UNIT – II

Conventional and Modern Software Management: The principles of conventional software Engineering, principles of modern software management, transitioning to an iterative process.
Life cycle phases: Engineering and Production stages, Inception, Elaboration, Construction, Transition Phases.

4. CONVENTIONAL AND MODERN SOFTWARE MANAGEMENT

4.1 THE PRINCIPLES OF CONVENTIONAL SOFTWARE ENGINEERING

1.**Make quality** Quality must be quantified and mechanisms put into place to motivate its achievement

2.**High-quality software is possible**. Techniques that have been demonstrated to increase quality include involving the customer, prototyping, simplifying design, conducting inspections, and hiring the best people

3.**Give products to customers early**. No matter how hard you try to learn users' needs during the requirements phase, the most effective way to determine real needs is to give users a product and let them play with it

4.**Determine the problem before writing the requirements**. When faced with what they believe is a problem, most engineers rush to offer a solution. Before you try to solve a problem, be sure to explore all the alternatives and don't be blinded by the obvious solution

5.**Evaluate design alternatives**. After the requirements are agreed upon, you must examine a variety of architectures and algorithms. You certainly do not want to use" architecture" simply because it was used in the requirements specification.

6.**Use an appropriate process model**. Each project must select a process that makes ·the most sense for that project on the basis of corporate culture, willingness to take risks, application area, volatility of requirements, and the extent to which requirements are well understood.

7.**Use different languages for different phases**. Our industry's eternal thirst for simple solutions to complex problems has driven many to declare that the best development method is one that uses the same notation through-out the life cycle.

8.**Minimize intellectual distance**. To minimize intellectual distance, the software's structure should be as close as possible to the real-world structure

9.**Put techniques before tools**. An undisciplined software engineer with a tool becomes a dangerous, undisciplined software engineer

10.**Get it right before you make it faster**. It is far easier to make a working program run faster than it is to make a fast program work. Don't worry about optimization during initial coding

11.**Inspect code**. Inspecting the detailed design and code is a much better way to find errors than testing

12.**Good management is more important than good technology**. Good management motivates people to do their best, but there are no universal "right" styles of management.

13.**People are the key to success.** Highly skilled people with appropriate experience, talent, and training are key.

14.**Follow with care**. Just because everybody is doing something does not make it right for you. It may be right, but you must carefully assess its applicability to your environment.

15.**Take responsibility**. When a bridge collapses we ask, "What did the engineers do wrong?" Even when software fails, we rarely ask this. The fact is that in any engineering discipline, the best methods can be used to produce awful designs, and the most antiquated methods to produce elegant designs.

16.**Understand the customer's priorities**. It is possible the customer would tolerate 90% of the functionality delivered late if they could have 10% of it on time.

17.**The more they see, the more they need**. The more functionality (or performance) you provide a user, the more functionality (or performance) the user wants.

18. **Plan to throw one away**. One of the most important critical success factors is whether or not a product is entirely new. Such brand-new applications, architectures, interfaces, or algorithms rarely work the first time.
19. **Design for change**. The architectures, components, and specification techniques you use must accommodate change.
20. **Design without documentation is not design**. I have often heard software engineers say, **''I** have finished the design. All that is left is the documentation. "
21. **Use tools, but be realistic.** Software tools make their users more efficient.
22. **Avoid tricks**. Many programmers love to create programs with tricks constructs that perform a function correctly, but in an obscure way. Show the world how smart you are by avoiding tricky code
23. **Encapsulate.** Information-hiding is a simple, proven concept that results in software that is easier to test and much easier to maintain.
24. **Use coupling and cohesion**. Coupling and cohesion are the best ways to measure software's inherent maintainability and adaptability
25. **Use the McCabe complexity measure**. Although there are many metrics available to report the inherent complexity of software, none is as intuitive and easy to use as Tom McCabe's

26.**Don't test your own software**. Software developers should never be the primary testers of their own software.

