UNIT – I

CONVENTIONAL SOFTWARE MANAGEMENT

- The best thing about software is its flexibility. It can be used to programmed any thing.
- The worst thing about software is also its flexibility. The anything characteristic has made it difficult to plan, monitor, and control software development.
- The analysis in software development states that, the success rate of software project is very low. The following are three main reasons for low success rate.

a) Software development is highly unpredictable. Only 10% software projects are delivered successfully within budget and time.

b) Management discipline is more discriminator in success or failure than technology advances.

c) The level of software scrap and rework is indicative of an immature process.

The waterfall model:

- In software engineering the waterfall model is conventional software development process and it treated as benchmark of that process.

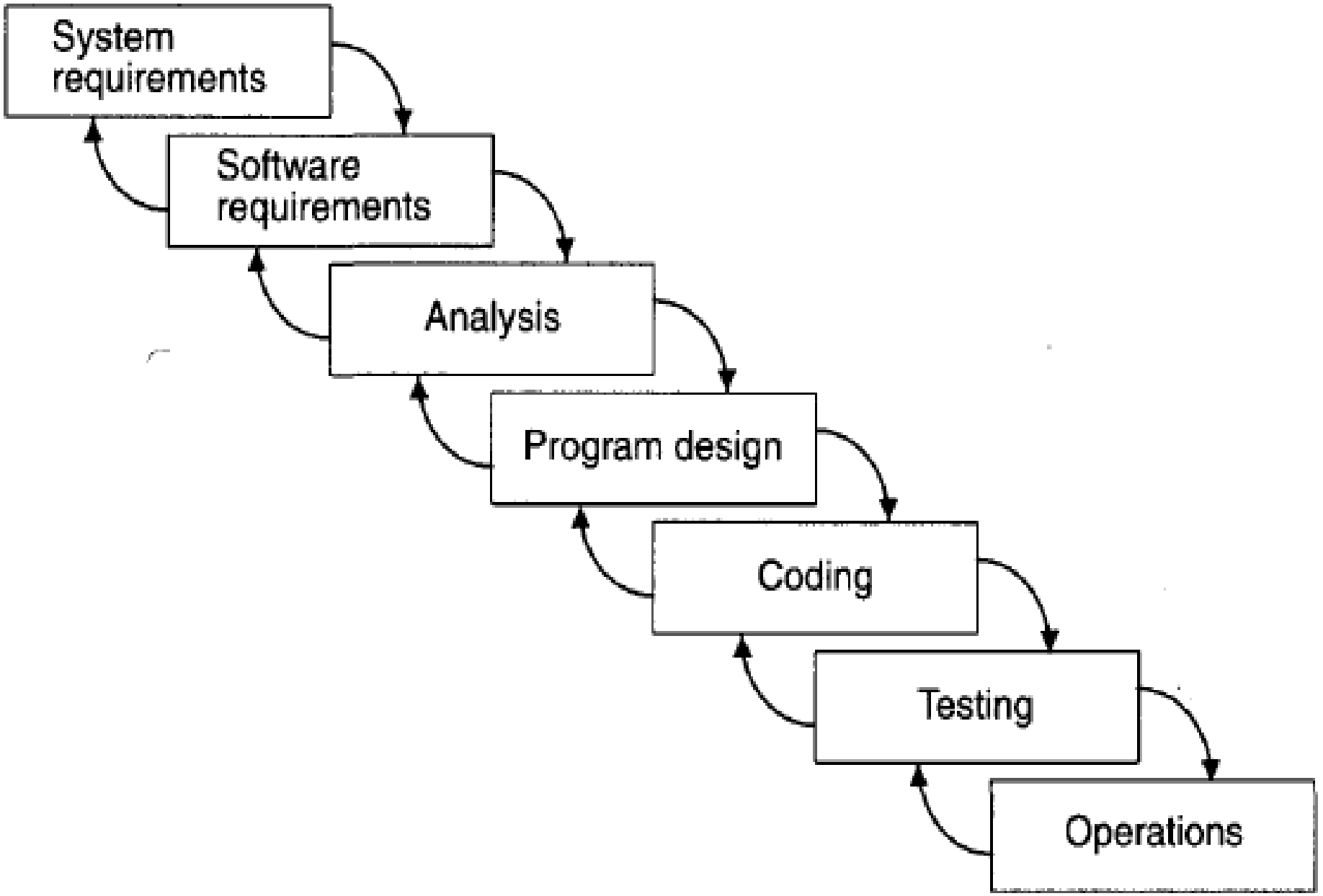
Waterfall in theory: The waterfall model made three primary points.

a) There are two essential steps common to the development of computer programs: Analysis and coding.

b) To control intellectual freedom associated with software development, we introduce system requirements, software requirements, program design and testing.

c) The basic framework in the waterfall model is risky and invites failure.

The following diagram shows waterfall model profile and the basic steps.


System requirements
Software requirements
Analysis
Program design
Coding
Testing
Operations

• The following are five improvements to the basic waterfall process to eliminate the development risks.

1)Program design comes first: We should take-up preliminary program design as the first step even before the requirements and analysis phases.

•At this point the program designer assures that the software will not be fail because of storage, timing and data flux.

•When we performing analysis as the next phase, the program designer analyze operational constraints and senses the consequences.

- Due to the early program design phase we should easily understand the total resources allocated to the project are insufficient, developing operational design is right or wrong and etc.
- we implement the early program design by follow the following points.

a)Start the design process with program designer not with analysts or programmers.

b)Design, define, and allocate data processing modes even at being wrong. Allocate processing functions, design DB

Allocate execution time, define interfaces and operating system, describe input and output process, define preliminary operating procedures.

c) Write an overview document that is understandable, informative, and current so that every worker on the project can gain an elemental understanding of the system.

2) Document the design: We know that document design is the required a lot for software programs.

- the programmers, analysts, designers, are mostly depending on documentation to finish their work.

- Documentation is important for the following reasons;

a)The software designers communicate with interfacing designers, managers, and with customers through documentation.

b)During early phases documentation is the design.

c)The value of documentation is to support later modifications by separate test team, separate maintenance team and operational personnel who are not software literate.

3) Do it twice: The software delivered to the customer for operational purpose is actually the second version, after considering critical design and operational issues.

- In first version of software development, the team have a special broad competence where they can quickly sense trouble spots in the design, model them, model alternatives.

4) Plan, control, and monitor testing: The combination of manpower, computer time, and management is called the test phase. This phase has greater risk in terms of cost and schedule

- The following are the important things in the test phase;

a)Employ a team of test specialists who were not responsible for the original design.

b)Employ visual inspections to spot the obvious errors like dropped minus, missing factors of two, jump to wrong address etc.

c)Test every logic path.

d)Employ the final checkout on the target computer.

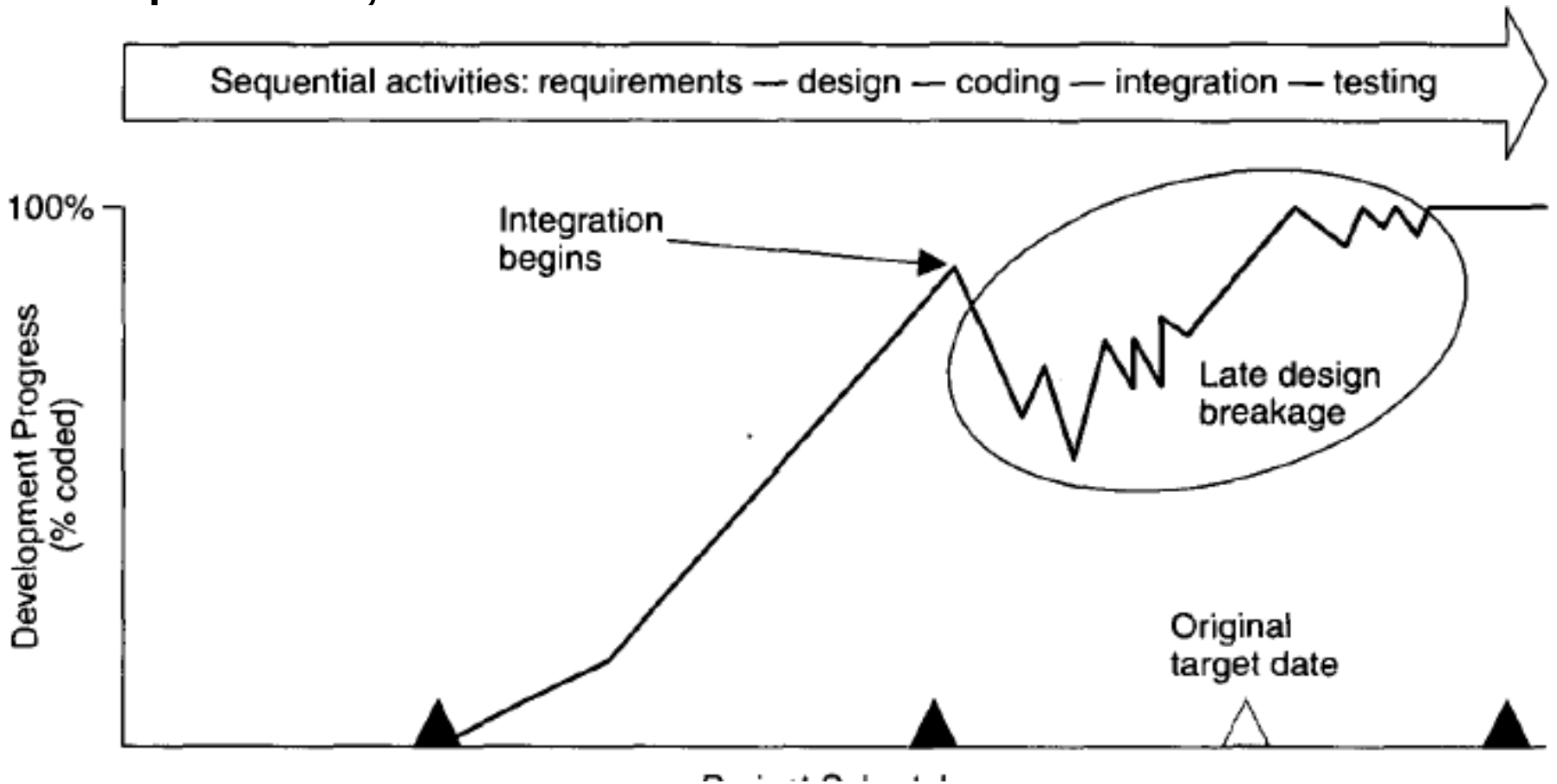
5) Involve the customer: some times software design going to wide interpretations even after previous agreement. So it is important to involve the customer in a formal way so that he has committed himself at earlier points before final delivery.

- Insight, judgment and commitment of customer can boost the project development process. It may done at preliminary software review following the preliminary program design step, sequence of final software acceptance review.

Waterfall model in Practice: Some software projects still follow the conventional software management process. Here we summarize the characteristics of conventional model.

- Protracted integration and late design breakage.
- Late risk resolution.
- Requirements driven functional decomposition.
- Adversarial stakeholder relationships.
- Focus on documents and review meetings.

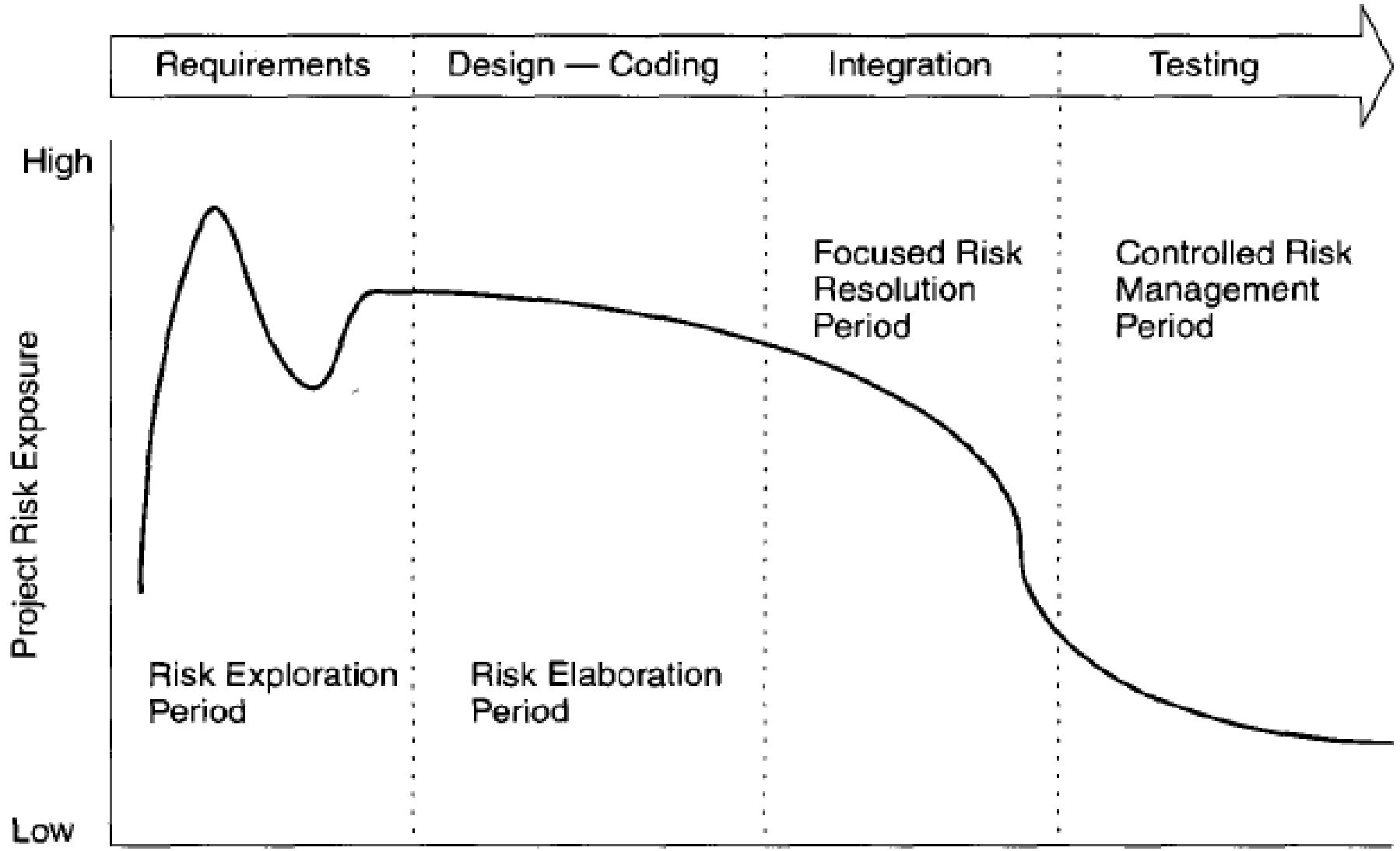
Protracted integration and late design breakage: the following diagram shows development progress versus time. The following sequence of issues is common in this phase;



- Early success via paper designs and through briefings.
- Commitment to code late in life cycle.
- Integration difficulties due to unforeseen implementation issues and interface ambiguities.
- Heavy budget and schedule pressure to get the system working.
- Late response of non-optimal fixes, with no time for design.
- A very delicate, unmaintainable product delivered late.

- In conventional approach the use of immature languages and technologies is difficult to understand the software project design and to change it in future.
- In conventional model the entire system was designed on paper, and implemented all at once, then integrated. Here we perform system testing at the end of the process to check the fundamental architecture is good or not.
- **Late risk resolution:** A serious problem in waterfall model was lack of early risk resolution.

• The following diagram shows risk profile for conventional model projects covers four distinct periods.



- At the early stage risk is defined as missing a cost, schedule, feature or quality. At this stage requirement being specified so risk exposure was unpredictable.
- After the design concept was available even on paper, risk exposure stabilized.
- In the next step, after the system was coded some of the individual component risks was resolved. When begin the integration the real system level risks are touchable.

- **Requirement driven functional decomposition:** We know that software development process has been requirements driven. We should present precise requirements definition and then implement exactly those requirements.
- It mean that the requirements completely unambiguous before development activity begins.
- We should treat all the requirements are equally important, but it is not possible in real world. So the specification of requirements is difficult and important part in software development

- Another issue in conventional process is requirements are specified in functional manner. In waterfall model requirements are allocated to the components.
- The decomposition is very difficult based on object oriented design and use of existing components.

Adversarial stakeholder relationships: the conventional process cause adversarial stakeholder relationships, in large part because of the difficulties of requirements specification and the exchange of information only through paper documents

and engineering information in adhoc formats. The following sequence of things are happened in contractual software development.

1. The contractor prepared a draft contract deliverable document that captured an intermediate artifact and delivered it to the customer for approval.

2.The customer as expected to provide comments.

3.The contractor incorporated these comments and submitted a final version for approval.

- This one shot review process is highly sensitive on the part of customers and contractors. The exchanges of paper reviews are intolerable.
- **Focus on documents and review meetings:** the conventional software development process focused on producing various documents that attempted to describe the software product.
- Here formal meetings were conducted to exchange specific documents. The developers produce tons of papers to describe the project progress to the customers rather than to reduce risk and produce quality software.

- Most review meetings has low engineering values and high cost in terms of effort and schedule involved in their preparation and conduct.

CONVENTIONAL SOFTWARE MANAGEMENT PERFORMANCE

- Barry Bhoom describes the objective characterization of state of software development. Many of the metrics describe fundamental economic relationships that resulted from conventional software process practiced over the long period.

1. Finding and fixing a software problem after delivery costs 100 times more than finding and fixing the problem in early design phase.
2. We can compress software development schedules 25% of normal, but no more – one reason for this statement is N% reduction in schedule would require M% increase in personnel resources. Sometimes an optimal schedule be extended almost arbitrary, depending on people, could performed in much longer times with fewer resources

3. For every rupee we spend on development, we will spend two rupees on maintenance – This is known as iron law of software development. When we build long lived products which has commercial version upgrades or custom software system may require twice money spent on maintenance rather than development lifecycle.

4. Software development and maintenance costs are primarily a function of the number of sources lines of code – this metric is primarily predominance of custom software development

5. Variations among people account for the biggest differences in software productivity – this metric is subject to both overhype and under hype of people. We don't know why the software projects are succeeded or failed. It totally depending on the quality of people.

6. The overall ratio of software to hardware costs is still growing. In 1955 it was 15:85; in 1985 it was 85:15.

7. Only 15% of software development effort is devoted to programming – This is important indicator is need to balance software development activities. There were many activities besides the coding such as requirements management, design, testing, planning, project control, change management etc.

8. Software systems and products typically cost three times as much per SLOC as individual software programs. Software system products costs nine times as much – the exponential growth is called diseconomy of scale.

9. Walkthroughs catch 60% of the errors – walkthroughs and other human inspections are good for catching surface problems and style issues.

10. 80% of contribution comes from 20% of contributors – this is motherhood statement that is true across any engineering discipline. The following are fundamental postulates to rationale modern software management process work.

- 80% of engineering is consumed by 20% of equipment.
- 80% of errors caused by 20% of components.
- 80% of software const is consumed by 20% of components.
- 80% of progress made by 20% people.

Software Economics

Most software cost models can be described five parameters: Size, Process, Personnel, Environment, and required Quality

1.The size of the end product is typically measured in terms of number of source lines or the number of functional points developed to achieve the required functionality.

2.The process used to produce the end product, the ability of the process is to avoid non value adding activities. (rework, bureaucratic delays, communications overhead)

3. The capability of software engineering personnel, and particularly their experience with the computer science issues and the applications domain issues of the project.

4. The environment which is made up of the tools and techniques available to support efficient software development and to automate the process.

5. The required quality of the product, including its features, performance, reliability and adaptability.

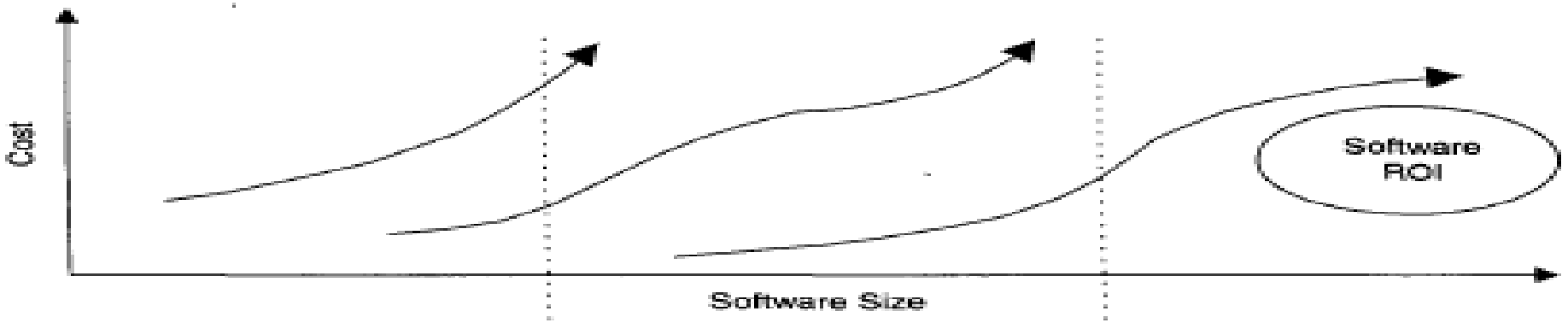
The following equation shows the relationship between size, process, personnel, environment, quality parameters and between the estimated cost.

$$\text{Effort} = (\text{personnel})\,(\text{Environment})\,(\text{Quality})\,(\text{Size}^{\text{Process}})$$

- Here one important feature of software economics is the relationship between effort and size exhibits is a diseconomy of scale.
- It indicates that more software we build it consumes more expensive per unit of the software.

