UNIT 5:

CCPDS-R Case Study and Future Software Project Management Practices Modern Project Profiles, Next-Generation software Economics, Modern Process Transitions

COMMAND CENTER PROCESSING AND DISPLAY SYSTEM-REPLACEMENT (CCPDS-R)

- The Command Center Processing and Display Sys-tem-Replacement (CCPDS-R) project was performed for the U.S. Air Force by TRW Space and Defense in Redondo Beach, California. The entire project included systems engineering, hardware procurement, and software development, with each of these three major activities consuming about one-third of the total cost. The schedule spanned 1987 through 1994.

a The metrics histories were all derived directly from the artifacts of the project's process. These data were used to manage the project and were embraced by practitioners, managers, and stakeholders.

There are very few well-documented projects with objective descriptions of what worked, what didn't, and why. This was one of my primary motivations for providing the level of detail contained in this appendix. It is heavy in project-specific details, approaches, and results, for three reasons:

1. Generating the case study wasn't much work. CCPDS-R is unique in its detailed and automated metrics approach. All the data were derived directly from the historical artifacts of the project's process.

2. This sort of objective case study is a true indicator of a mature organization and a mature project process. The absolute values of this historical perspective are only marginally useful. However, the trends, lessons learned, and relative priorities are distinguishing characteristics of successful software development.

3. Throughout previous chapters, many management and technical approaches are discussed generically. This appendix provides in a real-world example at least one relevant benchmark of performance

- The CCPDS-R project produced a large-scale, highly reliable command and control system that provides missile warning information used by the National Command Authority. The procurement agency was Air Force Systems Command Headquarters, Electronic Systems Division, at Hanscom Air Force Base, Massachusetts. The primary user was US Space Command, and the full-scale development contract was awarded to TRWs Systems Integration Group in 1987. The CCPDS-R contract called for the development of three subsystems:

1. The Common Subsystem was the primary missile warning system within the Cheyenne Mountain Upgrade program. It required about 355,000 source lines of code, had a 48-month software development schedule, and laid the foundations for the subsystems that followed (reusable components, tools, environment, process, procedures). The Common Subsystem included a primary installation in Cheyenne Mountain, with a backup system deployed at Offutt Air Force Base, Nebraska.

2. The Processing and Display Subsystem (PDS) was a scaled-down missile warning display system for all nuclear-capable commanders-in-chief. The PDS software (about 250,000 SLOC) was fielded on remote, read-only workstations that were distributed worldwide.

3. The STRATCOM Subsystem (about 450,000 SLOC) provided both missile warning and force management capability for the backup missile warning center at the command center of the Strategic Command

CCPDS-R LIFE-CYCLE OVERVIEW

- The CD phase was very similar in intent to the inception phase. The primary products were a system specification (a vision document), an FSD phase proposal (a business case, including the technical approach and a fixed-price-incentive and award-fee cost proposal), and a software development plan. The CD phase also included a system design review, technical interchange meetings with the government stakeholders (customer and user), and several contract-deliverable documents. These events and products enabled the FSD source selection

to be based on demonstrated performance of the contractor-proposed team as well as the FSD proposal.

- From a software perspective, there was one additional source selection criterion included in the FSD proposal activities: a software engineering exercise. This was a unique but very effective approach for assessing the abilities of the two competing contractors to perform software development. The Air Force was extremely concerned with the overall software risk of this project: Recent projects had demonstrated dismal software development performance. The Air Force acquisition authorities had also been frustrated with previous situations in which a contractor's crack proposal team was not the team committed to perform after contract award, and contractor proposals exaggerated their approaches or capabilities beyond what they could deliver.

  CCPDS-R was also a very large software development activity and was one of the first projects to use the Ada programming language. There was serious concern that the Ada development environments, contractor processes, and contractor training programs might not be mature enough to use on a full-scale development effort. The purpose of the software engineering exercise was to demonstrate that the contractor's proposed software process, Ada environment, and software team were in place, were mature, and were demonstrable

- The software engineering exercise occurred immediately after the FSD proposals were submitted. The customer provided both bidders with a simple two-page specification of a "missile warning simulator." This simulator had some of the same fundamental requirements as the CCPDS-R full-scale system, including a distributed architecture, a flexible user interface, and the basic processing scenarios of a simple CCPDS-R missile warning thread. The exercise requirements included the following:

- Use the proposed software team.
- Use the proposed software development techniques and tools.
- Use the FSD-proposed software development plan.