27.**Analyze causes for errors**. It is far more cost-effective to reduce the effect of an error by preventing it than it is to find and fix it. One way to do this is to analyze the causes of errors as they are detected

28.**Realize that software's entropy increases**. Any software system that undergoes continuous change will grow in complexity and will become more and more disorganized

29.**People and time are not interchangeable**. Measuring a project solely by person-months makes little sense

30.**Expect excellence**. Your employees will do much better if you have high expectations for them.

4.2 THE PRINCIPLES OF MODERN SOFTWARE MANAGEMENT

Top 10 principles of modern software management are. (The first five, which are the main themes of my definition of an iterative process, are summarized in Figure 4-1.)

1. **Base the process on an *architecture-first approach.*** This requires that a demonstrable balance be achieved among the driving requirements, the architecturally significant design decisions, and the life-cycle plans before the resources are committed for full-scale development.
2. **Establish an *iterative life-cycle process* that confronts risk early**. With today's sophisticated software systems, it is not possible to define the entire problem, design the entire solution, build the software, and then test the end product in sequence. Instead, an iterative process that refines the problem understanding, an effective solution, and an effective plan over several iterations encourages a balanced treatment of all stakeholder objectives. Major risks must be addressed early to increase predictability and avoid expensive downstream scrap and rework.
3. **Transition design methods to emphasize *component-based development.*** Moving from a line-of-code mentality to a component-based mentality is necessary to reduce the amount of human-generated source code and custom development.

4. **Establish a *change management environment.*** The dynamics of iterative development, including concurrent workflows by different teams working on shared artifacts, necessitates objectively controlled baselines.

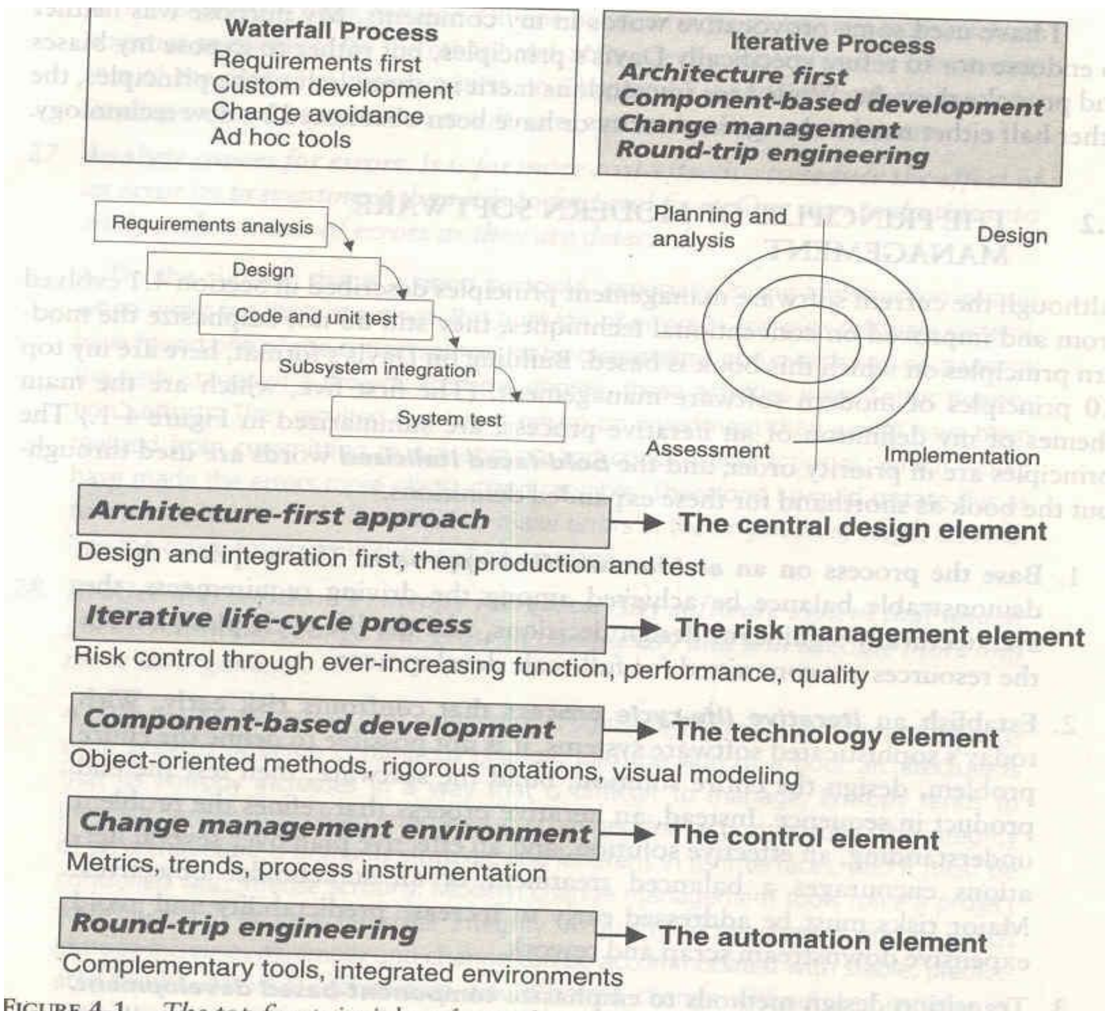
Waterfall Process
Requirements first
Custom development
Change avoidance
Ad hoc tools
Iterative Process
Architecture first
Component-based development
Change management
Round-trip engineering
Requirements analysis
Design
Code and unit test
Subsystem integration
System test
Planning and analysis
Design
Assessment
Implementation
Architecture-first approach
The central design element
Design and integration first, then production and test
Iterative life-cycle process
The risk management element
Risk control through ever-increasing function, performance, quality
Component-based development
The technology element
Object-oriented methods, rigorous notations, visual modeling
Change management environment
The control element
Metrics, trends, process instrumentation
Round-trip engineering
The automation element
Complementary tools, integrated environments

