge—a kind that I think of as "software engineering principles"—that does not have a three-year half-life. These software engineering principles are likely to serve a professional programmer throughout his or her career.

Steve McConnell

**Software Practice Core Principles**

- Software exists to provide value to its users
- Keep it simple stupid (KISS)
- Clear vision is essential to the success of any software project
- Always specify, design, and implement knowing that someone else will have to understand what you have done to carry out his or her tasks
- Be open to future changes, don't code yourself into a corner
- Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems that require them
- Placing clear complete thought before any action almost always produces better results
**Principles that Guide Process**

- **Principle #1. Be agile.** Whether the process model you choose is prescriptive or agile, the basic tenet of agile development should govern your approach.
- **Principle #2. Focus on quality at every step.** The exit condition for every process activity, action, and task should focus on the quality of the work product that has been produced.
- **Principle #3. Be ready to adapt.** Process is not a religious experience and degrees have no place in it. When necessary, adapt your approach to constraints imposed by the problem, the people, and the project itself.
- **Principle #4. Build an effective team.** Software engineering process and practice are important, but the bottom line is people. Build a self-organizing team that has mutual trust and respect.

---

**Principles that Guide Practice**

- **Principle #5. Build software that exhibits effective modularity.** Separation of concerns (Principle #1) establishes a philosophy for software. Modularity provides a mechanism for realizing the philosophy.
- **Principle #6. Look for patterns.** Brad Appleton (App06) suggests that “The goal of patterns within the software community is to create a body of literature to help software developers resolve recurring problems encountered throughout all of software development.”
- **Principle #7. When possible, replace the problem and its solution from a number of different perspectives.**
- **Principle #8. Remember that someone will maintain the software.**

---

**Principles that Guide Framework Activities**

- **Communication Principles**
- **Planning Principles**
- **Modeling Principles**
- **Agile models**
- **Requirements models**
- **Design models**
- **Construction Principles**
- **Coding principles and concepts**
- **Testing principles and concepts**
- **Deployment Principles**

---

**Principles that Guide Framework Activities**

**Communication Principles**

- **Principle #1. Listen.** Try to focus on the speaker’s words, rather than formulating your response to those words.
- **Principle #2. Prepare before you communicate.** Spend the time to understand the problem before you meet with others.
- **Principle #3. Someone should facilitate the activity.** Every communication meeting should have a leader (a facilitator) to keep the conversation moving in a productive direction; (2) to mediate any conflict that does occur, and (3) to ensure that other principles are followed.
- **Principle #4. Face-to-face communication is best.** But it usually works better when some other representation of the relevant information is present.

---

**Planning Principles**

- **Principle #1. Understand the scope of the project.** It’s impossible to use a roadmap if you don’t know where you’re going. Scope provides the software team with a destination.
- **Principle #2. Involve the customer in the planning activity.** The customer defines priorities and establishes project constraints.
- **Principle #3. Recognize that planning is iterative.** A project plan is never engraved in stone. As work begins, it’s very likely that things will change.
- **Principle #4. Estimate based on what you know.** The intent of estimation is to provide an indication of effort, cost, and task duration, based on the team’s current understanding of the work to be done.

---

**Planning Principles**

- **Principle #5. Consider risk as you define the plan.** If you’ve identified risks that have high impact and high probability, contingency planning is necessary.
- **Principle #6. Be realistic.** People don’t work 100 percent of every day.
- **Principle #7. Adjust granularity as you define the plan.** Granularity refers to the level of detail that is introduced as a project plan is developed.
- **Principle #8. Define how you intend to ensure quality.** The plan should identify how the software team intends to ensure quality.
- **Principle #9. Describe how you intend to accommodate change.** Even the best planning can be obviated by uncontrolled change.
- **Principle #10. Track the plan frequently and make adjustments as required.** Software projects fail behind schedule one day at a time.
Modeling Principles

- General - Agile Models
  In software engineering work, two classes of models can be created:
  - Requirements models (also called analysis models) represent the customer requirements by depicting the software in three different domains: the information domain, the functional domain, and the behavioral domain.
  - Design models represent characteristics of the software that help practitioners to construct it effectively: the architecture, the user interface, and component-level detail.