- So the per-line cost of smaller applications is less than for the larger applications. This feature is used to managing interpersonal communications as the number of team members scales up.
- The following diagram shows three generations software economics in the form of basic technology advancement in tools, components, and processes
- Here the required level of quality and personnel are assumed to be constant.
-

Target objective: improved ROI



1960s–1970s	1980s–1990s	2000 and on
- Waterfall model - Functional design - Diseconomy of scale	- Process impovement - Encapsulation-based - Diseconomy of scale	- Iterative development - Component-based - Return on investment

Corresponding environment, size, and process technologies

Conventional	Transition	Modern Practices
Environments/tools: Custom	**Environment/tools:** Off-the-shelf, separate	**Environment/tools:** Off-the-shelf, integrated
Size: 100% custom	**Size:** 30% component-based 70% custom	**Size:** 70% component-based 30% custom
Process: Ad hoc	**Process:** Repeatable	**Process:** Managed/measured

Typical project performance

Predictably bad	Unpredictable	Predictable
Always: Over budget Over schedule	Infrequently: On budget On schedule	Usually: On budget On schedule

- The phases represents the life cycle of software business in the organization. So we can define the three generations of software development as follows;

1 Conventional: the software is developed in conventional manner between 1960 and 1970; here development is treated as craftsmanship.

- Organizations used custom tools, custom processes, and virtually custom components built in primitive languages.
- Here the performance of the project is highly predictable and cost, schedule and quality objectives were always underachieved.

2. Transition: this is the middle age of the software development which is between 1980 and 1990; at this time some software engineering principles are formed and organizations used repeatable and off the shelf tools.

- During this phase customer components are built in high level languages and some components are available commercially such as operating system, database management system, networking and graphical user interface etc.

- In this phase organizations achieve economy scale but due to the growth in complexity of applications the existing languages, techniques, and technologies are not sustained in longer time.

3. Modern practices: this phase of software development process started in year 2000 and later.

- In this phase software development is treated as a production. Here we use integrated automation environments, off the shelf components.

- The technologies for environment automation, size reduction and process improvement are not independent of one another. Now a day's balancing growth is achieved in all technologies.
- The transition and modern practices represents improved software economics but at the same time no guaranteed. The developers always realistic in software execution, use of new technologies compared with old history.

UNIT-II

Improving Software Economics

- Improvement in the software economics is not a difficult task, but to measure the improvement is highly difficult because there were inconsistent units of measures, disagreement among the experts, and so many issues.
- To improve the software economics we should consider the following five basic parameters.

1.Reducing the size or complexity what needs to be developed.

2. Improving the development process.

3. Using more skilled personnel and better teams.

4. Using better environments.

5. Trading off or backing of quality thresholds.

Reduction of software product size:

- The most considerable way to improve the affordability and return on investment is to produce a product that achieves the design goals with the minimum amount of human generated source material.

- To reduce the software product size, the developers may follow component based development model, object oriented technology, automatic code generators and so may tools.
- The use of high level languages such as Java, .NET, Visual Basic for software development focused on few lines of human generated source code.
- But the other aspect is, when size reducing technologies are used, they certainly generate human generated source lines, however increase computer process able executable code.

- **Languages:** To estimate how much human generated code is reduced to decrease the software project size which is developed with high level languages and tools, there are two popular parameters.
- One is Universal Function Points (UFP), which consider external user inputs, external outputs, internal logical data groups, external data interfaces, and external inquiries.
- These are used to indicate relative program size to implement required functionality.

- The second one is SLOC which is used to estimate the reduced size of the software project after a solution is formulated and an implemented language is known.
- The selection of the language for the software project development is depending on what kind of the data that the application (project) be processed because each language has a domain of usage.
- For example, Visual Basic is very expressive and powerful in building simple interactive applications, but it was not suitable for real-time, embedded programming.

- Generally the high level languages has numerous software engineering technology advances, language enforced configuration control, separation of interface and implementation, architecture control primitives, encapsulation, concurrency control, and many other to reduce the source lines of code.
- The reduction of code may increase the understandability, changeability, and reliability. But some of the drawbacks of reduction of size are high-level abstraction reduces the performance, increasing consumption of resources, and etc.

Object oriented methods and Visual modeling: The object oriented technology reducing overall size of what needs to be developed. The following are three reasons for succeed of object oriented projects (proposed by BOOCH).

1. An object oriented model of a problem and its solution encourages a common vocabulary between the end users of a system and its developers. Due to the shared understanding the problem being solved.

2. The use of continuous integration creates opportunities to recognize risk early and make incremental corrections without destabilizing the entire development effort.

3. Object oriented architecture provides a clear separation among different elements of a system, creating firewalls that prevent a change in one part of the system from entire architecture.

- Booch also propose the five characteristics of successful object oriented projects

1. Very high focus on the development of a system provides a well understood collection of essential minimal characteristics.

2.The current development process is centered on results, encourages communication, and not afraid of fail.

3. The effective use of object oriented modeling.

4. The reality of strong architectural vision.

5. The application of well managed iterative and incremental development life cycle.

Improving the software process: The process is a big term which represents various activities in the software development process. In a software oriented organizations there were several processes and sub processes are running. Generally the processes in organizations divided into three categories.

- Meta processes: Which represent organization policies, procedures and practices for pursuing the software business. Here we focus on organization economics, long term strategies and software return on investment (ROI).

- **Macro processes:** It describes policies, procedures, and practices for developing a complete software product within certain cost, schedule, and quality.
- **Micro processes:** These describe policies, procedures, and practices for achieving the goal of software by the development team. They focus on intermediate product baseline with adequate quality and functionality as economically rapidly as practical.
- The three levels of the processes are overlapped at some point in the project development process, even though they have different objectives, audiences, metrics, concerns and time sales.

- Here we consider the macro processes, which are project level processes to reduce the cost of project development. To make the project as a success one there should be integration of all processes which may implemented through sequential and parallel steps. Generally the project development has two types of process activities

1. Productive activities.
2. Overhead activities.

- The productive activities produce some tangible progress towards the end product. These activates may be prototyping, modeling, coding, debugging etc.

- The overhead activities have an intangible impact on the end product. These activities may be plan preparation, progress monitoring, risk assessment, financial assessment, configuration control, quality assessment, integration and testing etc.
- Really the overhead activities have value added efforts but in general less focus given to these activities. Finally the goal of processes is to maximize the resource allocation to productive activities and minimize the impact of overhead activities.
-

Improving the team effectiveness:

Managing the personnel has a great impact on the software project development process.

- The COCOMO model describes that combination of personnel skills and experience have an impact on productivity as a factor of four.
- At the same time it is difficult to measure the performance of the software team whether it is functioning correctly or wrongly. To form a good development team the managers follow the simple procedure "hiring of good people".

- The good software development team should have two characteristics, one is Balance and other one is coverage. Whenever a team is out of balance it is helpless.
- It have the good combination of planners, designers, coders, testers, trainers and soon. It means a good team need covers strong individual persons at key positions. To maximize the team management we should follow the bellow points;

1. A well managed project can succeed with nominal engineering team.
2. A mismanaged project will almost never succeed, even with an expert team of engineers.

3. A well architected system can be built by a nominal team of software builders.

4. A poorly architected system will have difficulty even with expert team of builders.

- In examining how to staff a software project, Boehm offered the five staffing principles.

1. The principle of top talent: Use better and fewer people.

2. The principle of job matching: Fit the tasks to the skills and motivation the people available.

3. The principle of career progression: An organization does best in the long run by helping its people to self actualize.

4. The principle of team balance: Select people who will balance and go with one another.

5. The principle of phase out: Keeping a misfit on the team does not benefit anyone.

- Software development team is an exercise. Mangers must take care of teamwork and results rather than individual accomplishment.
- Out of above five points, team balance and job matching be the primary issues. The top talented and phase out is the next important issues. We give final importance to career progression.

- The software managers also have good leadership qualities to improve the team effectiveness. The following are some qualities that a software manager should have.

1. Hiring skills. Few decisions are as important as hiring decisions. Placing the right person in the right job seems obvious, but at the same time difficult.

2. Customer-interface skill. Avoiding adversarial relationships among stakeholders is a prerequisite for success.

3. Decision making skill. We know that a good leader run into problems and decision making.

4. Team building skill. The manager should establish trust, motivate progress, transition average people into top performers, eliminate misfits, and consolidate diverse opinions into a team direction.

5. Selling skill. Successful managers must sell all stakeholders on decisions and priorities. Selling achievements against objectives.

Improving automation through software environments: The tools and environments used in software development process have a linear effect on the productivity.

- Planning tools, requirements management tools, visual modeling tools, compilers, editors, debuggers, quality assurance analysis tools, test tools, and user interfaces provide crucial automation and support for evolving the software engineering objectives.
- In addition to above configuration management environments provide the foundation for executing and instrumenting the process.

- Automate the manual tasks that are inefficient or error prone. A software project success shown that a highly integrated environment is necessary both to facilitate and to enforce management control of the process.
- An environment that provides semantic integration and process automation can improve productivity, improve software quality, and accelerate the adoption of modern techniques.

Achieving required quality: Today what are the best practices accepted by the software industry are derived from development processes and technologies. These practices would help to increase the software project cost efficiency at the same time those are also help to improve the software quality.

The following are the some key issues to improve the software quality.

1. During the software development focuses on the key requirements and identifies the critical cases at the beginning of the development. The continuous monitoring of requirements makes the balance between the requirements evolution, design evolution, and plan evolution.

2. Use the elegant metrics, indicators to measure the progress and quality of the software from its beginning of the life cycle to fully completed product.

3. Provide best environment where to integrate various processes to support configuration control, change management, rigorous design methods, document automation, and regression test automation.

4. Use the advanced tools such as visual modeling techniques, higher level languages that support architectural control, abstraction, reliable programming, reuse, and self documentation.

5. Focus on early and continuous insight into performance issues through demonstration base evaluations.

- Insights into the runtime performance issues are also important to assess the quality of the software project. The following are some of key issues in project development life cycle which affects the quality of the software.

Project beginning: The proposed design was declare to be low risk with adequate performance margin.

Initial design review: Positive assessments of the design margin done mostly on paper. The actual application algorithms, databases are also fairly understood, but the operating system overhead, database management overhead, inter-processes and network communication overhead are secondary and misunderstood.

Mid-life-cycle design review: The assessment started in cut fashion at the middle, as early benchmarks and initial tests are correct.

Integration and test: Serious performance problems are finding out then change the architecture. At this point the used infrastructure may be the scrap.

Peer inspections: A pragmatic view

- Peer inspections are sometimes leads to overestimation of quality of the system. Generally peer reviews are important to compare the quality of the system against to the primary quality mechanisms and indicators.
- The following are some of primary quality mechanisms to be followed during the development of the project.

1. Transitioning engineering information from one artifact to another. At this moment we are assessing the consistency, feasibility, understandability, and technology constraints in engineering.

2. Major milestone demonstrations to be assessed against tangible criteria in the context of relevant use cases.

3. Environment tools that ensure representation rigidness, consistency, completeness, and change control.

4. Life cycle testing for detailed insight into critical trade-offs, acceptance criteria, and requirements completeness.

5. Change management metrics for insight into multiple perspective changes or divergence from quality and process goals.

- The peer inspections are useful to have the products of junior team members reviewed by seniors or amateurs to experts for accelerating the getting knowledge and skill in new personnel.
- Generally all the developers represent their product as a natural byproduct of the process; therefore the peer inspections should cover all the developers rather than all the components of the system.
- We are also performing random component inspections periodically to monitor the system performance.

- We are also comparing the software developed by the one team with another one to gain the varying levels of information.
- Naturally an inspection is focused on resolving existing issues in an effective way to determine the cause or arrive at a resolution once the cause is understood.
- Software development organizations perform more and more inspections on several issues but actually 20% of technical or engineering issues deserve the peer inspections.

- Sometimes human inspections degenerate comments on style and first order semantic issues. Significant design or architectural errors really need the peer inspections by the superficial.
- . To resolve the problems in design and architectural issues we follow the bellow activities.

1. Analysis, prototyping, or experimentation.
2. Constructing design models.
3. Committing the current state of the design model to an executable implementation.

4. Demonstrate the implementation strengths and weaknesses in the context of use cases and scenarios.

5. Incorporating lessons learned back into the models, use cases, implementations and plans.

- Finally the quality assurance is everyone's responsibility and should be integral to all process activities instead of a separate discipline performed by quality assurance specialist.

Principles of conventional software engineering:

- Over the past two decades there has been a significant re-engineering of the software development process.
- conventional management and technical practices have been replaced by new approaches that combine themes of successful project experience with advances in software engineering technology. The following are thirty major principles of software engineering.

1. Make quality #1: Quality must be quantified and mechanisms put into place to motivate its achievement.
2. High quality software is possible: Techniques that have been demonstrated into increase quality include involving customer, prototype simplifying design, conducting inspections and hiring the best people.
3. Give products to customers early: No matter how hard to learn user's needs during the requirements phase the most effective way to determine real needs to give users a product and let them play with

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 try to solve a problem, be sure to explore all the alternatives and don't be blinded by obvious solution.
5. Evaluate design alternatives: After the requirements are approved, the developer must examine variety of architectures and algorithms. The architecture should relevance to the design specification.

6. Use an appropriate process model: Each project must follow 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: Generally software development follows one notation throughout the development life cycle. But it is not optimal in all cases.
8. Minimize the intellectual distance: To minimize the intellectual distance, the software structure should be close to the real world structure.

9. Put techniques before tools: An undisciplined software engineer with a tool becomes a dangerous.
10. Get it right before make it faster: It is easy to make a working program run faster than it is to make a fast program. That is don't worry about initial optimizations.
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: The best technology will not compensate for poor management and good manager can produce great result even with management resources.

13. People are the key to success: Highly skilled people with appropriate experience, talent, and training are key. The right people with insufficient tools, languages, and process will succeed. Wrong people with appropriate tools, languages, an d process will probably fail.

14. Follow with care: Just everybody is doing something does not make it right for you. It may be right, but you must carefully assess it applicability to the environment.
15. Take responsibility: In any engineering discipline, the best methods can produce unpleasant designs, the out of date methods produce elegant designs.
16. Understand the customer's priorities: One of the fact is the customer would tolerate 90% of functionality delivered late if they could have of it on time.

17. The more they (customer) see, the more they need: the more functionality if the developer provides to a user, the more functionality the user wants.

18. Plan to throw one way: One of the important success factors is whether or not the project is new. The brand new applications rarely work the first time.

19. Design for change: The architecture, components, and specification techniques must accommodate the change.

20. Design without documentation is not a design

21. Use tools but realistic: we know that software tools make the users more efficient.

22. Avoid tricks: Some time the developers use tricks to develop the programs, those are functionally correct but in unclear way.

23. Encapsulate: Information hiding is simple, and this concept results easy to test the software and much easy to maintain.

24. Use coupling and cohesion: Use coupling and cohesion are the best ways to measure software's natural maintainability and adaptability.

25. Use the McCabe complexity measure: There are so many methods to measure the complexity of software, but use McCabe method which is simple one.

26. Don't test your own software: The software developer should never be the primary tester of their own software.

27. Analyze causes for errors: First analyze the causes for errors when those are detected because it is cost effective to reduce the effect of an error by preventing it then to find and fix it.

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 exclusively by person-months makes little brains.

30. Expect excellence: The team members will do much better if team leader have expectations for them.

The principle of modern software management:

- The conventional software management principles are not highlight or suitable to the current trends followed in the industry. In this connection the modern software principles were evolved. The following are ten (10) modern software principles.

1. **Base the process on an architecture first approach:** To start the software development, the developer first thoroughly identifies the requirements then chooses the significant design decision. Before start the lifecycle (development) process check the selected resources are sufficient for complete software development.

2.Establish iterative lifecycle process that face risk early: Today the organizations not follow to define the entire problem at once, design the entire solution, build the software, then test and deploy in sequence.

- The organizations adopt follows iterative development approach which facilitates, problem understanding, effective solution, and an effective plan over several iterations. It also balances the stockholder's objectives. The main advantage of iterative approach is risks are identified early to avoid software scrap and rework.

3. **Transition design methods to highlight component based development:** The developers should transits from line-of-code mentality to component based mentality to reduce the amount of human generated code and custom development.

- A component based model is unified set of interface and behavior. preexisting line of code either in source or executable format, with a defined

4. **Establish change management environment:** The dynamics of iterative development, concurrent workflows require the controlled baselines.

5. **Enhance change freedom through tools that support round trip engineering (iterative):** The iterative development process supports automate and synchronize engineering information in different formats. Without substantial automation, change management, documentation, and testing it is difficult to reduce project development time. Change freedom is necessity in iterative process at the same time the role of environment is crucial.

6. **Capture design artifacts in rigorous, model based notion:** Generally model based approach supports rich graphical and textual design notions. Visual modeling and formal machine process able language provides good measures than traditional design approach.

7. **Instrument the process for objective quality control and progress assessment:** Project development progress and the quality should be assessed in the middle of the development lifecycle.

- The best assessment mechanisms are derived directly from the engineering artifacts and able to integrate all the activities in the development process.

8. **Use demonstration based approach to assess intermediate artifacts:** Transit from current development artifacts to executable one to early elimination of architectural defects.

9. **Plan intermediate releases in groups of usage situations with evolving levels of detail:** It is necessary that the software management process has continuous demonstrations within the operational context.

- The evolution of project increments and generations must be equal with current level of understanding of requirements and architecture.

10. **Establish configurable process that is economically scalable:** No single process is suitable for software development. A practical process frame work must be organizing a broad spectrum of applications. The software development process must integrate economy of scale, development automation, architecture patterns and components.

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.
- To implement the modern techniques it is necessary that an initial version of the system be rapidly constructed early in the development process with an importance of addressing high-risk areas, stabilizing the basic structure, refining requirements,

developing system on core architecture with number of reviews until achieving desired level of functionality, performance and robustness.

- The iterative development gives importance to whole system rather than the individual parts. Risks are identified early through continuous integration and refinement of requirements, architecture, and plans.
- We are also achieving some economical benefits with iterative approach, but to quantify the benefits is a difficult issue.

- The following are some points represented by the COCOMO-II in the development of software projects by using the iterative approach.

Application precedent: Domain experience is an important factor in understanding how to plan and execute a software project. For extraordinary systems, the key goal is to face risks and establish early model, even if they are incomplete or experimental. This is the primary reason that the software industry move to an iterative life cycle process.

Process flexibility: Today modern software has large solution space and many interrelated issues that there is a vital need for continuous changes. These changes natural in problem understanding, solution space, or in plans. So project development process must support an efficient change management.

Architecture risk resolution: Architecture first development is a central theme in a successful iterative development process. The project team first develops and then stabilizes the architecture before developing all components that comprise entire application.

This approach initiates integration activity early in the life cycle as the verification activity of the design process and products.

Team cohesion: Successful teams are cohesive and cohesive teams are successful. The successful and cohesive teams share common objectives and priorities. Successful and cohesive teams avoid project confusion and entropy that may generated from stakeholders expectations.

Software process maturity: The software engineering institute's Capability Maturity Model (CMM) is a well accepted benchmark for software process assessment. The CMM is crucial important for avoiding software development risks and develops organization's software assets. Here we follow object quality control.

UNIT-III
Life-Cycle Phases

- The most important characteristic of a successful software development is the well defined separation between research and development and production phases.
- Suppose any software project is failed the primary reason is improper separation of these two stages. The unsuccessful projects exhibit the following characteristics.

1. Overemphasis on research and development. Too many analyses or paper studies are performed or construction of engineering baseline is differed.

2. Overemphasis on production. Rush-to-judgment designs, premature work by overage coders, and continuous hacking is typical.

A modern software development process must define to have the following characteristics.

1. Evolution of plans, requirements, and architecture, together with well defined synchronization points.

2. Risk management and objective measures of progress and quality.

3. Evolution of system capabilities through demonstrations of increasing functionality

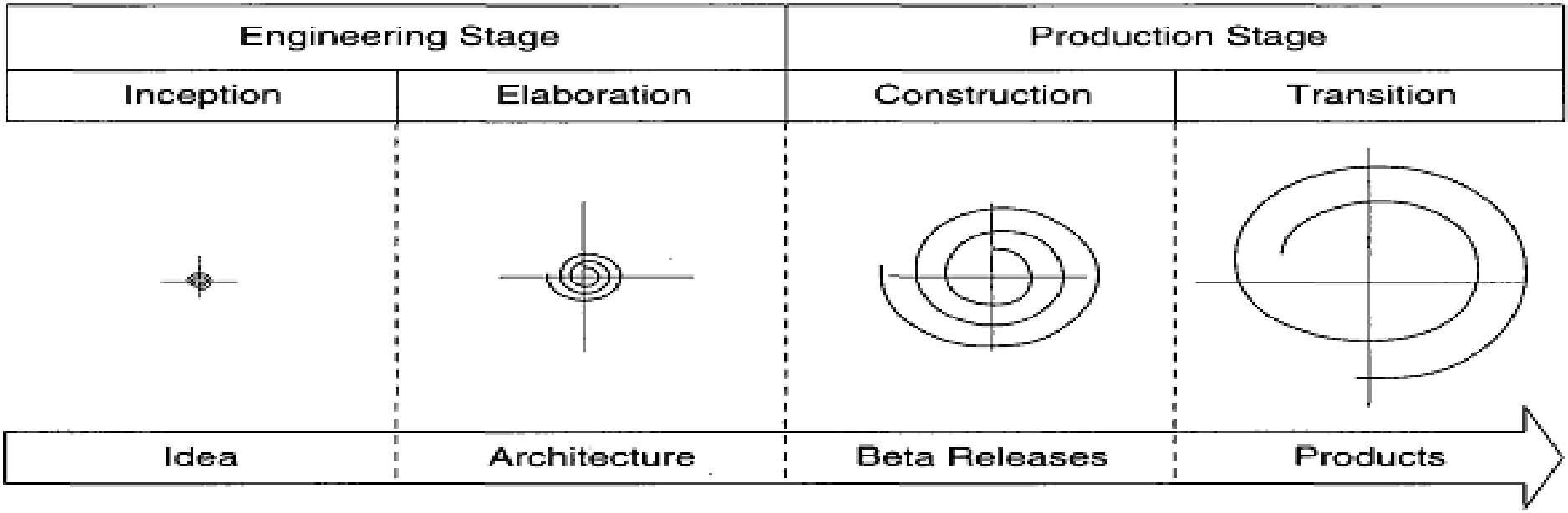
Engineering and Production phases:

- The economic factors provide a simple frame work for deriving a lifecycle picture. To achieve the scale of economics, and higher returns on investment, the developers should develop the software with process automation and in component based development model. At the starting of the project development the following are two stages of the lifecycle.