- Conduct a mock design review with the customer 23 days after receipt of the specification.
- Four primary use cases were elaborated and demonstrated.
- A software architecture skeleton was designed, prototyped, and documented, including two executable, distributed processes; five concurrent tasks (separate threads of control); eight components; and 72 component-to-component interfaces.
- A total of 4,163 source lines of prototype components were developed and executed. Several thousand lines of reusable components were also integrated into the demonstration.
- Three milestones were conducted and more than 30 action items resolved.
- Production of 11 documents (corresponding to the proposed artifacts) demonstrated the automation inherent in the documentation tools.
- The Digital Equipment Corporation VAX/VMS tools, Rational R1000 environment, LaTeX documentation templates, and several custom-developed tools were used.
- Several needed improvements to the process and the tools were identified. The concept of evolving the plan, requirements, process, design, and environment at each major milestone was considered potentially risky but was implemented with rigorous change management.
- In preparing for the CCPDS-R project, TRW placed a strong emphasis on evolving the right team. The CD phase team represented the essence of the architecture team which is responsible for an efficient engineering stage. This team had the following primary responsibilities:
- Analyze and specify the project requirements
- Define and develop the top-level architecture
- Plan the FSD phase software development activities
- Configure the process and development environment

- Establish trust and win-win relationships among the stakeholders

1.Network Architecture Services (NAS). This foundation middleware provided reusable components for network management, interprocess communications, initialization, reconfiguration, anomaly management, and instrumentation of software health, performance, and state. This CSCI was designed to be reused across all three CCPDS-R subsystems.

2. System Services (SSV). This CSCI comprised the software architecture skeleton, real-time data distribution, global data types, and the computer system operator interface.

3. Display Coordination (DCO). This CSCI comprised user interface control, display formats, and display population.

4. Test and Simulation (TAS). This CSCI comprised test scenario generation, test message injection, data recording, and scenario playback.

5. Common Mission Processing (CMP). This CSCI comprised the missile warning algorithms for radar, nuclear detonation, and satellite early warning messages.

6. Common Communications (CCO). This CSCI comprised external interfaces with other systems and message input, output, and protocol management

MODERN PROJECT PROFILES

Continuous Integration

In the iterative development process, firstly, the overall architecture of the project is created and then all the integration steps are evaluated to identify and eliminate the design errors. This approach eliminates problems such as down stream integration, late patches and shoe-horned software fixes by implementing sequential or continuous integration rather than implementing large-scale integration during the project completion

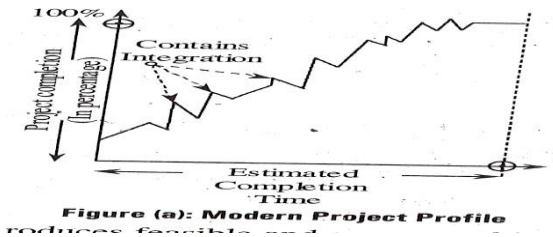

Figure (a): Modern Project Profile

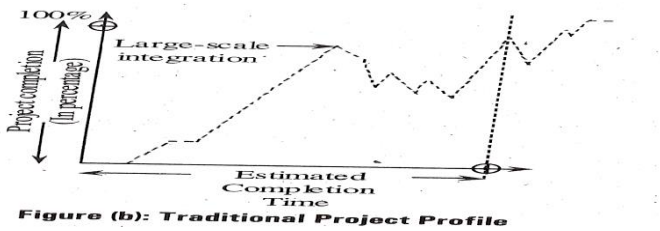

Figure (b): Traditional Project Profile

- Moreover, it produces feasible and a manageable design by delaying the 'design breakage' to the engineering phase, where they can be efficiently resolved. This can be one by making use of project demonstrations which forces integration into the design phase.
- With the help of this continuous integration incorporated in the iterative development process, the quality tradeoffs are better understood and the system features such as system performance, fault tolerance and maintainability are clearly visible even before the completion of the project.