Agile Modeling Principles

- Principle #1. The primary goal of the software team is to build software, not create models.
- Principle #2. Travel light—don’t create more models than you need.
- Principle #3. strive to produce the simplest model that will describe the problem or the software.
- Principle #4. Build models in a way that makes them amenable to change.
- Principle #5. Be able to state an: explicit purpose for each model that is created.
- Principle #6. Adapt the models you develop to the system of interest.
- Principle #7. Try to build useful models, not forget about building perfect representations.
- Principle #8. Don’t become dependent about the syntax of the model. If it communicates content successfully, representation is secondary.
- Principle #9. If your instincts tell you a model isn’t right even though it seems okay on paper, you probably have reason to be concerned.
- Principle #10. Get feedback as soon as you can.

Requirements Modeling Principles

- Principle #1. The information domain of a problem must be represented and understood.
- Principle #2. The functions that the software performs must be defined.
- Principle #3. The behavior of the software (as a consequence of external events) must be represented.
- Principle #4. The models that depict information, function, and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.
- Principle #5. The analysis task should move from essential information toward implementation detail.

Design Modeling Principles

- Principle #1. Design should be traceable to the requirements model.
  - Principle #2. Always consider the architecture of the system to be built.
  - Principle #3. Design of data is as important as design of processing functions.
  - Principle #4. Internal and external interfaces must be designed with care.
  - Principle #5. User interface design should be tuned to the needs of the end-user. However, in every case, it should stress ease of use.
  - Principle #6. Component-level design should be functionally independent.
  - Principle #7. Components should be loosely coupled to one another and to the external environment.
  - Principle #8. Design representations (models) should be easily understandable.
  - Principle #9. The design should be developed iteratively. With each iteration, the design should evolve for greater simplicity.

Construction Principles

- The construction activity encompasses a set of coding and testing tasks that lead to operational software that is ready for delivery to the customer or end-user.
- Coding principles and concepts are closely aligned programming style, programming languages, and programming methods.
- Testing principles and concepts lead to the design of tests that systematically uncover different classes of errors and to do so with a minimum amount of time and effort.

Coding: Preparation Principles

- Before you write one line of code, be sure you:
  - Understand the problem you're trying to solve.
  - Understand basic design principles and concepts.
  - Pick a programming language that meets the needs of the software to be built and the environment in which it will operate.
  - Select a programming environment that provides tools that will make your work easier.
  - Create a set of unit tests that will be applied once the component your code is completed.

Coding: Programming Principles

- As you begin writing code, be sure you:
  - Comply with the style guide you've defined.
  - Consider the use of parallel programming.
  - Select data structures that will meet the needs of the design.
  - Understand the software architecture and create interfaces that are consistent with it.
  - Keep conditional logic as simple as possible.
  - Create nested loops in a way that makes them easy to testable.
  - Select meaningful variable names and follow other local coding standards.
  - Write code that is self-documenting.
  - Create a visual layout (e.g., indentation and blank lines) that aids understanding.

Coding: Validation Principles

- After you've completed your first coding pass, be sure you:
  - Conduct a code walkthrough when appropriate.
  - Correct unit tests and correct errors you've uncovered.
  - Refactor the code, if the process of restructuring existing computer code without changing its external behavior.

Coding: Testing Principles

- Al Davis [Dav95] suggests the following:
  - Principle #1. All tests should be traceable to customer requirements.
  - Principle #2. Tests should be planned long before testing begins.
  - Principle #3. The Paradox principle applies to software testing.
  - Principle #4. Testing should begin “in the small” and progress toward testing “in the large.”
  - Principle #5. Exhaustive testing is not possible.
Deployment Principles

- **Principle #1.** Customer expectations for the software must be managed. Too often, the customer expects more than the team has promised to deliver, and disappointment occurs immediately.
- **Principle #2.** A complete delivery package should be assembled and tested.
- **Principle #3.** A support regime must be established before the software is delivered. An end-user expects responsiveness and accurate information when a question or problem arises.
- **Principle #4.** Appropriate instructional materials must be provided to end-users.
- **Principle #5.** Buggy software should be fixed first, delivered later.