1. The engineering stage, driven by less expected but smaller teams doing design and synthesis activities.

2. The production stage, driven more predictable but larger teams doing construction, test, and development activities.

- The transition between the engineering and production phases is crucial for stakeholders. Generally the production plans are agreed after enough understanding of the problem; all stakeholders are agreed to the particular solution and committed to go for production.
- Consequently, the engineering stage is decomposed into two stages – ***Inception,*** and ***Elaboration*** and production stage into ***Construction*** and ***Transition***. These four phases are mapped to the spiral model. The following diagram shows the development of project stages.



- Here the size of the spiral corresponds to the sluggishness of the project with respect to the artifacts that has been developed.
- This sluggishness visualizes the maintaining the process or development consistency, regression testing, documentation, quality analysis, and configuration control

- Increase in sluggishness may have little effect on activity but it requires huge changes in architectural design, requirements specification, planning shifts, and other subsequent phases. In iterative model each phase include all activities, in various proportions.

Inception Phase:

- The overall goal of the inception phase is to achieve the concurrence between the stakeholders and lifecycle phases of the project. The following are some primary and essential activities of the inception phase.

Primary Objectives:

1. Establish the project software scope and boundary conditions, including operational concepts, acceptance criteria, and a clear understanding of what is and is not planned to be in the project.
2. Selecting critical use cases of the system and the primary states of operation that will reduce the major design trade-offs.
3. Demonstrating at least one candidate architecture against some of the primary situations.
4. Estimating potential risks.

Essential Activities:

1. Formulating the scope of the project. This activity involves capturing the requirements and operational concepts in an information repository that describes user's view of the requirements. The information repository should be sufficient to define the problem space and derive the acceptance criteria for the end product.

2. Synthesizing the architecture. Here design drawbacks, problem space ambiguities, and available solution space assets are evaluated. The information repository is created is sufficient to demonstrate the feasibility of at least one candidate architecture and an initial baseline of make/buy decisions. So the cost, schedule, and resource estimation can derive.

3. Planning and preparing the business cases. Here alternatives for risk management, staffing, iteration plans, and cost, schedule, profitability trade-offs are evaluated. The infrastructure is sufficient for the project development is determined.

Primary Evaluation Criteria:

1.Do all the stakeholders agree on the scope definition and cost and schedule estimates?

2 .The requirements are well understood, as evidenced by reliability of critical use cases?

3. The cost and schedule estimates, priorities, risks, and development processes believable?

4. The depth and breadth of an architecture prototype demonstrate the preceding criteria?

5. The actual resource expenditures versus planned expenditures acceptable?

Elaboration Phase:

- We know that elaboration phase is most difficult one than remaining three phases. At the end of the elaboration phase, the total project engineering is considered to be completed and project goes to production.
- Whether to start the production or not, is depends upon the transition from a simple operations with low risk to the operations with higher cost and risk.
-

- The elaboration phase must ensure that the architecture, requirements, and plans are stable enough, and risks are sufficiently mitigated, cost and schedule for the completion of the development can be predicted within acceptable range.

Primary objectives:

1. Base lining the architecture as rapidly as practical.
2. Base lining the vision.
3. Base lining high reliability plan for the construction phase.

4. Demonstrating that baseline architecture will support the vision at reasonable cost and reasonable time.

Essential Activities:

1. Elaborate the vision: This activity involves establishing high reliable and understanding of critical use cases that make architectural or planning decisions.

2.Elaborating the process and infrastructure: The construction process, the use of tools and automation support, their respective evaluation criteria is established.

3. Elaborating the architecture and selecting components: Potential components are evaluated and make/buy decisions are understood so that construction phase cost and schedule can be determined with confidence.

Primary Evaluation Criteria:

1. Is the vision stable?

2. Is the architecture stable?

3. Executable demonstration shows that major risk elements have been addressed?

3. All the stakeholders agree that the current vision met the current plans to develop the complete system by using the current architecture?

Construction Phase:

- The construction represents a production process, in which emphasis is placed on managing resources and controlling operations to optimize costs, schedules, and quality.
- At this moment the management thinks about transition from development of intellectual property during inception and elaboration activities to the development of deployable products during construction and transition activities.

- Many projects are produced by following the parallel increment construction methodology. These parallel activities significantly accelerate the availability of deployable releases, it also increase the complexity of resource management and synchronization of workflows and teams.

Primary Objectives:

1.Minimizing development costs by optimizing resources and avoiding unnecessary scrap and rework.

2.Achieving adequate quality as rapidly as practical.

3. Achieving useful versions as rapidly as practical.

Essential Activities:

1.Resource management, control, and process optimization.

2. Complete component development and testing against evaluation criteria.

3. Assessment of product releases against acceptance criteria of the vision.

Primary Evaluation Criteria:

1. Is the project baseline mature enough to be deployed in the user community?

2. Are the stakeholders ready for transition to the user community?

3. Are actual resource expenditures versus planned expenditures acceptable?

Transition Phase:

- The transition phase is entered when project to be deployed in the end user domain. The transition phase requires a usable system has been developed with acceptable quality levels and user documentation. This phase includes following activities.

1. Beta testing has performed to validate the project against user expectations.
2. Beta testing and parallel operations performed to replace the previous system.
3. Conversion of operational databases.
4. Training of users and maintainers.

- The transition phase was completed when the developed system achieved the complete vision. For some projects, the end point of this life cycle phase is starting point for next version of the product. For other projects deployment completely deliver information to the user for operation, maintenance, and enhancement.
- Generally the transition phase focuses on the activities to place the complete system in the hands of the user. It includes, beta releases, bug fix, and enhancement releases. Here the developer put his efforts for developing user oriented documentation, training users, supporting users for the product use.

Primary Objectives:

1. Achieving user self supportability.

2. Achieving stakeholder concurrence that developed system was complete and consistent with the vision.

3. Achieving final product as rapidly and cost effectively as practical.

Essential Activities:

1. Synchronization and integration of construction into development phases.

2. Deployment specific engineering.

3. Assessment of deployment against the complete vision and acceptance.

The Artifact Sets:

- To manage the complete software development process, we should collect various kinds of information and organized it into artifact sets.
- Each artifact set may have related artifacts that are persistent and are in a uniform representation format. Here set represents a complete view of the system and an artifact represents unified information that is developed as a single entity.
- Generally the artifacts are divided into five distinct sets in the project development lifecycle. Those are management set artifacts, requirement artifact set, design artifact set, implementation artifact set, and deployment artifact set.

The Management Artifacts:

- The management artifact set consisting of the artifacts related with process planning and execution. This set use ad hoc notations such as text; graphics required to represents the agreements among project personnel, among stakeholders and project personnel and stakeholders.
- Management set artifacts are evaluated, assessed, and measured through the combination of the following.

1. Relevant stakeholder review.

2. Analysis of changes between current artifacts with previous artifacts.

3. Demonstrations to balance all artifacts and in accuracy of business case.

The Engineering Set Artifacts:

- The engineering set artifacts consists of requirement set, design set, implementation set, and deployment set. The primary goal to evaluating quality of each artifact set is transiting the information from one set to another set. So we can maintain balance among the requirements, design, implementation, and deployment.

Requirement Artifacts: To specify the vision statement we use structured text in requirement artifact, this vision document describe the scope of the project and agreement between the funding authority and project team

- Here we use some ad hoc formats for supplementary specifications. Here we use UML notion for engineering representation of requirements model.
- The requirement set is primary to evaluate the remaining three sets of artifacts. Requirement artifacts are evaluated, assessed and measured through combination of the following

1. Analyze the consistency with the release specifications of management set.

2. Analyze the consistency between the vision and the requirements models.

3. Mapping the design, implementation and deployment sets to evaluate the consistency and completeness.

4. Analyze the changes between current versions of requirement artifacts with previous version.

5. Individual review of other dimensions of quality.

Design Artifacts: The solution or output of the project is designed by using the specific design model which is in UML notation. The design set consists of varying levels of ideas that represent the components solution space.

- The design model contains both structural and behavioral information to find out bill of materials. The design model information directly converted into the implementation and deployment artifacts.
- The design set artifacts includes, the design model, test model, software architecture description. The design set artifacts are evaluated, assessed and measured through a combination of the following.

1. Analyze the internal consistency and quality of the design model.

2. Analyze the consistency with the requirements models.

3. Translate the design models into implementation and deployment sets and then evaluate consistency, completeness between the sets.

4. Analyze the changes between current version design model and previous version design model.

5. Individual review of other dimensions of quality.

Implementation Artifacts: The implementation artifact set includes source code that is useful to develop the working system components and any executables required to test these components.

- The executables are primary parts of the end system which may custom components, application programming interfaces (API), reusable or legacy components of programming languages. Implementation set artifacts can be translated into deployment set.

- Some specific artifacts in this set has self documenting product source code and associated files, self documenting test source code and associated files, stand-alone component executables etc.
- Generally implementation artifact set are in human readable format that are evaluated, assessed, and measured through the following.

1. Analyze the consistency with design models.
2. Translate into deployment set notation.

3. Assessment of component source codes or executable files through inspection, analysis, demonstration, or testing.

4. Execute stand-alone components and compare the results with actual results.

5. Analyze the difference between current versions of artifacts with previous version.

Individual review of other dimensions of quality.

Deployment Artifacts: The deployment set includes user deliverables and machine language notations, executable software and build scripts, installation scripts, executable target data to use the project in user environment.

- Deployment information to be installed, executed, and dynamically reconfigured to support the end system. Deployment artifacts are evaluated, assessed and measured through a combination of the following.

1. Test the deployment set against the system usage and the quality to be evaluated with respect to consistency and completeness.

2. Test the system partitioning, replication, and allocation strategies in relevance with implemented components and physical resources.

3. Test the usage scenarios – user manual, mainstream usage, anomaly management.

4. Analyze the changes between the current version and previous version.

5. Individual review of other dimensions of quality.

The goal of artifacts is to optimize the process activities and objectives. There is no scientific methodology for selecting management, requirement, design, implementation and

- Generally each artifact set use different plans to gain the relevant information of the system. The management artifacts concentrate on plans, process, business objectives, and acceptance criteria.
- Requirement artifacts concentrate on engineering information and operational concepts.
- Design artifacts concentrates on engineering blue prints. Implementation artifacts concentrate on building block of the solution in human readable form. Deployment artifacts concentrate on solution in machine readable form.

- The modern software development process map to any one of the artifact set.

1. Management: Work flow, defect tracking, change management, documentation, spreadsheet, resource management, and presentation tools.

2. Requirements: Requirements of management tools.

3. Implementation: Compiler/Debugger tools, code analysis tools, rest coverage analysis tools, and test management tools.

4. Deployment: Test coverage and test automation tools, network management tools, commercial components and installation tools.

Implementation set versus Deployment set:

- The implementation set and deployment set have their own concepts, the differences between these two sets are essential.
- The structure of information delivered to the user is entirely different from the structure of source code executed in user environment.

- We know that the engineering decisions has great impact on quality and deployment of the system, at the same time it is difficult to understand the engineering decisions. The engineering decisions have impact on the following areas of the project.

1.Dynamically reconfigurable parameters such as, color patterns, number of servers, number of simultaneous clients, data files and run time parameters.

2.Effects on compiler / link optimizations such as space optimization and speed optimization.

3. Performance under certain allocation strategies such as centralized versus distributed computing, dynamic load balancing, hot back up etc.

4. Virtual machine constraints such as file descriptors, garbage collection, heap size, maximum record size, etc.

5. Process level concurrency issues such as deadlock and race conditions.

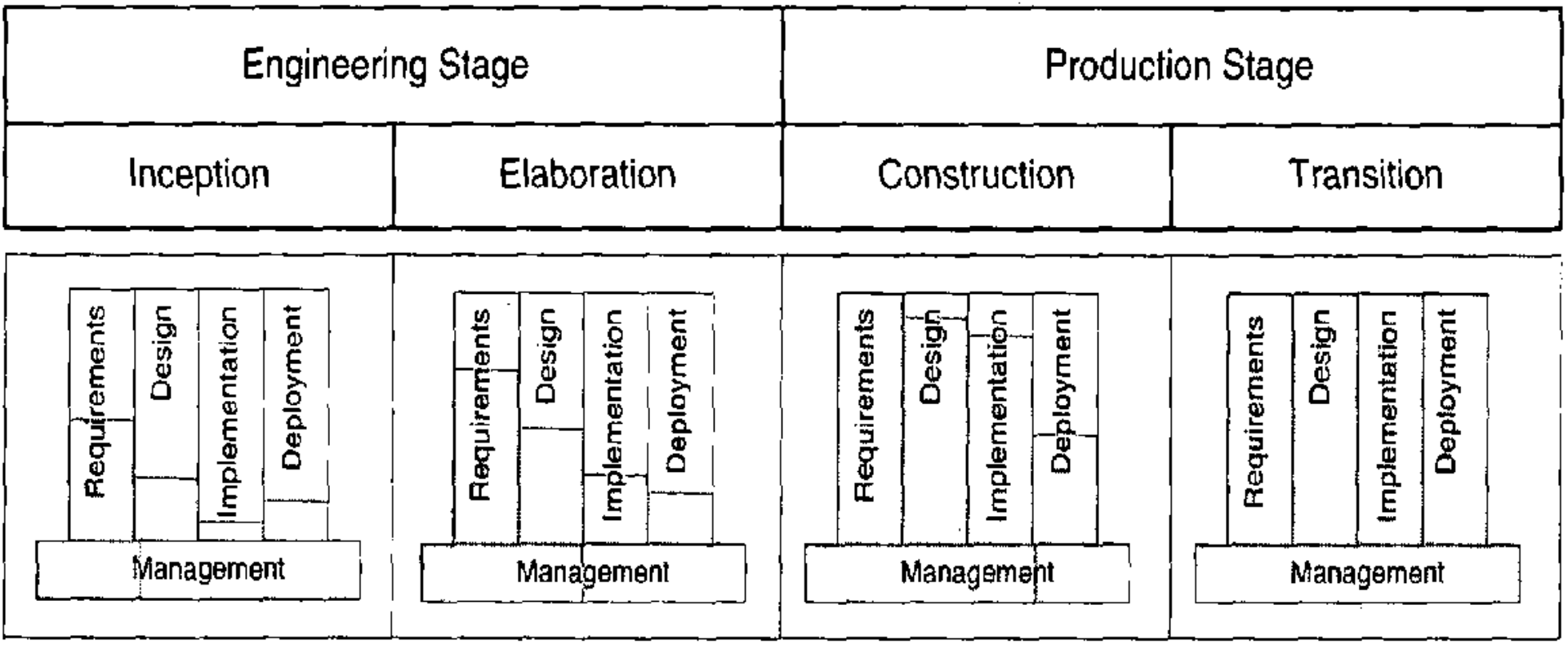
6. Platform specific differences in performance or behavior.

- The above configuration information either exists in implementation set or in deployment set. In case of dynamically reconfigurable or portable system (components) it is essential to bifurcate source code implementation (deployment) from target environment (implementation).

Artifact evolution over the lifecycle:

- Each state of project development represents a certain amount of precision in the final system description. At the beginning of the lifecycle, precision is low and the representation high.

- Finally the precision is high and everything is specified in full detail. At any point in life cycle, the five sets of artifacts will be different states of completeness.
- But, at the project completing stage all the artifacts are compatible and reasonably traceable to one another.
- Each phase of project development focuses on particular artifact set. At the end of the each phase the system development will process all the sets. The following diagram represents the life cycle evolution of artifact set.



- The inception phase focuses on critical requirements, initial deployment view, and little focus on implementation and primary focus on design architecture.

- During elaboration phase more concentration on requirements, much more in design and further work on implementation and deployment issues such as performance trade-offs and make / buy analysis.
- The elaboration phase activities may include generation of executable prototype. Here we generally understanding of component interfaces.
- The main focus on construction phase is design and implementation. At the starting of construction phase should analyze design artifacts in depth.

- Here we understand design of source code and individually tested components. The construction phase drives the requirements, design and implementation sets almost to complete.
- The main focus on transition phase is to achieving consistency and completeness of the deployment set in the context of other sets. Remaining defects are resolved and feedback from alpha, beta, and system testing is incorporated.
-

Test Artifacts: In conventional software testing, document driven techniques are used.

- Generally the development teams built requirement documents, top level design documents, and detailed design documents in software development process.
- In the same way before conducting the software testing, the system test plan documents, system test procedure documents, integration test plan documents, unit test plan documents are constructed by the testing team

- In the modern approach the test notations and artifacts used for software development and testing are similar. The test procedure is considered as an important activity in end product.
- The testing is done basically by identifying the test infrastructure. Due to this, the following engineering constraints are used in testing process.

1.The test artifacts must be developed concurrently with the product from inception through deployment. Thus testing is a full life cycle activity.

2. The test artifacts are communicated, engineered and developed within the same artifact sets as the development product.

3. The test artifacts are implemented in programmable and repeatable formats.

4. The test artifacts are documented in the same way that the product is documented.

5. Developers of test artifacts use the same tools, techniques, and training as the software engineers developing the product.

•

- The testing is only one aspect of the assessment of workflow. Other aspects include inspection, analysis, and demonstration.
- Testing refers to the explicit evaluation through execution of deployment set components under a controlled scenario with an expected and objective outcome.
- The success of a test can be determined by comparing the expected outcome to the actual outcome with well defined mathematical precision.

Management Artifacts:

- The management artifact set is used to capture intermediate results and additional information required to document the product/process.
- This information have how to maintain product, how to improve the product and improvements of the process.

Business case: The business case information is used to determine whether investing in the project in beneficial or not.

- The business case artifact includes the expected cost, revenue, backup data and technical management plans.

- In the business case is a detail demonstration of risks and reality associated with management plans.
- For large projects business case is a full scale proposal with multiple volumes of information. For small projects it is a brief plan along with spreadsheet.
- The main purpose of business case is to transform the vision into economic terms, so an organization can make accurate ROI

- **Software Development Plan:** The software Development Plan (SDP) describes the process framework in full detail.
- It is a defined document of project's process. It must fulfill organization standards, project design and requirements, any contracts, used across all subordinate organizations doing software development.
- Two important indicators of SDP is (a) periodic updating and (b) understanding and acceptance by managers.
-

- **Work Breakdown Structure:** The work breakdown structure (WBS) is key feature for budgeting and collecting costs.
- The project manager should have an idea about project costs and how they are expended to monitor and control project's financial performance.
- Suppose the WBS may plan improperly it may cause for project failure, because it has large impact on design and production stages of the project.

- The project manager concentrates on lower level of WBS after particular level of stability achieved in the project development.
- **Software Change Order Database:** Managing change is one of the fundamental issues in iterative development process. With good change freedom, the project may develop more productively.
- The change freedom can increase the flexibility in project contents, quality, and number of iterations that a project can achieve within a given schedule. Generally change freedom is achieved through process automation.

- **Release Specifications:** The project's plan scope and objective is assessed before release.
- This assessment criterion is depending on the project's vision statement and many other parameters.
- These artifacts are evolved in accordance to the process development, thereby achieving consistency and requirements understandability is improved throughout the lifecycle.
- There are two important forms of requirements, (i) vision statement (ii) evaluation criteria.

- **Vision statement:** The vision statement describes the contract between the buyer and the development team.
- This information evolving and changing slowly across the lifecycle phases. The vision statement may use ad hoc or some other formats for easily understand to the buyer.
- The use case model in vision statement serves to understand the operational concepts of the project by the user/buyer.

- **Evaluation criteria:** The evaluation criteria contain the release specifications, are brief snapshots of objectives for an intermediate lifecycle phases.
- The evaluation criteria defined management artifacts instead of requirement artifacts. The evaluation criteria may also derived from the vision statement and many other sources.
- The vision statement represents the system requirements that are related to user/buyer. But actually lower level requirements are evolved by making the several iterations in project development phases.
-

- **Inception iterations:** The inception iterations can capture key issues associated with the critical use cases that have an impact on architecture alternatives and the overall business case.
- **Elaboration iterations:** These iterations are used to demonstrate against the candidate architecture, verify the critical use cases and critical requirements of vision statement can be met with low risk.
- **Construction iterations:** These iterations are associated with meaningful set of use cases, when passes, develop useful subset of product that can be transitioned to formal set or alpha or beta releases.

- **Transition iterations:** It uses complete set of use cases and associated evaluation criteria compose acceptance criteria associated with deploying a version into operation.
- **Release descriptions:** Release description documents describe the result of each release, including performance against evolution criteria in corresponding release specification.
- The release description document should contain release baselines that represent evolution criteria and provides confirmation that each criterion has been addressed in an acceptable manner.

- The release description document should also have metrics summary that quantify the quality and relative terms.
- This document can also have outstanding issues, recommendations for process and product improvement, trade-offs in addressing evaluation criteria, follow-up actions and similar information.
- **Status assessment:** Status assessment provides periodic snapshots of project health and status, including the software project manager's risk assessment, quality indicators and management indicators.