In the modern project profile, the distribution of cost among various workflows or project is completely different from that of traditional project profile as shown below

Software Engineering Workflows	Conventional Process Expenditures	Modern process Expenditures
Management	5%	10%
Environment	5%	10%
Requirements	5%	10%
Design	10%	15%
Implementation	30%	25%
Assessment	40%	25%

Deployment	5%	5%
Total	100%	100%

As shown in the table, the modern projects spend only 25% of their budget for integration and Assessment activities whereas; traditional projects spend almost 40% of their total budget for these activities. This is because, the traditional project involve inefficient large-scale integration and late identification of design issues

EARLY RISK RESOLUTION

- In the project development lifecycle, the engineering phase concentrates on identification and elimination of the risks associated with the resource commitments just before the production stage. The traditional projects involve, the solving of the simpler steps first and then goes to the complicated steps, as a result the progress will be visibly good, whereas, the modern projects focuses on 20% of the significant requirements, use cases, components and risk and hence they occasionally have simpler steps.
- To obtain a useful perspective of risk management, the project life cycle has to be applied on the principles of software management. The following are the 80:20 principles.
- The 80% of Engineering is utilized by 20% of the requirementsBefore selecting any of the resources, try to completely understand all the requirement because irrelevant resource selection (i.e., resources selected based on prediction) may yield severe problems.
- 80% of the software cost is utilized by 20% of the components
- Firstly, the cost-critical components must be elaborated which forces the project to focus more on controlling the cost.
- 80% of the bugs occur because of 20% of the components

- Firstly, the reliability-critical components must be elaborated which give sufficient time for assessment activities like integration and testing, in order to achieve the desired level of maturity.
- 80% of the software scrap and rework is due to 20% if the changes.
- The change-critical components r elaborated first so that the changes that have more impact occur when the project is matured.
- 80% of the resource consumption is due to 20% of the components.
- Performance critical components are elaborated first so that, the trade-offs with reliability; changeability and cost-consumption can be solved as early as possible.
- 80% of the project progress is carried-out by 20% of the people
- It is important that planning and designing team should consist of best processionals because the entire success of the project depends upon a good plan and architecture.
- The following figure shows the risk management profile of a modern project.

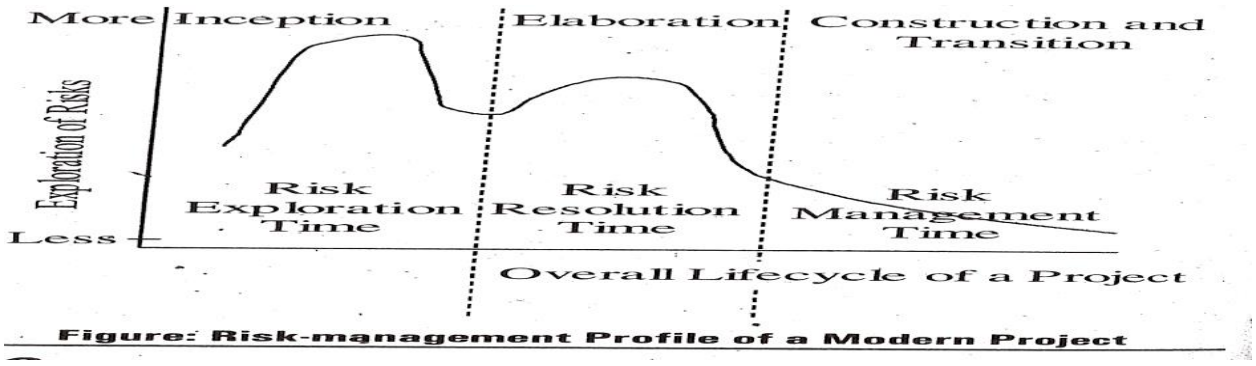

Figure: Risk-management Profile of a Modern Project

EVOLUTIONARY REQUIREMENTS

- The traditional methods divide the system requirements into subsystem requirements which in turn gets divided into component requirements. These component requirements are further divided into unit requirements. The reason for this systematic division is to simplify the traceability of the requirements.