FIGURE 4-1. *The top five principles of a modern process*

5. **Enhance change freedom through tools that support round-trip engineering**. Round-trip engineering is the environment support necessary to automate and synchronize engineering information in different formats(such as requirements specifications, design models, source code, executable code, test cases).
6. **Capture design artifacts in rigorous, model-based notation.** A model based approach (such as UML) supports the evolution of semantically rich graphical and textual design notations.
7. **Instrument the process for objective quality control and progress assessment**. Life-cycle assessment of the progress and the quality of all intermediate products must be integrated into the process.
8. **Use a demonstration-based approach to assess intermediate artifacts.**
9. **Plan intermediate releases in groups of usage scenarios with evolving levels of detail.** It is essential that the software management process drive toward early and continuous demonstrations within the operational context of the system, namely its use cases.
10. **Establish a configurable process that is economically scalable.** No single process is suitable for all software developments.

Table 4-1 maps top 10 risks of the conventional process to the key attributes and principles of a modern process

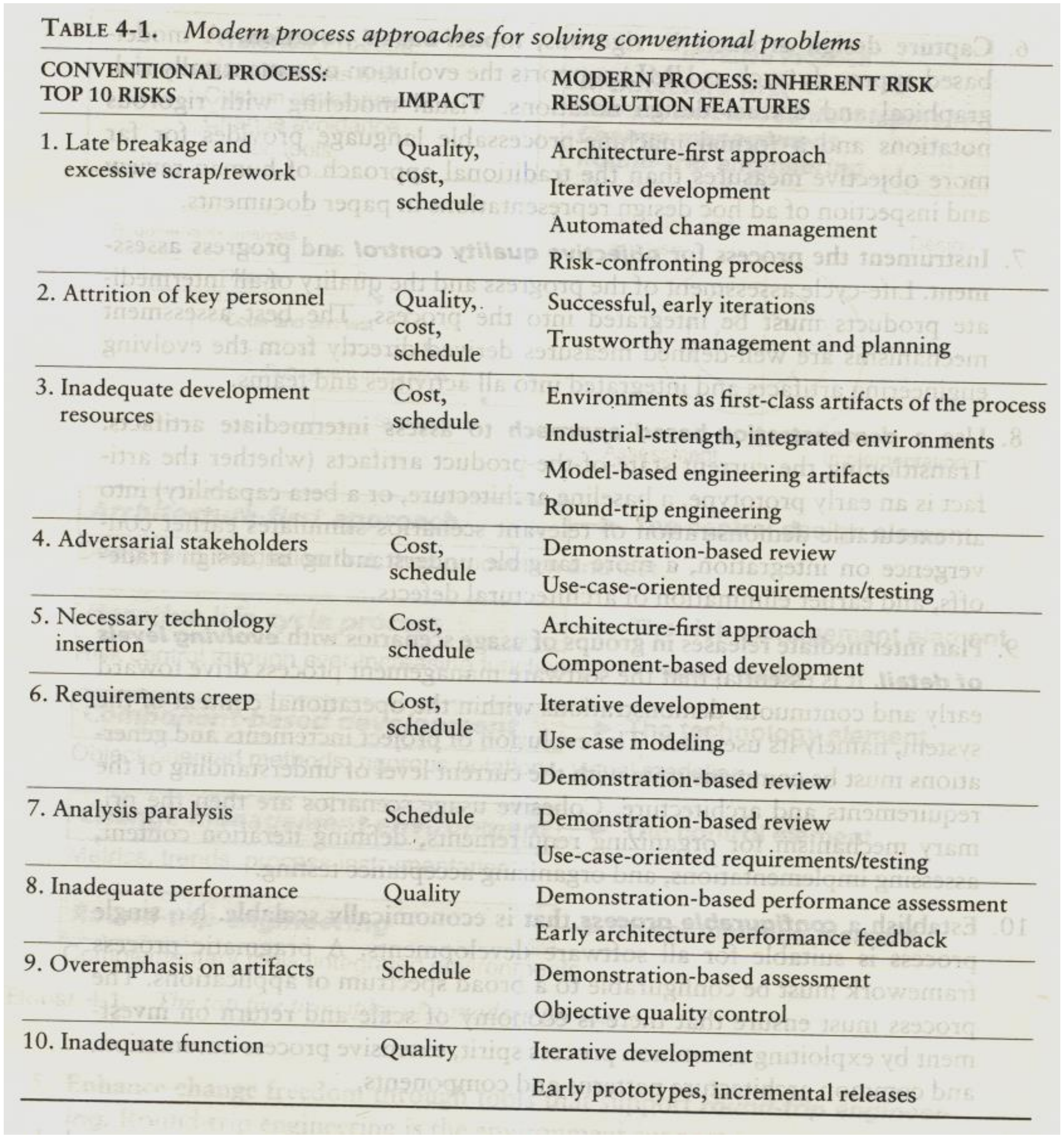

TABLE 4-1. *Modern process approaches for solving conventional problems*