- **Environment:** An important issue in modern software development process is to define development and maintenance environment.
- A healthy and integrated development environment must support the automation of the development process.
- Generally this kind of environment must have requirements management, visual modeling, document automation, and host, target programming tools, automated regression testing and continuous and integrated change management, and feature defect tracking.

- A common feature in all successful projects is to hire good people and provide them with good tools to accomplish their jobs.
- Automation of the software development process returns in quality, ability to estimate costs, and schedules, overall productivity using a smaller team.
- **Deployment:** A deployment document can take many forms. Depending on the project, it could include several sub documents for transitioning the product into operational status.

- Here one of the big issues is, suppose the project maintenance is handover to separate maintenance organization then deployment artifacts include computer system operations manuals, software installation manuals, plans and procedures for cutover, site surveys, etc should transfer to the maintenance organization.
- For commercial software products, deployment artifacts may include marketing plans, sales plans and training courses.

Engineering Artifacts:

- Most engineering artifacts are captured as per accurate engineering planning such as UML, programming languages, or executable machine codes. There are three engineering artifacts are more important, those are;

Vision document.

Architecture description.

Software user manual.

- **Vision document:** The vision document provides a complete vision for the software system under development. It also supports the contract between funding authority and development organization.

- Whether the project is huge confidential one or it may be a small commercial product, they should capture the expectations of the stakeholders.
- A project vision to be challengeable as for the requirements, architecture, plans, and technology. But a good vision document should change slowly
- Generally the vision document should be developed in user perspective, containing essential features of the system, and acceptable levels of quality.
-

- The vision document should contain two appendixes. The first appendix should describe the operational concepts using use cases.
- the second appendix should describe change risks inherent in vision statement. The vision statement also has what to be included and what features are considered but not included.
- The vision document should specify operational capabilities of the system, user profiles, interoperational interfaces with entities outside of system boundary

- The vision should not define for initial operating level; it paves the path for system adaptability.

Architectural description: The architecture description provides an organized view of the software architecture.

- The architectural design is extracted from the design model and includes views of design, implementation and deployment information is sufficient to understand operational concepts.
- The breadth of the architecture description may vary from project to project depending on many factors. Generally the project architecture is described as a subset of design model or abstraction of design model or combination of the both.

- **Software user manual:** The software user manual provides the user with the reference documentation necessary to support the delivered software.
- The user manual contents are variable across the application domains. The user manual should have installation procedures, usage procedures and guidance, operational constraints, and user interface description etc.
- The software user manuals should developed at the beginning of the lifecycle, because it is a necessary mechanism for communicating and stabilizing a part of requirements.

Pragmatic Artifacts:

- We known that the documents are more important in the software development process, but the conventional document driven approaches wasted valuable engineering time on developing, polishing, formatting, reviewing, updating, and distributing documents. The documents should be maintained in any software development process due to two reasons.
- There were no exact engineering methods or languages for requirements specification or design in project development. Here paper documents with ad ho text and graphical representations were the default formats.
-

- Predictable language implementations and deployment were extremely puzzling and highly unstructured.
- So to present the details of software structure and behavior to other interested reviewers good human readable formats were needed.
- Due to the good documentation it is easy to monitor the progress of the software development at various stages and judged properly.
- When we dealing with documents, one of the critical issue is quality of the documents than the quality of engineering information they represented.

- To monitor the quality of the documents, document production cycles, review cycles, and update cycles also inserted into the software development process.
- Due to this the software development schedules and synchronization points are prolonged.
- The software developers would follow an effective approach to redirect the efforts from documentation to improving the understandability of hardness of information source and allow on-line review of present information by using the smart browsing and navigation tools.

- This approach eliminates the huge, unproductive source of information, scrap and rework. This approach addresses the following issues.

People want to review information but don't understand the language of the artifacts – Suppose the artifacts were written in the engineering language, the most of the reviewers try to understand the engineering language first.

- We know that the reviewers having knowledge about engineering concept, but the engineering language learning capability is less.

- Suppose the artifacts are in high engineering language, then to understand these artifacts the reviewers take much more time and effort.
- **People want to review the information but don't have capability to access the tools –** Generally every software development organization not fully tooled (automated).
- Only some stakeholders have the capability to review the engineering artifacts through on-line.
-

- Suppose some organizations forced to exchange the paper documents with the help of some standardized languages, visualization tools and the web then other stakeholders does not accept this approach then the efficiency in software development process was corrupted.

- **Human-readable engineering artifacts should use exact notations that are complete, consistent and used in a self-documenting manner** - Properly spelled English words should be used for all identifiers and descriptions.

- Acronyms and abbreviations should be used only where they are well accepted in the context of component's usage. The usage of encryptions, programming language tools may simplify the artifact's author's work but causes errors in the software development process.
- We know that software is written only once, but it read many times. Therefore, readability should be given importance and usage of proper English words should be required in all engineering artifacts.

- **Useful documentation is self-defining** - The self-documenting engineering artifacts should be developed to avoid separate documents to describe all the details of a model, component, or test procedure.
- Suppose the document is generated but not used, and then eliminate it by considering the requirements which are used in its completion.
- **Paper is tangible; electronic artifacts are too easy to change** – Naturally paper documents are static, persistent and tangible but those are preferable by only some stakeholders. The online and web-based artifacts have many advantages but suffer with easy to change and volatile in nature.

UNIT – IV
Model Based Software Architectures

- **Introduction:** Software architecture is a key concept for every software system. The software architecture has several dimensions of complexity.
- Generally the software system performance issues and features of software system has not described by any laws of software engineering. Those are directed by well accepted form of mathematics. So the software architecture has no certain principles.
- There were many assumptions and unclear guidelines for software architecture, but the good software architecture is situation dependent.

- Due to the lagging in theory, software architects depend on some form of experimentation in framing software architectures. This is the main reason for moving from traditional software development approach to an iterative process.
- There was no well definition for the software architecture, because early software systems were less powerful than present day systems.
- . Early systems would require only informal representations in single computer, single program system, the mapping among design objects, implementation objects and deployment objects was unimportant.

- But today software systems advanced to multiple distinct models and views to exploit the advantages of modern technologies such as commercial components, object oriented methods, open systems, distributed systems, host and target environments and modern languages.
- *Here a model is a relatively independent abstraction of a system. A view is a subset of a model that abstracts a specific, relevant perspective*.
-

Architecture: A Management Perspective

- The most critical technical product of a software project is its architecture which includes the infrastructure, control, and data interfaces that permit software components to cooperate as a system and software designers to cooperate efficiently as a team.
- We know that establishing accurate and precise communications among teams of people is an unchanging problem in any software organization.
-

- Suppose the communication media has multiple languages, inter group latency varies then the communication problem is become complex and unsolvable.
- If a software development team is be successful, the interproject communications in software architecture must be accurate and precise. From management perspective there was three different aspects of architecture.

1. Architecture (Intangible design concept) is the design of a software system; it is not a design of a component.

- It includes all engineering required to specify a complete bill of materials.
- Significant make or buy decisions are resolved and all necessary components are elaborated so that individual component cost and construction or assembly costs can be determined with confidence

2. Architecture baseline (Tangible artifacts) is a slice of information across the engineering artifact set to satisfy all stakeholders that vision can be achieved within the parameters of the business case.

3. Architecture description is an organized subset of information extracted from the design set models.

- It includes the additional ad hoc notion necessary to clarify the information in the models. The architecture description communicates how the intangible concept is realized in the tangible artifacts.

- The definitions are necessarily theoretical, because architecture has different forms in different domains.

- Generally the architecture of the software for air traffic control system is very different from the software architecture of small development tool. The relation between software architecture and modern software development process is represented bellow.

1. A valuable project in which important make or buy decisions are solved early stages leads to the path of achieving stable software architecture. Here all software development phases are involved and these lifecycle events cause transition from one stage to another stage.

2. Architecture representations provide a basis for balancing the trade-offs between the problem space and the solution space.

3. The architecture and process encapsulate many of the important communications among individuals, teams, organizations, and stakeholders.

4. Poor architectures and immature processes are often given as reasons for project failures.

5. A mature process, an understanding of the primary requirements, and a demonstrable architecture are important requirements for predictable planning.

6. Architecture development and process definition are the intellectual steps that map the problem to a solution without violating the constraints; they require human innovation and cannot be automated.

Architecture: A Technical Perspective

- Software architecture includes the structure of software systems (collection of elements and composition of elements into progressively larger sub system), their behavior, (collaboration among elements) and prototypes that guide these elements, their collaborations and their compositions.

- The context of software architecture, behavior, and patterns must include functionality, performance, flexibility, clarity, economic trade-offs, technology constraints and artistic concerns.
- Generally an architecture framework is defined in terms of views that are abstraction of UML models in the design set.
- . An architecture view is a concept of design modeling. It has architecturally important information.
- Naturally software systems require four views: Design, Process, Component, and deployment. The following is the purpose of these views.

- **Design view:** The design view describes architecturally significant elements of the design model which includes structure and function.
- This view is an idea of design model. It modeled statically using class and object diagrams and dynamically using any of the UML behavioral diagrams.
- **The process view:** The process view describes the runtime collaboration issues involved in executing the architecture on a distributed deployment model, including the logical software network topology, interprocess communication, and state management.

- This view is modeled statically using deployment diagrams and dynamically using UML behavioral diagrams.
- **Component view:** It describes the architecturally important elements of implementation set.
- This view is an idea of the design model, which includes software source code in perspective of project's developers. It modeled statically using component diagrams and dynamically using nay of UML behavioral diagrams.

- **Deployment view:** It addresses the executable realization of the system, including the allocation of logical processes in the distribution view to physical resources of the deployment network.
- It is modeled statically using deployment diagrams, dynamically using UML behavioral diagrams.
- Generally architecture descriptions take different forms and styles in different organizations and domains. At any moment architecture requires a subset of artifacts in each engineering set.

- The actual level of content in this set is situation dependent. Architecture baseline includes the following.

1. Requirements: Critical use cases, system level quality objectives, and priority relationships among features and qualities.

2. Design: Names, attributes, structures, behaviors, groupings and relationships of significant classes and components.

3. Implementation: Source component inventory and bill of materials of all primitive components.

4. Deployment: Executable components sufficient to demonstrate the critical use cases and the risk associated with achieving the system qualities.

Workflows of the Process:

Introduction: Generally software processes are described in the format of sequence of activities. These sequential descriptions of processes are simple to understand, represent, plan and conduct.

- At the same time these simple sequential activities are not realistic on software projects because software projects required team efforts.

- The team work may require many artifacts must be synchronized, cross-checked, homogenized, merged, and integrated. So the distributed nature of the software process and its assistant workflows is the primary management complexity.
- One of the drawbacks of conventional software process development was representing the macro process as a sequential thread of activities in the lifecycle model from requirement phase to deployment phase.
- But it is hard to specify the boundaries between the phases of lifecycle model and additional efforts are required to complete a particular phase and engineering decisions are slowed and stopped.

- A modern software development lifecycle activity are turned into predominant activities, such as inception, elaboration, construction, and transition. Here we specify the process as a states rather than phases.

Software process workflows:

- The software process is described in two ways, either by macro process or micro process.
- The macro process contains discrete phases and iterations but not discrete activities. Generally we know that continuous activities are carried out in each phase and iteration.

- The micro processes are also known as workflows which produces workflows. The workflow is used to a thread of cohesive and sequential activities. The following are various kinds of workflows.

Management workflows: Controlling the process and ensuring win conditions for all stakeholders.

Environment workflows: Automating the process and developing the maintenance environment.

Requirement workflow: Analyzing the problem space and developing the requirements artifacts.

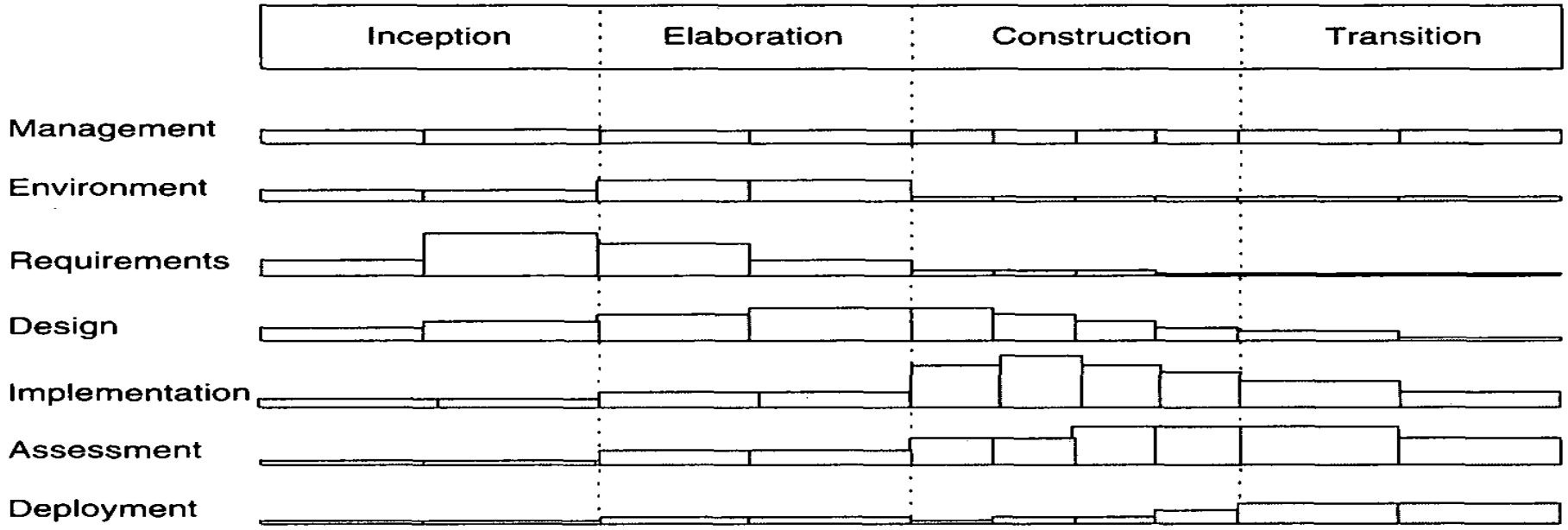
Design workflow: Modeling the solution and developing the architecture and design artifacts.

Implementation workflow: Programming the components and developing the implementation and deployment artifacts.

Assessment workflow: Assessing the trends in process and product quality.

Deployment workflow: Transitioning the end products to the user.

The following diagram shows the levels of effort required by the workflows in various project development phases. It also represents modern process framework and provides a viewpoint from modern software management.



- **Architecture first approach:** Deep requirement analysis, design, implementation and assessment activities are performed before the construction phase.
- The implementation and architecture testing occurs prior to the full scale development and components testing. Here we also focus on completeness and product quality.

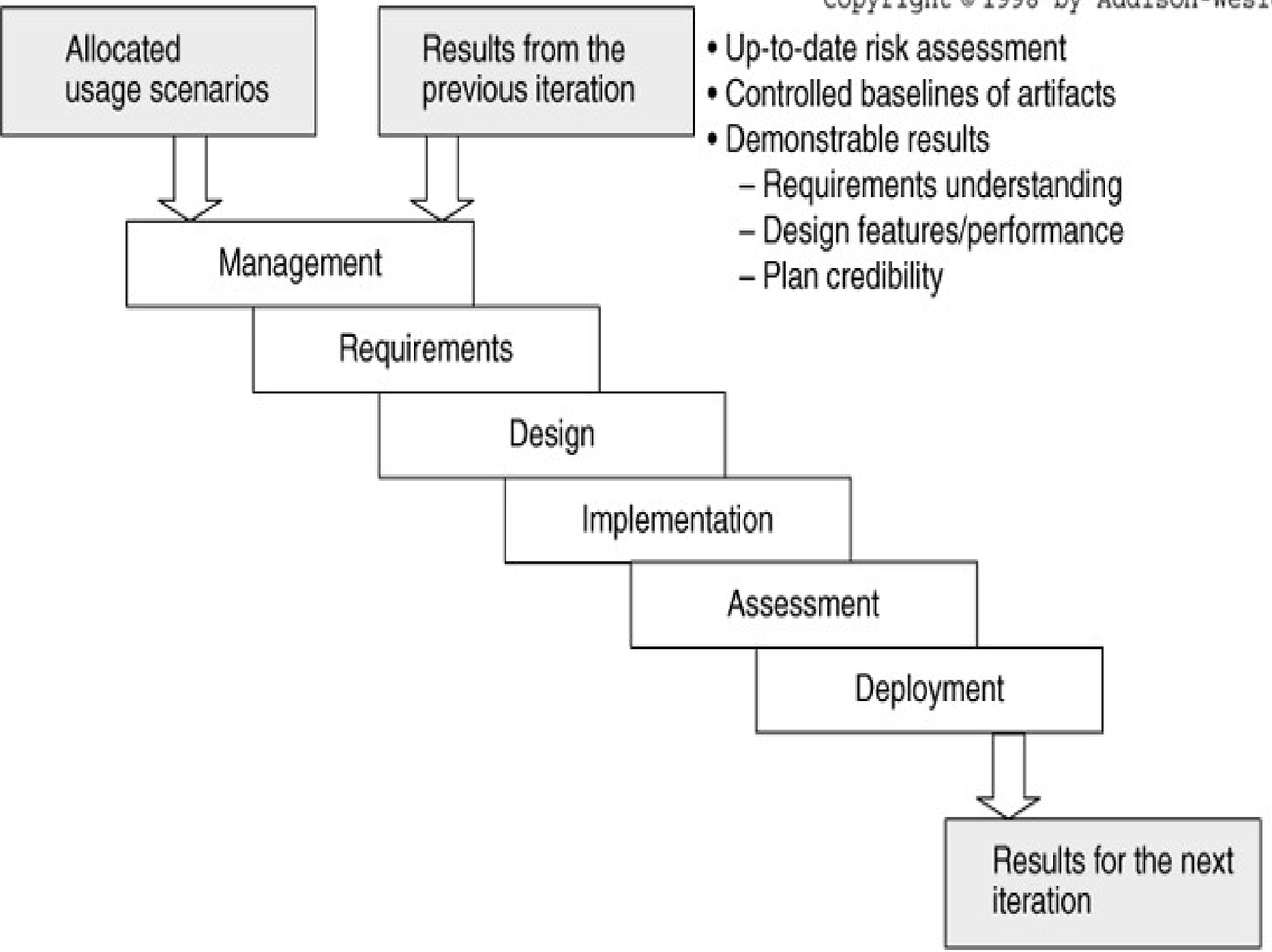
- **Iterative life-cycle process:** As shown in the above diagram, each life-cycle phase represents at least two iterations of each of seven workflows.
- It is not necessary to have two iterations of each workflow. Some projects need only one iteration and some others need much iteration in each phase.
- It is advantageous to have more than one iteration in each phase for better and adequate results.

- **Round trip engineering:** Improving environmental activities is risky. The environment is the tangible area of the project's process, methods and notations for producing artifacts.
- **Demonstration based approach:** The implementation and assessment activities gets started in the initial stages of lifecycle. Hence now focus on the constructing the executable subsets of the architecture.
- ----------
-
-

- The documentation results are the by-product of the workflow activities and quality assurance is not separately carried out because implicitly it occurs in almost all activities.
- The following table shows, allocation of artifacts and workflow in each lifecycle phases of inception, elaboration, construction and transition.
- workflows in life-cycle phases

Iteration workflows:

- Iteration consists of sequential set of activities in various sizes, depending on position of iteration in development cycle.
- Each iteration is defined in terms of set of allocated usage scenarios. The components needed to implement all selected scenarios are developed and integrated with the result of previous iterations. The following diagram shows individual iterations workflow.



Management: Project Management is concerned with the content of the iterations, assigning work, and the contents of anticipated releases.

Environment: Environment is concerned primarily with maintaining the software change database – tracking, prioritizing, addressing any changes that arise. All tools, standards, procedures are part of this.

Requirement: Requirements is concerned with looking carefully at the baseline plan, architecture, and requirement set artifacts needed to fully expand the use cases that need to be demonstrated at the end of this iteration – as well as their evaluation criteria.

•

Design: It is concerned with developing the design model and test model by evolving the baseline architecture and design set artifacts against the evaluation criteria for a specific iteration; Also need to update the design set artifacts in response to the activities within this iteration.

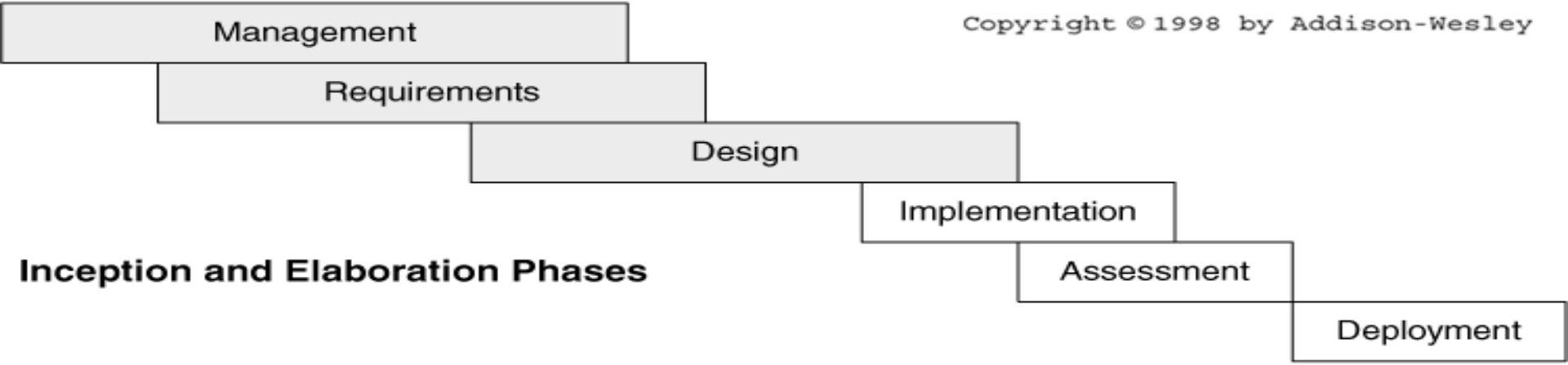
Implementation: It addresses developing (from scratch), customizing, or acquiring (purchase, reuse) new components and to test and integrate these components into to current architectural baseline.