- In the project life cycle the requirements and design are given the first and the second preference respectively. The third preference is given to the traceability between the requirement and the design components these preferences are given in order to make the design structure evolve into an organization so it parallels the structure of the requirements organization.
- Modern architecture finds it difficult to trace the requirements because of the following reasons.
    - Usage of Commercial components
    - Usage of legacy components
    - Usage of distributed resources
    - Usage of object oriented methods.

Moreover, the complex relationships such as one-one, many-one, one-many, conditional, time-based and state based exists the requirements statement and the design elements

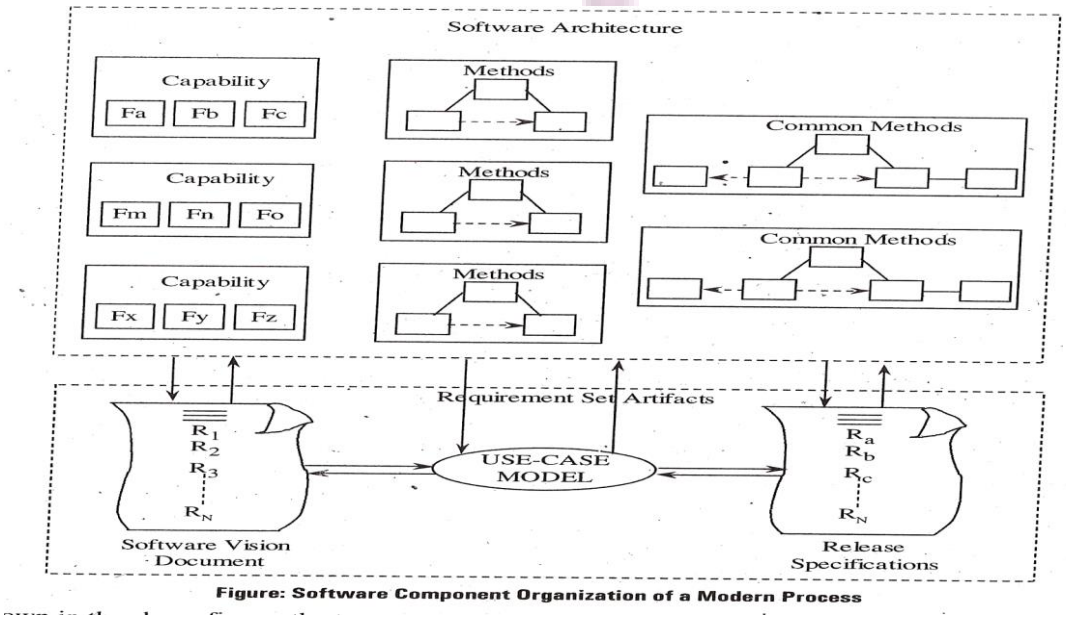

Figure: Software Component Organization of a Modern Process

As shown in the above figure, the top category system requirements are kept as the vision whereas, those with the lower category are evaluated. The motive behind theses artifacts is to gain fidelity with respect to the progress in the project lifecycle. This serves as a significant different from the traditional approach because, in traditional approach the fidelity is predicted early in the project life cycle

TEAMWORK AMONG STAKEHOLDERS

- Most of the characteristics of the classic development process worsen the stakeholder relationship s which in turn makes the balancing of requirement product attributes and plans difficult. An iterative process which ahs a good relationship between the stakeholders mainly focuses on objective understanding by each and every individual stakeholder. This process needs highly skilled customers, users and monitors which have experience in both the application as well as software. Moreover, this process requires an organization whose focus is on producing a quality product and achieves customer satisfaction.