CONVENTIONAL PROCESS: TOP 10 RISKS	IMPACT	MODERN PROCESS: INHERENT RISK RESOLUTION FEATURES
1. Late breakage and excessive scrap/rework	Quality, cost, schedule	Architecture-first approach Iterative development Automated change management Risk-confronting process
2. Attrition of key personnel	Quality, cost, schedule	Successful, early iterations Trustworthy management and planning
3. Inadequate development resources	Cost, schedule	Environments as first-class artifacts of the process Industrial-strength, integrated environments Model-based engineering artifacts Round-trip engineering
4. Adversarial stakeholders	Cost, schedule	Demonstration-based review Use-case-oriented requirements/testing
5. Necessary technology insertion	Cost, schedule	Architecture-first approach Component-based development
6. Requirements creep	Cost, schedule	Iterative development Use case modeling Demonstration-based review
7. Analysis paralysis	Schedule	Demonstration-based review Use-case-oriented requirements/testing
8. Inadequate performance	Quality	Demonstration-based performance assessment Early architecture performance feedback
9. Overemphasis on artifacts	Schedule	Demonstration-based assessment Objective quality control
10. Inadequate function	Quality	Iterative development Early prototypes, incremental releases

4.3 TRANSITIONING TO AN ITERATIVE PROCESS

Modern software development processes have moved away from the conventional waterfall model, in which each stage of the development process is dependent on completion of the previous stage.