Assessment: concerned with evaluating the results of each iteration to insure compliance with evaluation criteria and to identify rework needed before deployment of this release or allocated to the next iteration; also, assess the results for this iteration so as to improve the next iteration's procedure.

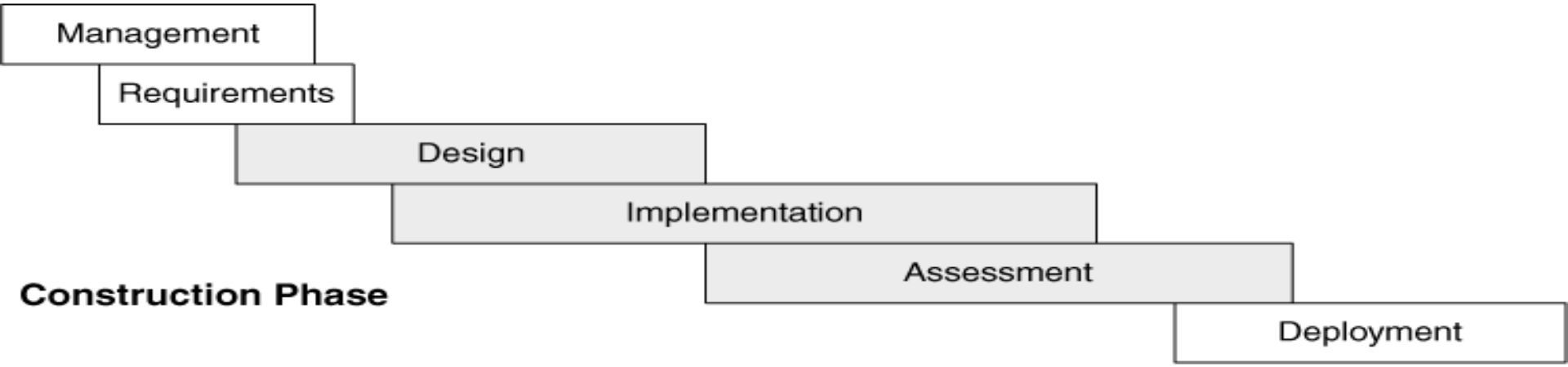
Deployment: concerned with transmitting the release either internally or to an external organization for exercise.

•

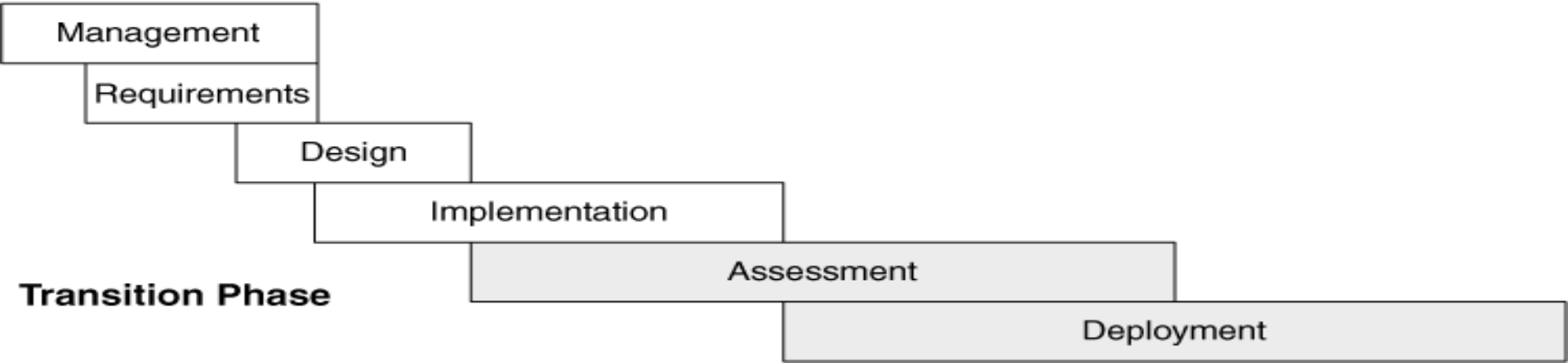
- In software development process activities are generally goes on concurrently for example requirements analysis is not done continuously as a single phase. It interlinks with management, design, implementation, and soon.
- Iterations in inception and elaboration phases focus on management, requirements, and design activities. Iterations in construction phase focus on design, implementation and assessment.
- Iterations in the transition phase focus on assessment and deployment. The following diagram shows importance activities across the life cycle.



Inception and Elaboration Phases



Construction Phase



Transition Phase

UNIT – V
Checkpoints of the Process

- **Introduction:** Checkpoints are also referred as milestones. A milestone is defined as a planned or organized event used to measure the progress of the project.
- Good milestones in software lifecycle was needed for gather the stakeholders face to face, to discuss progress and plans of the project.
- The purpose of the milestones is not only checking how well a project is performed but also achieving the following actions.

1. Coordinate stakeholder's expectations and achieve concurrence on requirements, design and plan.

2. Coordinate related artifacts into a consistent and balanced state.

3. Identify the major risks, issues, and out-of-tolerance conditions.

4. Perform global assessment for the whole lifecycle, not just for individual situation or intermediate product.

- Milestones must have well defined expectations and provide tangible results.

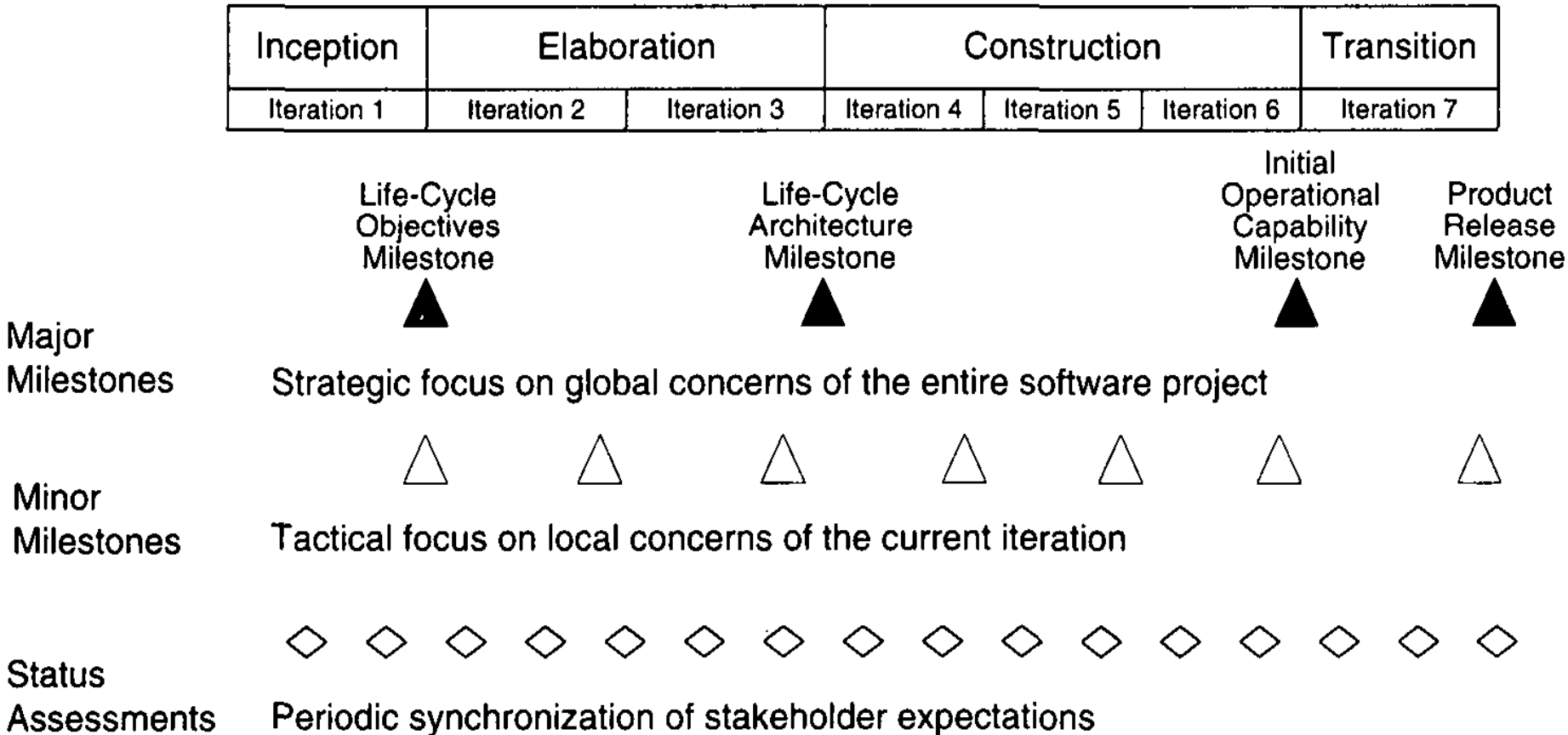
- It is not possible to change the milestone objectives once understanding the project trade-offs among requirements, design and plan.
- In order to achieve stakeholder's expectations throughout the software lifecycle, the project checkpoints or milestones are categorized into the three sequences.

1. Major milestones
2. Minor milestones
3. Status assessments

- **Major milestones:** These system wide events are held at the end of each development phase. They provide visibility of system wide issues, Synchronize the management and engineering perspectives and verify that the aims of the phase have been achieved.
- **Minor milestones:** These iteration focused events are conducted to review the content of iteration in detail and to authorize continued work. The formats and content of minor milestones are highly dependent on the project and the organizational culture.
-

- **Status assessments:** These periodic events provide management with frequent and regular insight into the progress being made. Periodic status assessments are crucial for focusing continuous attention on the evolving health of the project and its dynamic priorities.
- The level and the number of milestones will vary depending on several parameters, such as scale, number of stakeholders, business context, technical risk, and sensitivity of cost and schedule.
- Most of the projects have four milestones, on exceptional cases one more major

- For small projects few or no minor milestones be needed to manage intermediate results and the number of status assessments are also infrequent. The following diagram shows sequence of project milestones for large projects.



Major Milestones:

- Major milestones are the system wide events that are held at the end of each development phase to provide visibility to system wide issues synchronize the management and engineering perspectives and verify that the goals of the phase have been achieved.
- Major milestones are used in different process models, including conventional waterfall model.
- In iterative model, the major milestones are used to achieve concurrence among all stakeholders on the current state of the project. Naturally different stakeholders have different concerns.

Customers: Schedule and budget estimates, feasibility, risk assessment, requirements understanding, progress, product line compatibility.

Users: Consistency with requirements and usage scenarios, potential for accommodating growth, quality attributes.

Architectures and system engineers: Product line compatibility, requirements changes, trade-off analyses, completeness and consistency, balance among risk, quality, and usability.

•

Developers: Sufficiency of requirements detail and usage scenario descriptions, frameworks for component selection or development, resolution of development risk, product line compatibility, sufficiency of development environment.

Maintainers: Sufficiency of product and documentation artifacts, understandability, interoperability with existing systems, sufficiency of maintenance environment.

Others: Possibly many other perspectives by stakeholders such as regulatory agencies, independent verification and validation contractors, venture capital investors, subcontractors, associate contractors, and marketing teams.

•**Life-Cycle architecture milestone:** The lifecycle architecture milestone occurs at the end of the elaboration phase. The main goal of this milestone is to demonstrate an executable (working) architecture to all the stakeholders.

• In this milestone a more detailed plan for the construction phase is presented to all the stakeholders. It also considers the complex issues relating to the requirements and operational concepts.

• This milestone achieve consensus on baseline architecture, baseline vision, baseline software development plan, and evolution criteria for the initial operational capability.

- We know that the baseline architecture consists of both human readable representations and configuration controlled set of software components captured in engineering artifacts.
- A successful lifecycle architecture milestone will result in approval from stakeholders to proceed with the construction phase
- This is a most important milestone that transits the project from elaboration phase to the construction phase. A software project which is ready for this transition must have the following characteristics.

1. The complex issues in the project have been defined, by getting the approval from the stakeholders and codified into a set of scenarios to test the architecture.
2. A stable architecture has been baselined in source language format. Stability means the architecture should have performance, robustness, scalability, adaptability.
3. The risks in the project development were understood. At the same time all risks do not fully resolved, but the stakeholders should have an understanding of risks and serious consequences. The risks mitigation plans should be elaborated fully.

4. The development plan for construction and transition phases is defined with enough reliability that construction iterations can proceed with predictable results.

- The content of this milestone will vary across the project domains. It should include at least the following.

1.A presentation and overview of current project state.

2.A configuration controlled set of engineering information, available electronically or in hardcopy.

3.An executable demonstration of capability.

- **Initial operational capability milestone:** This milestone is occurs at the end of the construction phase.
- The goal of this milestone is to assess the readiness of the software to begin transition into customer/user site and to get permission to start the acceptance testing.
- This milestone would consider installation instructions to be followed, software version descriptions, user and operator manuals, and the ability of the development organization support to the user.

- This milestone can also reviewed quality parameters of the software for the customer usage. Here we perform acceptance testing incremental manner across multiple iterations.
- The initiation of transition phase is not meaning that the end of the construction phase. The construction and transition phases are overlapped until an initial product is delivered to the user for self sufficient operation.
- **Product Release milestone:** The product release milestone occurs at the end of the transition phase.

- The goal of this milestone is to check the completion of the software and its transition to the beneficial organization.
- Here the results of the acceptance testing are reviewed and installation instructions, software version descriptions, user, operator manuals, software support manuals, installation of the development environment at support sites are considered.
- This milestone is also considering the software quality parameters to determine whether quality is sufficient for the usage.
-

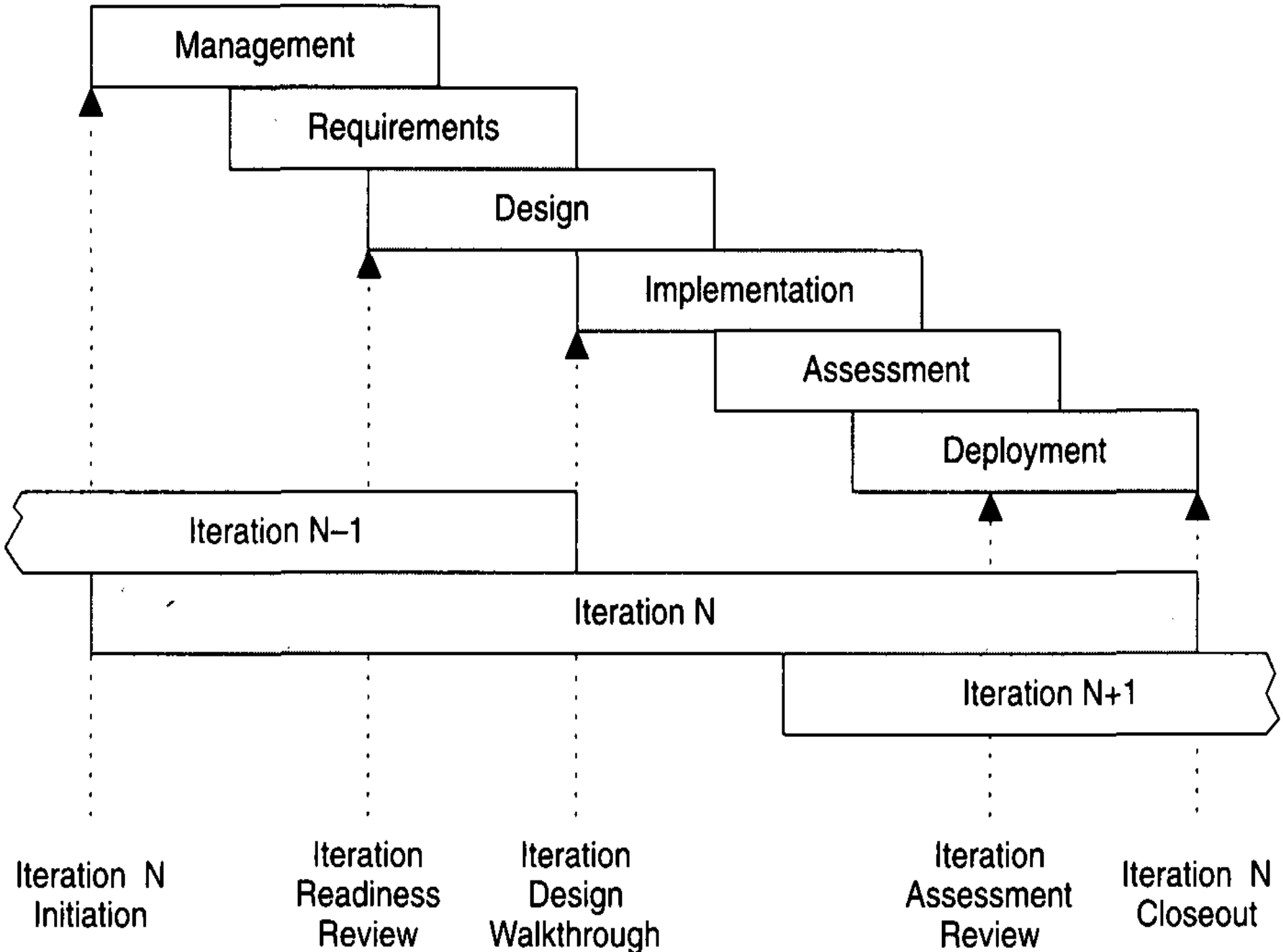
Minor Milestones:

- Minor milestones are called inch-pebbles. These are the iteration specific events are used to review iterative content in a detailed manner and authorized the continued work.
- Generally, the minor milestones are informal milestones which are depending on the content and length of the iteration.
- Normally iteration can take one-month to six-months of duration. In this duration only two minor milestones are required. One is *iteration readiness* review and another one is *iteration assessment review.*

- Sometimes for longer iterations one more intermediate minor milestone may be required. This intermediate minor milestone reviews the progress assessment and dissemination of work throughout the project.
- In minor milestones, all iterations should not have the same length and iteration can take different forms and priorities, depending on the position of the project in the lifecycle.
- Naturally early iterations focus on analysis and design, with the elements of discovery, experimentation and risk assessment.

- Later iterations focus on completeness, consistency, usability and change management.
- **Iteration readiness review:** This informal milestone is conducted at the start of each iteration to review the detailed iteration plan and the evaluation criteria that have been allocated to this iteration.
- **Iteration assessment review:** This informal milestone is conducted at the end of each iteration to assess the degree to which the iteration achieved its objectives and satisfied its evaluation criteria.

- The review of iteration results and the review of qualification test results used to determine the amount of rework to be done and to analyze impact of iteration results on the plan for subsequent iterations.
- The format and content of the minor milestones highly dependent on the project and organizational culture. The following diagram shows various minor milestones to be considered when a project is being planned


Management
Requirements
Design
Implementation
Assessment
Deployment
Iteration N–1
Iteration N
Iteration N+1
Iteration N Initiation
Iteration Readiness Review
Iteration Design Walkthrough
Iteration Assessment Review
Iteration N Closeout

Periodic Status Assessments:

- Risk management requires continuous attention to all activities of a software development process.
- Periodic status assessment is considered as a management review carried out on regular intervals of time. So the periodic assessment leads to indicate the project progress, quality, project dynamics, and maintaining continuous communication among all the stakeholders.

- Periodic status assessment is also considered as one of the critical project checkpoint, it provides special consideration on the gradual development of the software project properties.
- The purpose of status assessment in software development lifecycle provides the following.

1. A mechanism for openly addressing, communicating, and resolving management issues, technical issues and project risks.
2. Deriving objective data directly from present activities and evolving product configurations.

3. A mechanism for disseminating process, progress, quality trends, practices, and experience information to and from all stakeholders in an open form.

- There are some repeated themes from which status assessment is derived from successful and unsuccessful projects. Those are as follows.

1.From unsuccessful projects status assessments are:

a)High overhead activities due to the differences between daily work and status generated work.

b)Frequently cancelled, because high priority issues that require resolution.

2. From successful projects status assessments are:

a) Low overhead activities, because the material exists as everyday management data.

b) Rarely cancelled, because they considered too important.

- The default content of periodic status assessments includes the following topics.

1. Personnel: The contents of this topic include staffing plan vs actual attritions, additions and subtractions.

2. Financial trends: Expenditure plan vs actual for previous, current and next major milestones and revenue forecasts are the contents of the financial trends.

3. Top 10 risks: Critical resolution plans and issues and exposure of cost, time and quality comes under top 10 risks.

4. Technical progress: The content of technical progress includes, software management metrics and indicators, major milestone configuration baseline schedules, changed current trends and test quality assessments.

5. Major milestone plans and results: Schedules, risks and plans for the next major milestone, pass/fail results for all acceptance criteria are the contents of this topic.

6. Total product scope: Total size, growth and acceptance criteria included in the product scope.

•

The cost and schedule estimating process:

- We know that the project plans are developed or derived from two perspectives.

1. Forward-looking or Top-down approach.
2. Backward-looking or Bottom-up approach.

- The forward approach is starts with understanding of the general requirements and constraints, develops a macro level budget and schedule, and then decomposes these elements into lower level budgets and intermediate milestones.

- From the top-down perspective the following points should be considered for project planning.

1. First the software project manager can understands the project characteristics and then develops the requirements for overall size, process, environment, people, and quality of a project.

2. A macro level estimate of the total effort and schedule is developed using a software cost estimation model.

3. The software project manager partitions the estimate for the effort into a top level WBS which has management, environment, requirements, design, implementation, assessment and deployment.

- The project manager can also partitions the schedule into major milestone dates and partition the effort into staffing using the effort and schedule guidelines. This is the overall project plan. Here we ignore detailed project specific parameters.

4. Now a subproject managers given the responsibility for decomposing each of the WBS elements into lower levels using their top level allocation, staffing profile and major milestone dates.

- The backward-looking or bottom-up approach starts with the end in mid; analyze the micro-level budgets and schedules, then sum all these elements into the higher level budgets and intermediate milestones. From bottom-up perspective the following points should consider for the project planning.

1. The lowest level WBS elements are elaborated into detailed tasks, in which budgets and schedules are estimated by WBS element manager. These elements tend to incorporate the project specific parameters in an exaggerated way.

2. Estimates are combined and integrated into higher level budgets and milestones. This biases of individual estimators need to be homogenized so that there is a consistent basis of negotiation.

3. Comparisons are made with the top-down budgets and schedule milestones. Gross differences are assessed and adjustments are made in order to converge Between bottom-up and top-down estimates.

- If we compare the above two approaches, then top down approach enlarge the preferences of project management resulting an optimistic plan.
- Whereas bottom up approach enlarge the preferences of performance resulting in a pessimistic plan.

- For the development of plan through multi versions iterating is needed for validating results through one method and results refinement through another method.
- Throughout the project lifecycle uniformity of these approaches should be maintained.

The iteration planning process:

- Another method to derive the software project plane is to concentrate on the actual sequence of intermediate results produced during the each phase of project development.
- Here we plan the contents and schedule of major milestones and their intermediate iterations. This kind of iterative planning process is most tangible form of risk management.
- Here the term iteration is treated as complete synchronization of planning activities across the project along with good coordination of assessment activities of entire project baseline.

- Sometimes we may perform some micro iteration such as monthly, weekly, or daily are performed to achieve synchronization. The following appoints represents number of iterations in each and every phase of software project development.
- **Inception phase iterations:** Inception phase iterations mainly concentrate on the basic activities of the software project. These iterative activities demonstrate the candidate architecture and provide executable framework describing major critical areas within the project.

- The executable framework contains, existing components, commercial components, and custom prototypes, to demonstrate candidate architecture and for requirements understanding to establish good business case, vision and software development plan.
- Normally in the inception phase, large projects may require two planning iterations but most of the projects should have one iteration only.
- **Elaboration iterations:** The elaboration iterations concentrate on project architecture, complete framework and infrastructure for project execution.

- During the elaboration iterations the following major points should be considered.

1. Architecture initialization.

2. A design for worst case data processing in system.

3. A design for worst case control processing the system.

- Most of the projects have two iterations in this phase to achieve complete architecture baseline. Extraordinary projects may require additional iterations. But well established architecture project have one iteration only.

- **Construction iterations:** Most of the projects need at least two construction iterations. One is alpha release and another one is beta release.
- An alpha release has executable capability of all major points within the system. The alpha release represents 70% of total end product and quality levels below those expected in the final product.
- A beta release has 95% of the total product and achieves complete quality parameters. But to release the final product some more parameters are necessary such as robustness, performance to accept the product.

- Generally any project has two iterations here, but sometime we may add one or two iterations extra to manage risks or optimize resources.
- **Transition iterations:** Most of the projects have a single iteration to convert the beta release to the final product. But here number of micro iterations are required to resolve the final defects and include the performance improvements.

Conclusion: per the planning guidelines, most of the projects have four to nine planning iterations. But typical projects have six iteration profiles.

1. One iteration in inception: an architecture prototype.
2. Two iterations in elaborations: architecture prototype and architecture baseline.
3. Two iterations in construction: alpha and beta releases.
4. One iteration in transition: product release.

Pragmatic planning:

- Pragmatic planning plays a vital role in project development, which is more dynamic in an iterative environment.
- Project plan means ***the way in which the project requirements are transferred into a product within business constraints.***
- Good architectures and requirements understanding are the elements of adequate planning or good planning, where as bad architectures and misunderstood requirements are the elements of inadequate planning or bad planning.

- A successful project plan is not important one the project development has completed because it is rarely used by the performer on a day-to-day basis which does not create interest to end user.
- . Whereas unsuccessful project plan needs to be refined more as they are not produced instantly. So this unsuccessful plan is frequently used by performers.
- Planning document is worthless once the project development has completed. But action of planning is very important to achieve the project success.

- The planning document provides a framework and forcing functions for making decisions and transforms subjective, generic process framework into objective processes.
- A good project manager to compromise between current iteration plan and next iteration plan based on the results of current iteration and previous iterations.
- . For instance, the execution of iteration N in any phase, the software project manager required reviewing and controls the plan which was started iteration in N-1 and the planning is done for the iteration N+1.

- Planes are not certified by the managers, but also by the team members of the project. Project success is achieved when more open and visible plan exists.

UNIT-VI

PROJECT ORGANIZATIONS AND RESPONSIBILITIES

Line of Business Organizations:

- The following are the features of the software business organizations.

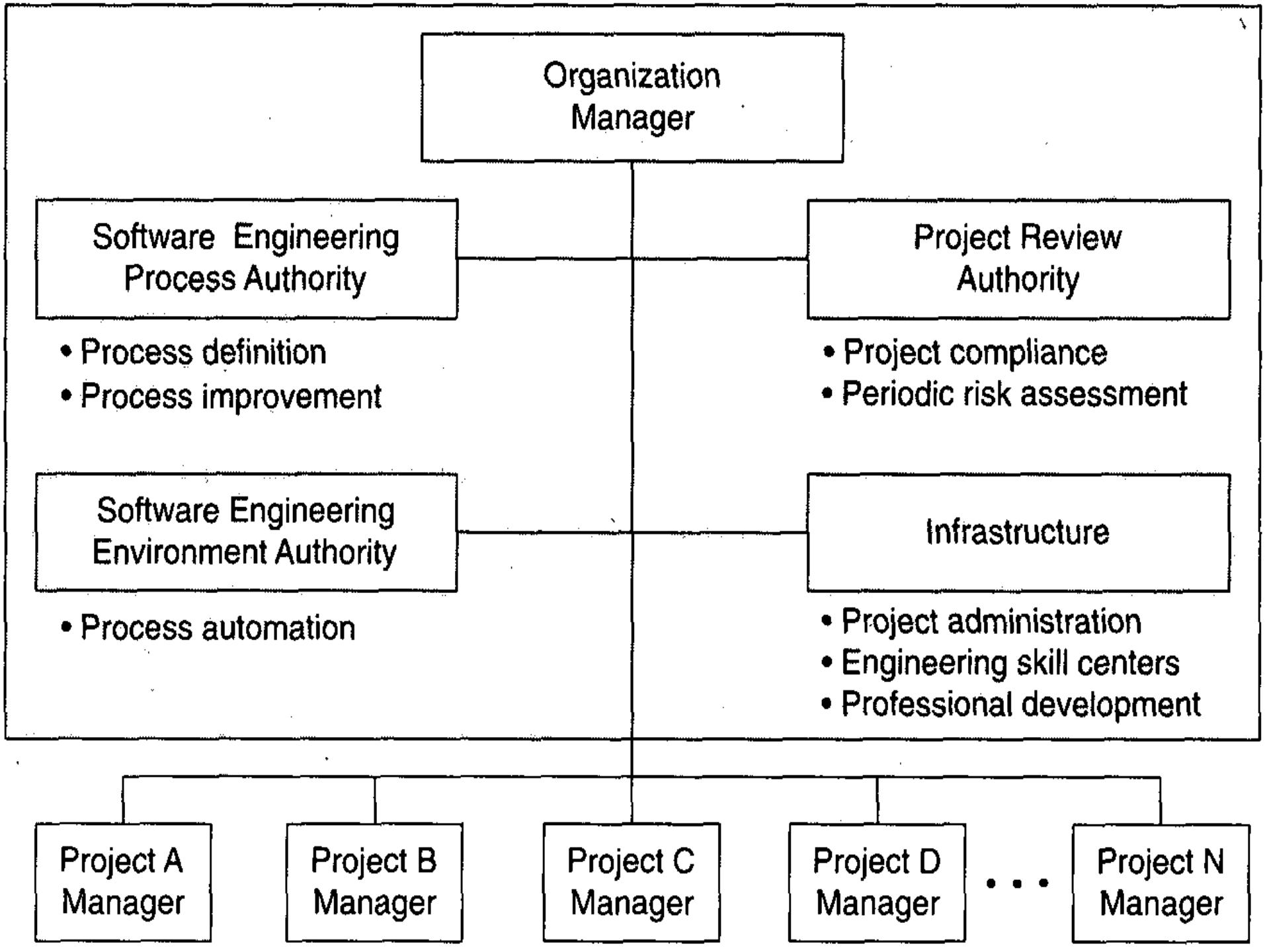
1. Responsibility of process definition and maintenance, which is specific to the line of business, where process commonality make the sense. For example a process for developing one kind of software is different from process to developing the other software.

2. Responsibility for process automation is an organizational role and it has equal importance with process definition role.

Normally projects achieve the commonality through the environment support and use of common tools.

3. Organizational roles may fulfilled by a single individual or several different teams, depending on the scale of organization.

- The following diagram shows the default roles of software business organization.


Organization Manager
Software Engineering Process Authority
• Process definition
• Process improvement
Project Review Authority
• Project compliance
• Periodic risk assessment
Software Engineering Environment Authority
• Process automation
Infrastructure
• Project administration
• Engineering skill centers
• Professional development
Project A Manager
Project B Manager
Project C Manager
Project D Manager
. . .
Project N Manager

- **Software Engineering Process Authority:** The software engineering process authority (SEPA) facilitates the exchange of information between all the project participants.
- SEPA role is necessary for any organization. It takes the responsibility and accountability for the process definition and its maintenance.
- . It also responsible for organization maintaining and current assessment of the project's development carried out within organization and plan for future process improvements.

- SEPA is a single individual person, such as general manager or sometimes it may be a team. Project process improvements and their distribution are carried out after the SEPA can thoroughly understands the project context.
- **Project Review Authority:** The project review authority is also a single individual team, responsible for software project fulfill with all organizational and business policies, practices, and standards.
- . Here the software project manager act as a review authority and responsible for requirements of a contract or some other project compliance standard.

- The PRA reviews both the project's conformance to contractual requirement and project's organizational policy requirements.
- Generally the project customer monitors contract requirements, contract milestones, contract deliverables, monthly management reviews, progress, quality, cost, schedule and risk.
- The RPA reviews customer commitments as well as stick to organizational policies, deliverables, financial performance, and risk events.

- **Software Engineering Environment Authority:** It is responsible for organization's process automation, keeping organization's standard environment, training projects etc. tools, techniques, and training can be implemented effectively only if SEEA is supporting and monitoring the standard environment.
- **Infrastructure:** An organization's infrastructure provides human resources support, project independent research and development and other capital software engineering assets

- The following are the some important software organization infrastructure components.

Project administration: Time accounting system, contracts, pricing, terms and conditions, corporate information systems integration.

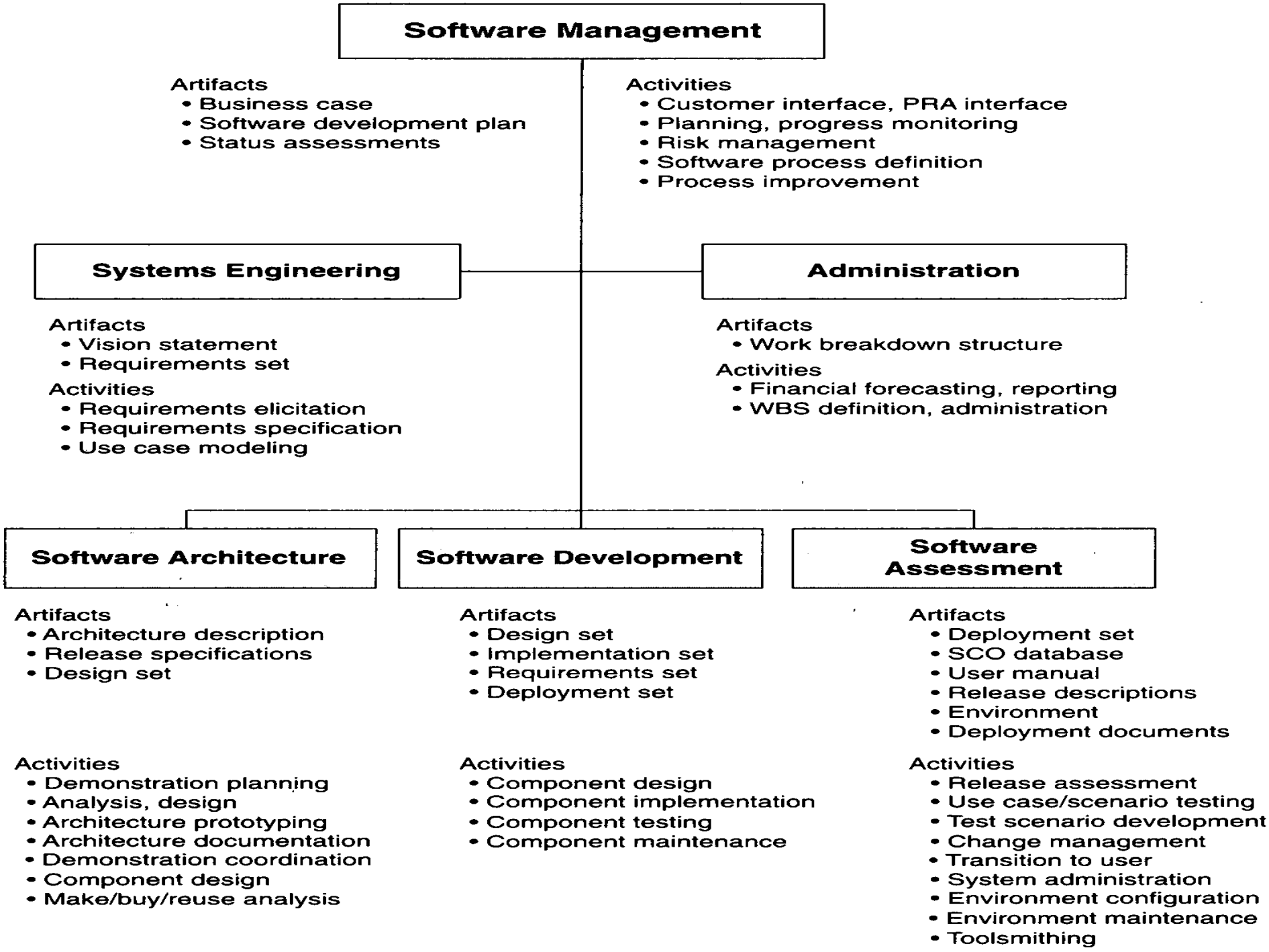
Engineering skill centers: Custom tools repository and maintenance, bid and proposal support, independent research and development.

Professional development: Internal training boot camp, personnel recruiting, personnel skills database maintenance, literature and assets library, technical publications.

- An organizational service center provides a standard environment funded by the line of business and maintained as capital asset for projects within the organization.
- The SEEA is a companion group of SEPA. The SEPA is responsible for process definition and improvements, and SEEA is responsible for process automation.

Project Organizations:

- The following diagram shows default software project organization and project level roles and responsibilities. The structure of any project organization is changed accordance with specific circumstance.


Software Management
Artifacts
• Business case
• Software development plan
• Status assessments
Activities
• Customer interface, PRA interface
• Planning, progress monitoring
• Risk management
• Software process definition
• Process improvement
Systems Engineering
Artifacts
• Vision statement
• Requirements set
Activities
• Requirements elicitation
• Requirements specification
• Use case modeling
Administration
Artifacts
• Work breakdown structure
Activities
• Financial forecasting, reporting
• WBS definition, administration
Software Architecture
Artifacts
• Architecture description
• Release specifications
• Design set
Activities
• Demonstration planning
• Analysis, design
• Architecture prototyping
• Architecture documentation
• Demonstration coordination
• Component design
• Make/buy/reuse analysis
Software Development
Artifacts
• Design set
• Implementation set
• Requirements set
• Deployment set
Activities
• Component design
• Component implementation
• Component testing
• Component maintenance
Software Assessment
Artifacts
• Deployment set
• SCO database
• User manual
• Release descriptions
• Environment
• Deployment documents
Activities
• Release assessment
• Use case/scenario testing
• Test scenario development
• Change management
• Transition to user
• System administration
• Environment configuration
• Environment maintenance
• Toolsmithing

- The following are the main features of any software project organization.

1. The project management team should be active; it is responsible for producing as well as managing each and every activity in the project development.

2. The architecture team is responsible for real artifacts (information) and for the integration of components.

3. the development team is responsible for component construction and maintenance activities. The assessment team is separate from the development.

- Here we consider testing and development activities go in concurrence with development.

4. Quality is the job of everyone, integrated into all activities and checkpoints. Each team takes responsibility for a different quality perspective.

Software Management Team: We know that the project development is constrained on schedules, costs, functionality, and quality parameters which are interrelated and require continuous negotiation with different kinds of stakeholders who have different goals.

- The software management team would always try to benefit all stakeholders. So the software manager can balance the several development activities. The following diagram shows the activities of software management team.

Artifacts

- Business case
- Vision
- Software development plan
- Work breakdown structure
- Status assessments
- Requirements set

- Systems engineering
- Financial administration
- Quality assurance

Responsibilities

- Resource commitments
- Personnel assignments
- Plans, priorities
- Stakeholder satisfaction
- Scope definition
- Risk management
- Project control

Life-Cycle Focus

Inception	Elaboration	Construction	Transition
Elaboration phase planning Team formulation Contract baselining Architecture costs	Construction phase planning Full staff recruitment Risk resolution Product acceptance criteria Construction costs	Transition phase planning Construction plan optimization Risk management	Customer satisfaction Contract closure Sales support Next-generation planning

- The software management team is responsible for planning, conducting, and adapting the plan to changes in understanding of requirements or design.
- The software management team should also take the responsibility of resource management and project scope and set operational priorities across the project development.
- The software management team takes the responsibility of quality. It could be achieved through manage the balance between various aspects.

- Finally the project is adequate for all stakeholders and optimal for many of them.
- **Software Architecture Team:** The software architecture team is responsible for project's architecture.
- This responsibility consists of accounting knowledge to specify billing of materials and engineering knowledge for make or buy decisions so that the components are constructed and assembly.
- The following diagram shows the activities of software architecture team.

Software Architecture

Artifacts

- Architecture description
- Requirements set
- Design set
- Release specifications

- Demonstrations
- Use case modelers
- Design modelers
- Performance analysts

Responsibilities

- Requirements trade-offs
- Design trade-offs
- Component selection
- Initial integration
- Technical risk resolution

Life-Cycle Focus

Inception	Elaboration	Construction	Transition
Architecture prototyping *Make/buy trade-offs* Primary scenario definition Architecture evaluation criteria definition	Architecture baselining *Primary scenario demonstration* Make/buy trade-off baselining	Architecture maintenance *Multiple-component issue resolution* Performance tuning Quality improvements	Architecture maintenanc *Multiple-component issu resolution* Performance tuning Quality improvements

- The skill of the software architecture team is crucial during any project development, because it provides framework for facilitating team communications for achieving system wide qualities and for implementing applications.
- In most projects, inception and elaboration phases are lead by software management team and software architecture team
- The software development team and software assessment teams are engaged in production phases.

- At the time of construction phase, the architecture is transitions into operational mode and must support by the architecture team. The success of architecture team includes the following activities.

1. Domain experience to produce an acceptable design view and use case view.

2. Software technology experience to produce an acceptable process view, component view, and development view.

- The architecture team is responsible for system level quality which includes reliability, performance and maintainability.

- **Software Development Team:** The software development team is an application specific group.
- The software development team consists of several sub teams dedicated to develop group of components that require common skill set. The following are different skill sets. The following diagram shows activities of software development team.
-

Component teams

Artifacts

- Design set
- Implementation set
- Deployment set

Responsibilities

- Component design
- Component implementation
- Component stand-alone test
- Component maintenance
- Component documentation

Life-Cycle Focus

Inception	Elaboration	Construction	Transition
Prototyping support Make/buy trade-offs	Critical component design Critical component implementation and test Critical component baseline	Component design Component implementation Component stand-alone test Component maintenance	Component maintenance Component documentation

Commercial components: Specialist with detailed knowledge of commercial components is central to system architecture.

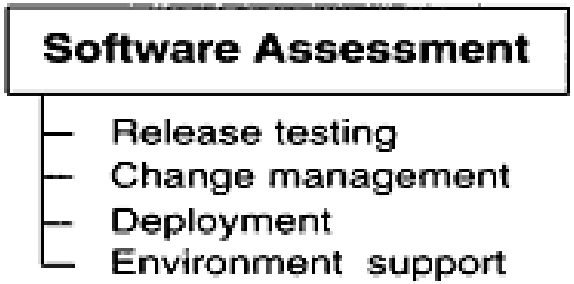
Database: Specialists with experience in the organization, storage, and retrieval of data.

Graphic user interface: Specialists with experience in the display organization, data presentation and user interaction necessary to support human input, output, and control needs.

Operating systems and networking: Specialists with experience in the execution of multiple software objects on a network of hardware resources.

- **Domain applications:** Specialists with experience in the algorithms, application processing, or business rules specific to the system
- The software development team is responsible for the quality of individual components, including all component development, testing, and maintenance.
- **Software Assessment Team:** We know that the software assessment is done independently because to assess the software quality and its pros and cons measure independently and to perform the assessment activities in concurrence with development activities.

- Due to the concurrence of assessment and development activities, software project plans are accelerated, change management, test planning, and test scenarios can also done in parallel with development. The following diagram shows the assessment activities.