The economic benefits inherent in transitioning from the conventional waterfall model to an iterative development process are significant but difficult to quantify. As one benchmark of the expected economic impact of process improvement, consider the process exponent parameters of the COCOMO II model. (Appendix B provides more detail on the COCOMO model) This exponent can range from 1.01 (virtually no diseconomy of scale) to 1.26 (significant diseconomy of scale). The parameters that govern the value of the process exponent are application precedentedness, process flexibility, architecture risk resolution, team cohesion, and software process maturity.

The following paragraphs map the process exponent parameters of CO COMO II to my top 10 principles of a modern process.

- **Application precedentedness**. Domain experience is a critical factor in understanding how to plan and execute a software development project. For unprecedented systems, one of the key goals is to confront risks and establish early precedents, even if they are incomplete or experimental. This is one of the primary reasons that the software industry has moved to an ***iterative life-cycle process.*** Early iterations in the life cycle establish precedents from which the product, the process, and the plans can be elaborated in ***evolving levels*** **of** ***detail.***
- **Process flexibility**. Development of modern software is characterized by such a broad solution space and so many interrelated concerns that there is a paramount need for continuous incorporation of changes. These changes may be inherent in the problem understanding, the solution space, or the plans. Project artifacts must be supported by efficient ***change management*** commensurate with project needs. A ***configurable process*** that allows a common framework to be adapted across a range of projects is necessary to achieve a software return on investment.
- **Architecture risk resolution**. ***Architecture-first*** development is a crucial theme underlying a successful iterative development process. A project team develops and stabilizes architecture before developing all the components that make up the entire suite of applications components. An ***architecture-first*** and ***component-based development approach*** forces the infrastructure, common mechanisms, and control mechanisms to be elaborated early in the life cycle and drives all component make/buy decisions into the architecture process.
- **Team cohesion**. Successful teams are cohesive, and cohesive teams are successful. Successful teams and cohesive teams share common objectives and priorities. Advances in technology (such as programming languages, UML, and visual modeling) have enabled more rigorous and understandable notations for communicating software engineering information, particularly in the requirements and design artifacts that previously were ad hoc and based completely on paper exchange. These ***model-based*** formats have also enabled the ***round-trip engineering*** support needed to establish change freedom sufficient for evolving design representations.
- **Software process maturity**. The Software Engineering Institute's Capability Maturity Model (CMM) is a well-accepted benchmark for software process assessment. One of key themes is that truly mature processes are enabled through an integrated environment that provides the appropriate level of automation to instrument the process for ***objective quality control.***

Important questions

1.	***Explain briefly Waterfall model. Also explain Conventional s/w management performance?***
2.	***Define Software Economics. Also explain Pragmatic s/w cost estimation?***
3.	***Explain Important trends in improving Software economics?***
4.	***Explain five staffing principal offered by Boehm. Also explain Peer Inspections?***
5..	***Explain principles of conventional software engineering?***
6.	***Explain briefly principles of modern software management***

5. Life cycle phases

Characteristic of a successful software development process is the well-defined separation between "research and development" activities and "production" activities. Most unsuccessful projects exhibit one of the following characteristics:

- An overemphasis on research and development
- An overemphasis on production.

Successful modern projects-and even successful projects developed under the conventional process-tend to have a very well-defined project milestone when there is a noticeable transition from a research attitude to a production attitude. Earlier phases focus on achieving functionality. Later phases revolve around achieving a product that can be shipped to a customer, with explicit attention to robustness, performance, and finish.
A modern software development process must be defined to support the following:

- Evolution of the plans, requirements, and architecture, together with well defined synchronization points
- Risk management and objective measures of progress and quality
- Evolution of system capabilities through demonstrations of increasing functionality

5.1 ENGINEERING AND PRODUCTION STAGES

To achieve economies of scale and higher returns on investment, we must move toward a software manufacturing process driven by technological improvements in process automation and component-based development. Two stages of the life cycle are:

1. The **engineering stage**, driven by less predictable but smaller teams doing design and synthesis activities
2. The **production stage**, driven by more predictable but larger teams doing construction, test, and deployment activities