Artifacts

- Deployment set
- SCO database
- User manual
- Environment
- Release specifications
- Release descriptions
- Deployment documents

Responsibilities

- Project infrastructure
- Independent testing
- Requirements verification
- Metrics analysis
- Configuration control
- Change management
- User deployment

Life-Cycle Focus

Inception	Elaboration	Construction	Transition
Infrastructure planning Primary scenario prototyping	Infrastructure baseline Architecture release testing Change management Initial user manual	Infrastructure upgrades Release testing Change management User manual baseline Requirements verification	Infrastructure maintenance Release baselining Change management Deployment to users Requirements verification

- A modern software process should employ use-case oriented or capability based testing done as a sequence of builds and mechanized by using two artifacts.

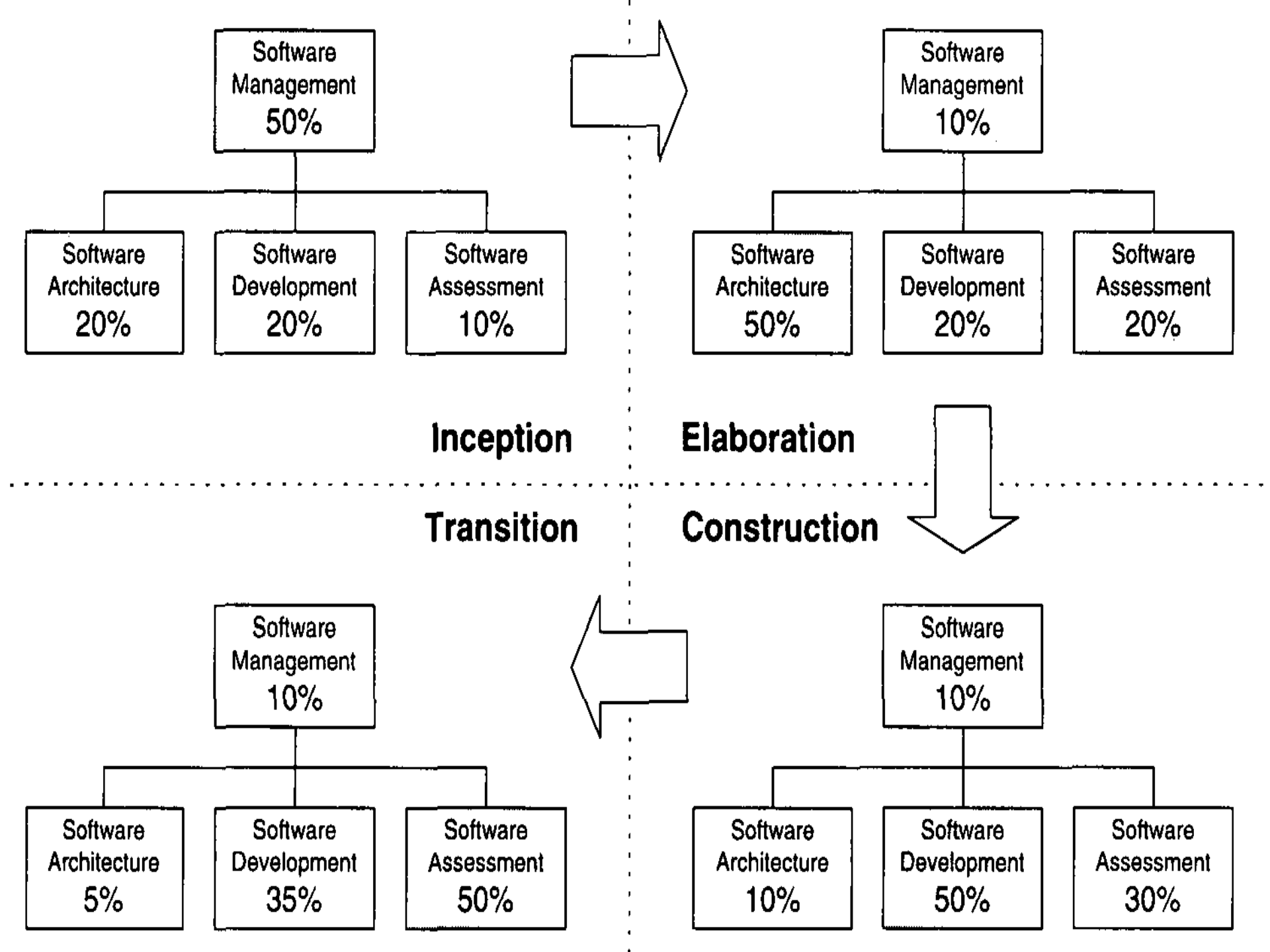
1. Release specifications.
2. Release description.

- We know that each release may have several components and evaluation criteria will document that the customer is expect to see at major milestones, and release descriptions will verify the test results.

- The final assessment iterations are equivalent to acceptance testing including the details of software plans, procedures and reports.
- The assessment team is responsible for the quality of baseline releases with respect to the requirements and customer expectations.
- The assessment team is also responsible for exposing any quality issues that affect the customer's expectations, whether or not these expectations are captured in the requirements.

Evolution of Organizations:

- We know that the project organizations are represented in the form of team structure.
- It is necessary to change the organization's structure in relevant to the project plan, which is part of the work breakdown architecture. The following diagram represents set of activities carried out by the organization according to the software project phases.


Software Management 50%
Software Architecture 20%
Software Development 20%
Software Assessment 10%
Inception
Software Management 10%
Software Architecture 50%
Software Development 20%
Software Assessment 20%
Elaboration
Construction
Software Management 10%
Software Architecture 10%
Software Development 50%
Software Assessment 30%
Transition
Software Management 10%
Software Architecture 5%
Software Development 35%
Software Assessment 50%

- **Inception Team:** Here the organization is focused on planning, with enough support from the other teams to ensure that the plans represent a consensus of all perspectives.
- **Elaboration Team:** Here the organization concentrates on architecture. At the point the organization tries to inculcate the architectural activities into the project.
- These activities are also supported by the software development team but also software assessment team. This kind of support is necessary to balance the software architecture activities.

- **Construction Team:** In good balanced software organizations, the most software activities are carried within software development and software assessment teams.
- **Transition Team:** This team mostly focused on consumer's product usage and feedback, depending on these further development activities are carried on.
- It is good and equally important to elaborate the details of each sub teams, responsibility, and work packages by planning the good work breakdown structure.
- All lower level team structures are prematurely plan, so it can reduce the project development inefficiencies.

- **Process Automation:** Many software development organizations are focused on developing mature process to improve the predictability of software management and performance of software business.
- To operate software business in profitable way in addition to process definition and development, process automation is also necessary.
- Today most software organizations are deal with the task of integrating their own environment and infrastructure for software development.

- This way leads to the selection of incompatible tools that have different information repositories are supported by developers, work on different platforms, use different jargon and based on different process assumptions.
- Automating the development process and establishing an infrastructure for supporting the various project workflows are important activities of the engineering stage of life cycle.
- They include tool selection, custom tool smiting, and process automation is necessary to perform against development plan with acceptable efficiency.

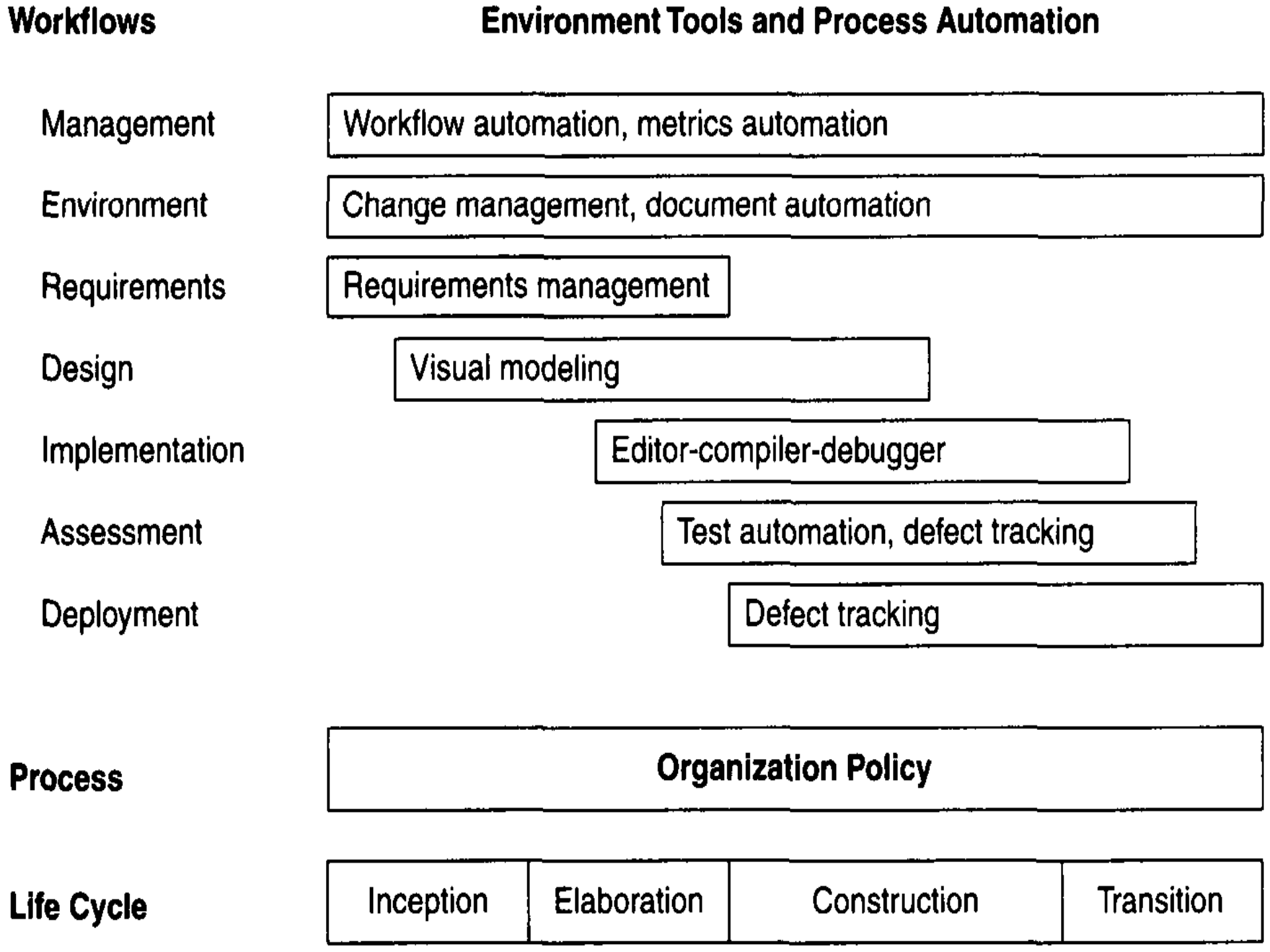
- We know that there three levels of process; each level should require some process automation for the process to be carried out with efficiency.
- **Meta process:** Organization's policies, procedures and practices for managing a software business.
- The automation support for this level is called infrastructure. An infrastructure is a inventory of preferred tools, artifacts templates, micro process guidelines, macro process guidelines, project performance repository, database organization skill set, and library of past project plans and results.

- **Macro process:** Project's policies, procedures and practices for producing a complete software product within certain cost, schedule, and quality. Here the automation support for the process is called environment.
- An environment is a specific collection of tools to produce a specific set of artifacts governed by project plan.
- **Micro process:** A project team policies, procedures and practices for achieving an artifact of software process. Here the automation support for the process is called a Tool.

- Naturally tools include requirement management, visual modeling, compilers, editors, debuggers, change management, metrics automation, document automation, test automation, test estimation and workflow automation.

Tools: Automated Building Blocks:

- Many tools are available to automate the software project development. Most of the software development tools map closely to one of the software process workflows. But each of the workflow requires distinct automation support. It should be represented in the following diagram.


Workflows
Environment Tools and Process Automation
Management
Workflow automation, metrics automation
Environment
Change management, document automation
Requirements
Requirements management
Design
Visual modeling
Implementation
Editor-compiler-debugger
Assessment
Test automation, defect tracking
Deployment
Defect tracking
Process
Organization Policy
Life Cycle
Inception
Elaboration
Construction
Transition

- **Management:** There are several opportunities for automating project planning and control activities of the management workflow. Software cost estimation tools and work breakdown structure tools are useful for generating planning artifacts.
- Sometimes the workflow management tools and project control panel can maintain online version status assessment of the project. Here the automation support can improve metrics collection and reporting.

- **Environment:** Configuration management and version controls are essential in modern iterative development process. These metrics are dependent measuring changes in software artifact baseline.
- **Requirements:** In modern software process, requirements are captured in vision document. But lower level requirements are collected by the process in iteration model through evaluation criteria.
- Iterative models allow the customer and the developer to work with real evolving version of the system. Practically, requirements can and must evolve along with an architecture and set of application increments.

- **Design:** The tools that support requirements, design, implementation, and assessment workflows are usually used together. The primary support of automation for designing is visual modeling, which is used for capturing design models, presenting them in human readable form and translating them into source code.
- **Implementation:** The implementation workflow primarily depending on programming environment such as editors, compilers, etc. it also include considerable integration with change management tools, visual modeling tools, and test automation tools to support productive iteration.

- **Assessment and Deployment:** The assessment workflow requires all the tools which are used in previous workflows. To increase the change freedom testing and document production must be automated.
- Defect tracking is another important tool that support assessment-it provides the change management instrumentation necessary to automate metrics and control baselines.

The Project Environment

- The project environment consists of three discrete states:

1. Prototype environment.
2. The development environment.
3. The maintenance environment.

- **The prototype Environment:** It is the platform for the evaluation of prototype architecture projects.
- Here we analyze the trade-offs of these projects during the inception and elaboration phases. This environment should also perform the following activities.

1. It analyze performance trade-offs and technical risks.

2. It determine make/buy trade-offs and feasibility studies for commercial products.

3. It also considers fault tolerance/dynamic reconfiguration trade-offs.

4. It analyzes the risks associated with transitioning to full scale implementation.

5. Development of test scenarios, tools, and instrumentation suitable for analyzing the requirements.

- **The development environment:** It should include full scale of development tools need to support the various process workflows and to support round-trip engineering to the maximum possible.
- **The maintenance environment:** It is a matured version of development environment. Sometimes maintenance environment is subset of the development environment delivered as one of the project's end product.
-

- The journey towards the fully developed software project process introduces new challenges and opportunities for management control of concurrent activities and for assessment of progress and quality.
- The real world projects should follow some management disciplines for the success of modern iterative development process.
- 1. Tools must be integrated to maintain consistency and traceability. Round trip engineering is the term used to describe this requirement for environment support of iterative development.

2. Change management must be automated and manage multiple iterations and to enable change freedom. Change is the fundamental rule of iterative development.

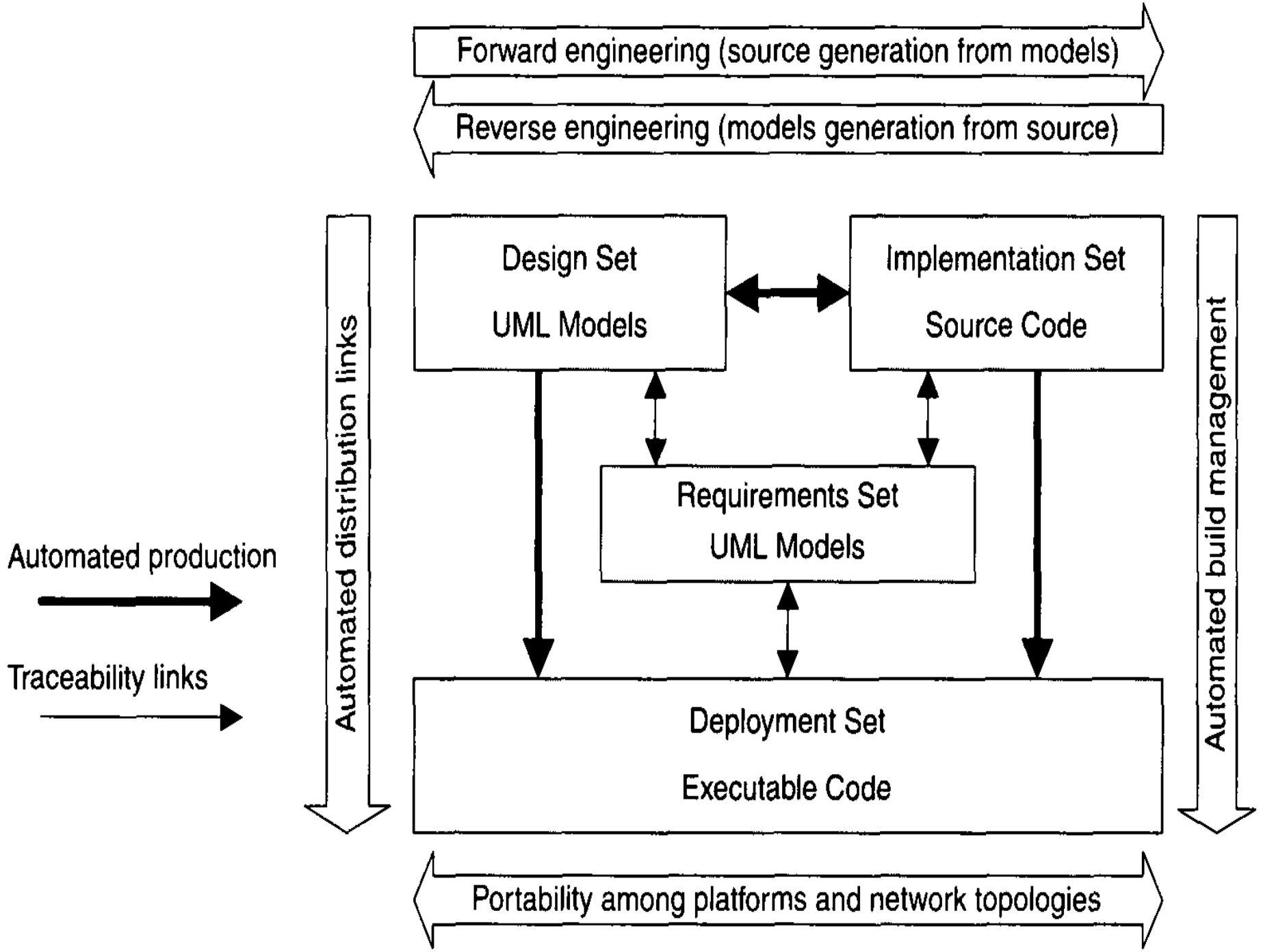
3. Organizational infrastructure allows the project architecture environment to be generated from common base of processes and tools.

- The common base provides the inter project consistency, reuse of training, reuse of lessons learned, and other strategic improvements.

4. Extending automation support for stakeholder environments enable further support for paperless exchange of information and effective review of engineering information.

Round Trip Engineering:

- A high level of automation support is required whenever a software organization maintains different information sets of the engineering artifacts for efficient and error free transition of data from one artifact to another.
- To achieve the consistency among the engineering artifacts, Round-trip engineering support is required. The following diagram shows important transitions occur between information repositories.


Forward engineering (source generation from models)
Reverse engineering (models generation from source)
Design Set
UML Models
Implementation Set
Source Code
Requirements Set
UML Models
Deployment Set
Executable Code
Automated distribution links
Automated build management
Automated production
Traceability links
Portability among platforms and network topologies

- The techniques of automated transition are applied to design models in order to translate them into respective source codes and process models.
- This is a straight forward phenomenon which is made possible through applications like Active X, and CORBA.
- We know that, automated transition of source code into executable code is made possible using compilers and linkers. So when we developing the project architecture we should consider what are the different components, platforms and languages are

- This kind of consideration necessary, when developing, managing and controlling large scale web projects.
- This complexity leads to the establishment of new requirements for configuration control and build management automation.
- Round-trip engineering is basically carried out to allow change freedom in data sources belonging to the software engineering.
- The configuration control associated with every technical artifact is important to maintain the consistency and error free presentation of changing product.

- It is not necessary that Round-trip engineering should consist of bi-directional translations. For example test cases are designed for logical object sets, but it is not possible to perform reverse engineering on the object from the test cases.
- The drawback of automated translation between the data sources is 100% completeness is not achieved. For example automated translation of design model into C++ code may generate structural and declaration information of source code presentation.
-

- **Software Change Orders:** Software change orders refers to the atomic component of software work that is allowed to create, alter the component present inside a configuration baseline.
- The main purpose of SCO is to divide, assign and schedule the software work against a software baseline which is well-established. In addition to this SCO is an important approach for analyzing the product improvement as well as its quality. The following diagram shows the essential components of SCO.

Title: _____

Description

Name: _____ Date: _____

Project: _____

Metrics

Category: _____ (0/1 error, 2 enhancement, 3 new feature, 4 other)

Initial Estimate

Breakage: _____

Rework: _____

Actual Rework Expended

Analysis: _____ Test: _____

Implement: _____ Document: _____

Resolution

Analyst: _____

Software Component: _____

Assessment

Method: _____ (inspection, analysis, demonstration, test)

Tester: _____ Platforms: _____ Date: _____

Disposition

State: _____ Release: _____ Priority _____

Acceptance: _____ Date: _____

Closure: _____ Date: _____

- In general, an SCO should be written against a single component so that it is easily allocated to a single individual. The basic fields of SCO are title, description, metrics, resolution, assessment, and disposition.
- ***Title:*** The originator is responsible to suggesting the title and it was finalized by the configuration control board (CCB). This filed should include an external software problem report, if the change was initiated by an external person (user).

- ***Description:*** The problem description includes the name of the originator, date of origination, CCB-assigned, SCO identifier, and relevant version identifiers of related software support.
- The textual problem description should provide as much detail as possible, along with attached code piece, display snapshots, error messages, and any other data that may help to isolate the problem and describe the change needed.
- ***Metrics:*** Metrics are gathered from individual SCO's for planning, scheduling, analyzing quality improvement activities. The metric field consists of different change categories.

- Type-0 indicate critical bug, Type-1 is normal bug, Type-2 indicate enhancement requirement, Type-3 indicate new features required, Type-4 may represent any other change.
- The metric field has six different items. Out of these, first two falls under the category of initial estimate and remaining under actual rework, expended category. The following are the six items.

a) The breakage represents the volume of change, and quantifies the extent of change and can be defined in units of SLOC, function points, files, components, or classes.

b) The rework estimates the amount of complexity that arises while changes are made.

c) The analysis item identifies the number of staff hours expended in understanding the required change.

d) The implementation item identifies the staff hours necessary to design and implement the resolution.

e) The test item identifies the amount of time required to test the resolution.

f) The documentation item identifies all effort expended in updating other artifacts such as user manual or release descriptions.

- ***Resolution:*** This field includes the name of the person responsible for implementing change, the components changed, the actual metrics and a description of change. This item also estimates the component fidelity.
- ***Assessment:*** This field describes the assessment technique as inspection, analysis, demonstration or test.
- It should also refer all existing test cases and new test cases executed, and it should identify all different test configurations, such as platforms, topologies, and compilers.
-

Disposition: The SCO is assigned one of the following states by configuration control board (CCB).

Proposed: Written, pending CCB review

***Accepted*:** CCB approved for resolution

Rejected: Closed, with justification, such as not a problem, duplicate, obsolete change, resolved by another SCO.

Archived: Accepted but postponed until a later release.

In progress: Assigned and actively being resolved by the development organization.

In assessment: Resolved by the development organization; being assessed by a test organization.

Closed: Completely resolved, with the concurrence of all CCB members.

Configuration Baseline: A specified collection of software components and its associated documentation, depending of change management and is upgraded, maintained, tested, and obsolesced as a unit is called configuration baseline. There are two general classes of baseline.

1.External product releases.

2.Internal testing releases.

A configuration baseline is a collection of components treated as a single unit and is a packaged exchange between groups.

- Generally most of the systems require three levels of baseline releases, Major, Minor and interim which are represented by the numbers N.M.X.
- The major release represents a new generation of product or project, while minor release represents the same basic product but with enhanced features, performance, or quality.

- An interim release corresponds to a developmental configuration that is intended to be transient.
- The major and minor releases are intended to be external product releases that are persistent and supported for a period of time.
- Once the software placed in a controlled baseline all changes are tracked. The following are various change categories.
- **Type-0:** It refers the critical failures, which are fixed before the external product releases. These failures can effects the performance and usability of the software project.

- **Type-1:** This type of errors does not produce any negative effects on the usefulness of the software project.
- But these errors create nuisances in critical use cases or to serious defects in secondary use cases that have a low probability of occurrence.
- **Type-2:** It is a change that causes the performance improvement rather than responding to a defect. Its sole purpose is to upgrade the performance, usability, testability.
- **Type-3:** It is a change that results from updating the requirements by producing the new additional features.

- **Type-4:** Changes that are not fall in any one of above categories is called type-4 changes.
- **Infrastructures:** From process automation view, the organization's infrastructure provides the organization's capital assets, including two key artifacts.
- Policy-That have the standards for software development process.
- Environment-That have inventory of tools. These tools are automation building blocks for configuration of project environment efficiently and economically.

- **Organization Policy:** The organization's policy is a handbook that defines the life cycle and the process primitives (major milestones, intermediate artifacts, engineering repositories, metrics, roles and responsibilities). The organization's policy provides answers to the following questions.
- What gets done?
- When does it get done?
- Who does it?
- How do we know that it is adequate?

- It is important to balance the organization policy because in several times for the organizations there were no institutionalized process or too much standardization and bureaucracy. The effective organization's policy has the following topics.

1. The organization's policies are concise. Avoid policies are thick bound books.

2.The organization's policies are confining to the practical to enforce them.

3.The policy statements avoid the word *should*, but the statements includes the *shoulds*. It represents policies need for concise mandatory standards.

4. Waivers are exception, not the rule.
5. Appropriate policy is written at the appropriate level.

- There are different organizations structures throughout the software development industry; most of them have three levels of organization. In which each level focus on different policy matters.

Highest organization level: In this level the software organization concentrate on;

- Strategic and long term process improvements
- General technology insertion and

- Comparability of project and business unit performance
- Mandatory quality control.

Intermediate line of business level: This state of organization promote;

- Tactical and short term process improvements.
- Domain specific technology insertion and training.
- Reuse of components, process, training, tools and personal experience.
- Compliance with organizational standards.

Lowest project level: This state of organization should have;

- Efficiency in achieving quality, schedule, and cost targets.
- Project specific training.
- Compliance with customer requirements.
- Compliance with organizational/business standards.
- Standardization should focus on line of business not on the top level organization or projects.
-

- But standardization of software development techniques and tools across line of business is difficult. So we may follow the standardization across the domains.
- The organization's policy is the defining document for the organization's software policies. In process assessment this policy document is an important artifact that represents what we do.
- Form this document, reviewers should be able to question and review projects and personnel and determine whether the organization does what it says.

- **Organization Environment:** The organization environment will provide automation of the process practically.
- It says that how the things get done and tools and techniques are used for process automation. The following are some components of automation building blocks.
- Standardized tool selections, which promote common workflows and higher ROI on training.
- Standard notion for artifacts, such as UML for all design models, or Ada for custom developed, reliability critical implementation artifacts.
- Tool adjuncts or customization.
- Activity templates.

UNIT – VII
PROJECT CONTROL AND PROCESS INSTRUMENTATION

- **Introduction:** The primary goal of the modern software development process is deal with central management issues of complex software, these are;

1. Getting the right design by focusing on the architecture.

2.Managing risk through iterative development.

3.Reducing the complexity with component based techniques.

4.Making software progress and quality tangible through instrumental change management.

5.Automating the overhead and bookkeeping through the use of round trip engineering and integrated environments.

- Software metrics instrument the activities and products of the software development/integration process.
- In modern software development process, software metrics are simple objective measures of how various perspectives of product and project are changing.
- The quality of software products and the progress made toward project goals must be measureable throughout the software development lifecycle.

- Here the goals of software metrics are to provide the development team and management team with the following.
- An accurate assessment of progress to date.
- Insight into the quality of developing software product.
- A basis for estimating the cost and schedule for completing the product with increasing accuracy over time.

The Seven Core Metrics:

- There are several metrics are available for managing software process. Here we discuss seven metrics which are applicable to any software project. Out of these three are management indicators and four are quality indicators.

- Management indicators:

1. Work and progress.
2. Budgeted cost and expenditures.
3. Staffing and team dynamics.

- Quality indicators:

1. Changing traffic and stability.
2. Breakage and modularity.
3. Rework and adaptability.
4. Mean time between failures and maturity.

- Each metric has two dimensions: a static value used as an objective and the dynamic trend used to manage the achievement of that objective.
- The metric values provide on dimension of insight, metric trends provide a more important perspective for managing the process.

- Metric trends with respect to time provide insight into how the process and product are developing.
- Measuring the change is the most important aspect of the metrics. The following table shows the seven core metrics.
-

METRIC	PURPOSE	PERSPECTIVES
Work and progress	Iteration planning, plan vs. actuals, management indicator	SLOC, function points, object points, scenarios, test cases, SCOs
Budgeted cost and expenditures	Financial insight, plan vs. actuals, management indicator	Cost per month, full-time staff per month, percentage of budget expended
Staffing and team dynamics	Resource plan vs. actuals, hiring rate, attrition rate	People per month added, people per month leaving
Change traffic and stability	Iteration planning, management indicator of schedule convergence	SCOs opened vs. SCOs closed, by type (0,1,2,3,4), by release/component/subsystem
Breakage and modularity	Convergence, software scrap, quality indicator	Reworked SLOC per change, by type (0,1,2,3,4), by release/component/subsystem
Rework and adaptability	Convergence, software rework, quality indicator	Average hours per change, by type (0,1,2,3,4), by release/component/subsystem
MTBF and maturity	Test coverage/adequacy, robustness for use, quality indicator	Failure counts, test hours until failure, by release/component/subsystem

- The seven core metrics can be used in number of ways to help manage projects and organizations.
- The seven core metrics are based on common sense and field experience with both successful and unsuccessful metrics programs. The following are the attributes of the metrics.

1. They are simple, objective, easy to collect, easy to interpret, and hard to misinterpret.
2. Collection can be automated and nonintrusive.
3. They provide consistent assessments throughout the lifecycle and are derived from the product baselines rather than subjective assessments.

4. They are useful for both management and engineering personnel for communicating progress and quality in a consistent format.

5. Their reliability improves across lifecycle.

- Metrics applied to the engineering stage is less accurate than those applied to the production stage. So prescribed metrics are adapted to the production stage, when cost risk is high and management value is calculated.

Management Indicators:

- There are three fundamental sets of management metrics: Technical progress, Financial status and Staffing progress. By examining these perspectives, management can generally assess whether a project is on budget and on schedule.
- **Work and Progress:** The purpose of this metric is determining a planned estimation of the work and then monitoring progress against the plan. The purpose of this metric is iterative planning, plan versus actual, and management indicators. The perspectives of the metrics as follows;

- Software architecture team: which demonstration of use-cases.
- Software development team: Source line of code under baseline change management and closing SCOs.
- Software assessment team: Here SCOs are opened followed by the execution of test hours, till the evaluation criteria is met.
- Software management team: completion of milestones.
-

- **Budgeted cost and expenditures:** Measurement of cost expenditures is required throughout the project life cycle for providing management control.
- The purpose of this metric is financial insight, management indicator, and plan versus actual.
- Financial progress monitoring can be achieved by using the earned value of the system, which provides the clear understanding of cost and schedule of the project. The following are basic parameters of earned value of the system

- Expenditure plan: It specifies the planned expenditure over a planned schedule for a project.
- Actual progress: It is the actual cost expenditure over an actual schedule in a project called “actual cost”.
- Earned value: It is the value that denotes the planned cost of the actual project.

Cost variance: It is the difference between actual cost and earned value. Cost variance = actual cost – earned value.

If cost variance >0 then it leads to the over budget else it leads to under budget.

- Schedule variance: It is the difference between the planned cost and earned value. If the schedule variance is positive then it leads to the behind scheduled situations else it leads to ahead of scheduled situations.
- The perspective of budgeted cost and expenditure metric are; measuring cost per month, percentage of splendid budget and full time staff per month.
- **Staffing and Team dynamics:** It refers the changes in project personnel over the time.
- Here we consider the maintenance team is

- The purpose of this metric is resource planning versus actual resources, rate of hiring personnel and deterioration rate.
- The main perspective of this metric is number of persons added per month and number of persons leaving per month.

Quality Indicators:

- The four quality indicators are primarily based on measurement of software change across evolving baselines of engineering data.

- **Change traffic and stability:** Change traffic is one specific indicator for progress and quality.
- Change traffic is defined as the number of software change orders opened and closed over the life cycle of the project.
- . This metric can measured by the change type, by release, across all releases, by team, by components, by subsystem, and soon.
- The change traffic metric is linked with work and progress metrics, and it provides insight into the stability of the software project.

- **Breakage and Modularity:** Breakage is defined as the average extent of change, which is the amount of software baseline that needs rework (in SLOC, function points, etc).
- Maturity is the average breakage trend over time. For successful projects the trend is in decreasing order.
- This metric provides insight into the gentle character of software change. In a mature iterative development process, earlier changes result more scrap than later changes.

- **Rework and Adaptability:** Rework is defined as the average cost of change, which is the effort to analyze, resolve, and retest all changes to software baselines.
- Adaptability is defined as the rework trend over time.
- All changes are equal. Some changes can be made in staff-hours, others take staff-weeks.
- This metric provides insight into rework measurement. In iterative development process early changes require more rework than later changes.

- **MTBF and Maturity:** MTBS (Mean Time Between Failures) is an indicator that computes the average time consumed between the software failures by dividing the test hours with *type-0 and type-1* software change orders (SCOs).
- Maturity metrics specifies how mean time between failures varies over time by comparing project schedule against the released baseline.
- To know more about maturity, traditional testing methods were implemented on conventional software system. The purpose of doing this was to test each and every line of code.

Life Cycle Expectations:

- There is no mathematical or formal derivation for using the seven core metrics. But the following are the reasons for using them in project development lifecycle.

1. The quality indicators are derived from the product rather than from the artifacts.

2. They provide insight into the waste generated by the process. Scrap and rework metrics are a standard measurement perspective of most manufacturing processes.

4. They recognize the inherently dynamic nature of an iterative development process.

5. The combination of insight from the current value and the current trend provides reliable indicators for management action.

- The actual values of these metrics can vary widely across projects, organizations, and domains.

- **Pragmatic software metrics:**

- Measuring is useful, it provides data to the decision makers to ask right questions, understand the context, and make objective decisions.

- Due to the dynamic nature of the software projects, these measures must be available at any time, translated into any subject for developing the product and maintain trends. The following are some of the characteristics of good metrics.

1.It is considered meaningful by the customer, manager, and performer: The metric should be understandable to the customers, managers and developers. If any of the people find difficulty in understanding the metrics, then the metric cannot be applied.

- The customer can use the metric only when the developer first can understand the metric properly.

2. *It demonstrates quantifiable correlation between process perturbations and business performance:* The main goal of the software organization is, increase in revenue, reduction of cost, increases in margin. To achieve these goals, metrics must explain properly how changes in software process can have an effect on the performance of organization.

3. *It is objective and unambiguously defined:* The metric objective should clear and precise. The metric objectivity should translate into some form of numeric representation such as numbers, percentages, ratios etc. and also into textual representations such as excellent, good, fair etc.

4. It displays trends: It is a important characteristic, the change in metric's value with respect to time, subsequent projects, subsequent releases is an important perspective. Metrics never drives an action directly, generally metric present perspective.

5. *It is natural byproduct of process:* The metric does not introduce new artifacts or overhead activities; it is derived directly from the main stream engineering and management workflows.

6. It is supported by automation: The most successful metrics are collected and reported by automated tools, because software tools require exact definition of the data they process.

Metrics Automation:

- There is a possibility to automate the project control activities of the software project.
- The Software Project Control Panel (SPCP) maintains an on-line version of display panel.
- This panel can support standard features such as warning lights, thresholds, variable scales, digital formats, and analog format to present overview of present project situation. It can also provide detailed situation analysis.

- This automation can improve management insight into progress and quality trends and improve the acceptance of metrics by the engineering team. To implement the SPCP it is necessary to develop the following things.
- *Metrics primitives:* indicators, trends, comparisons, and progressions.
- *A graphical user interface:* GUI support for software project manager.
- *Metrics collection agents:* Data extractions from environment tools maintain the engineering notions for various artifact sets.

- *Metrics data management server:* Data management support for populating the metric displays of the GUI and storing data extracted by agents.
- *Metric definitions:* Actual metrics presentations for requirements progress, design progress, implementation progress, assessment progress, and other progress dimensions.
- *Actors:* These are monitors and administrators.

- The whole display is called panel. Within a panel are graphical objects, which are types of layouts for information.
- Each graphical object displays a metric. A panel typically contains a number of graphical objects positioned in a particular geometric layout.
- Metric can display in two modes: one is value-refer to a given point in time.
- The second is graph-refer to multiple and consecutive points in time.
- Metrics can display with or without control values. A control value is an existing expectation, either absolute or relative, that is used for

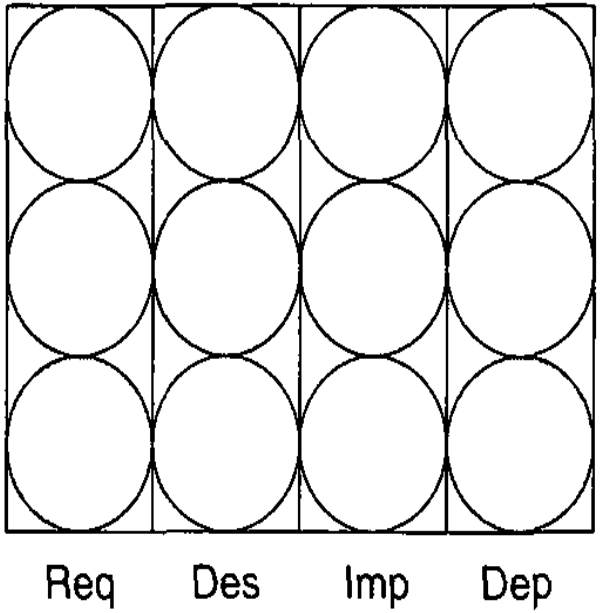
- Indicators may display data in formats that are binary such as black and white and tertiary such as red, yellow and green, in digital, or in other enumerated types.
- Indicators provide mechanism that can be summarized a condition or circumstance associated with another metric, or relationships between metrics and their associated control values.
- A trend graph presents values over time and permits upper and lower thresholds to be defined.
- A comparison graph presents multiple values together over time. Convergence or divergence among values may link to an indicator

- A progression graph presents percent complete, where elements of progress are shown as transitions between states and an earned value is associated with each state.
- Metrics information can be summarized in user defined level.
- Users can define summary structures for low levels.
- The following is a SPCP diagram for a sample project.

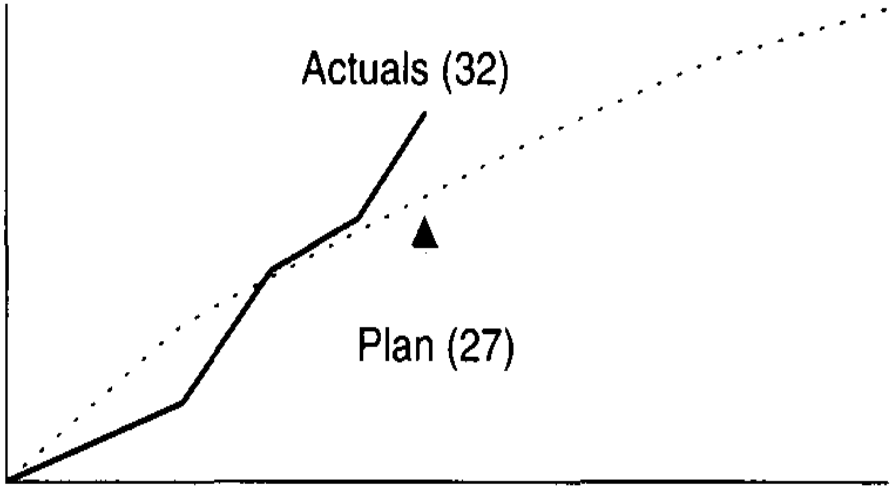
Top-Level WBS Activities

Management		– 4% ↓
Environment		+ 1% ↑
Requirements		+ 6% ↑
Design		– 5% ↓
Implementation		–25% ↓
Assessment		– 2% ↑
Deployment		– 2% ↑

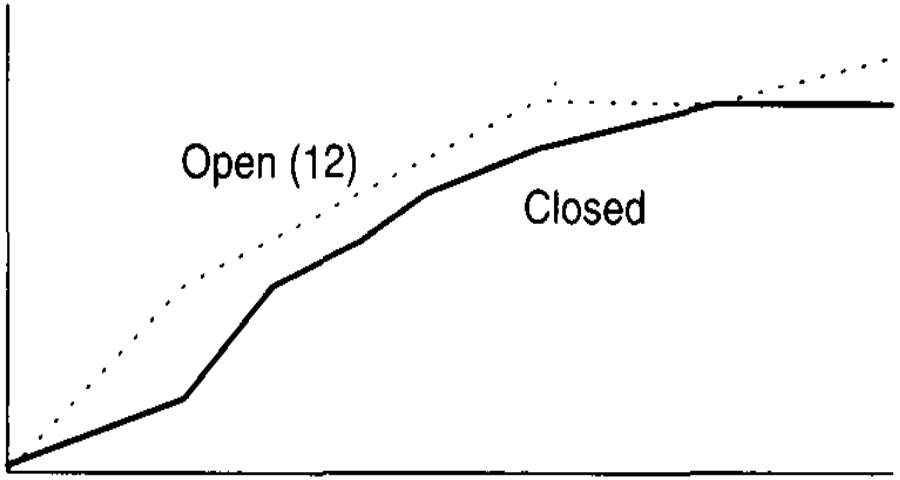
Technical Artifacts



Milestone Progress



Action Item Progress



1. *Project activity status:* The graphical object in the upper left provides an overview of the status of top-level WBS. Seven elements are coded in different colors
2. *Technical artifact status:* The graphical object in the upper right provides an overview of the status of the evolving technical artifacts.
3. *Milestone progress:* The graphical object in the lower left provides a progress assessment of the achievement of milestones against plan and provides indicators of the current value.
4. *Action item progress:* the graphical object in lower right provides a different perspective of progress.

- An SPCP should support the capability to drill down into the details for any given metric. The following top-level use case, which describe the basic operational concept for SPCP.
- Start SPCP. The SPCP starts and shows the most current information that was saved when the user last used the SPCP.
- Select a panel preference. The user selects from a list of previously defined default panel preferences. The SPCP displays the preference selected.

- Select a value or graph metric. The user selects whether the metric should be displayed for a given point in time or in a graph as trend.
- Select to superimpose controls. The user has a freedom to select superimpose the point in time to graph and vice versa.
- Drill down trend. The use points to a graphical object display, point in time and frills down to view the trends for the metric.
- Drill down to point in time. The user points graphical object display and frills down to view the values for the metric.

- Drill down lower level information. The user points to a graphical object display and drills down to view next level of information.
- Drill down to lower level indicators. The user points to a graphical object display and drills down to view the breakdown of next level indicators.
- The SPCP is a metrics automaton approach that collects, organizes, and reports values and trends extracted directly from the evolving engineering artifacts.
- **Process Discriminates:**