The table below shows the tangible results of major milestones in a modern process

Obvious result	**Actual result**
Demonstration at early stage reveals the design issued and uncertainty in a tangible form.	Demonstration firstly reveals the significant assets and risks associated with complicated software systems such that they can be worked out at the time of setting the life-cycle goals.
Non-Complaint design	Various perspective like requirements use cases etc are observed in order to completely understand the compliance.
Issues of influential requirements are reveals	Both the requirement changes and the design trade-offs

but without traceability	are considerably balanced.
The design is considered to be "guilty until its innocency is proved.	The engineering issues can be integrated into the succeeding iteration's plans.

- From the above table, it can be observed that the progress of the project is not possible unless all the demonstration objectives are satisfied. This statement does not present the renegotiation of objectives, even when the demonstration results allow the further processing of trade offs present in the requirement, design, plans and technology.
- Modern iterative process that rely on the results of the demonstration need al its stakeholders to be well-educated and with a g good analytical ability so as to distinguish between the obviously negative results and the real progress visible. For example, an early determined design error can be treated as a positive progress instead to a major issue.

Principles of Software Management

- Software management basically relies on the following principles, they are,

1. Process must be based on architecture-first approach

If the architecture is focused at the initial stage, then there will be a good foundation for almost 20% of the significant stuff that are responsible for the overall success of the project. This stuff include the requirements, components use cases, risks and errors. In other words, if the components that are being involved in the architecture are well known then the expenditure causes by scrap and rework will be comparatively less.

2. Develop an iterative life-cycle process that identifies the risks at an early stage

An iterative process supports a dynamic planning framework that facilitates the risk management predictable performance moreover, if the risks are resolved earlier, the predictability will be more and the scrap and rework expenses will be reduced.

3.After the design methods in-order to highlight components-based development.

The quantity of the human generated source code and the customized development can be reduced by concentrating on individual components rather than individual lines-of-code. The complexity of software is directly proportional to the number of artifacts it contains that is, if the solution is smaller then the complexity associated with its management is less.

4. *Create a change management Environment*

Highly-controlled baselines are needed to compensate the changes caused by various teams that concurrently work on the shared artifacts.

5. *Improve change freedom with the help of automated tools that support round-trip engineering.*

The roundtrip-engineering is an environment that enables the automation and synchronization of engineering information into various formats. The engineering information usually consists requirement specification, source code, design models test cases and executable code. The automation of this information allows the teams to focus more on engineering rather than dealing with over head involved

Design artifacts must be captured in model based notation.

The design artifacts that are modeled using a model based notation like UML, are rich in graphics and texture. These modeled artifacts facilitate the following tasks.

- **Complexity control**
- **Objective fulfillment**
- **Performing automated analysis**

7. *Process must be implemented or obtaining objective quality control and estimation of progress.*

The progress in the lifecycle as well as the quality of intermediately products must be estimated and incorporated into the process. This can be done with the help of well defined estimation mechanism that are directly derived from the emerging artifacts. These mechanisms provide detailed information about trends and correlation with requirements.

8. Implement a Demonstration-based Approach for Estimation of intermediately Artifacts

This approach involves giving demonstration on different scenarios. It facilitates earl integration and better understanding of design trade-offs. Moreover, it eliminates architectural defects earlier in the lifecycle. The intermediately results of this approach are definitive

9.The Points Increments and generations must be made based on the evolving levels of detail

Here, the 'levels of detail' refers to the level of understanding requirements and architecture. The requirements, iteration content, implementations and acceptance testing can be organized using cohesive usage scenarios.

10. Develop a configuration process that should be economically scalable

The process framework applied must be suitable for variety of applications. The process must make use of processing spirit, automation, architectural patterns and components such that it is economical and yield investment benefits.

BEST PRACTICES ASSOCIATED WITH SOFTWARE MANAGEMENT

- According to airline software council, there are about nine best practices associated with software management. Theses practices are implemented in order to reduce the complexity of the larger projects and to improve software management discipline.
- The following are the best practices of software management:

1. *Formal Risk Management:* Earlier risk management can be done by making use of iterative life cycle process that identifies the risks at early stage.

2. *Interface Settlement:* The interface settlement is one of the important aspects of architecture first approach because; obtaining architecture involves the selection of various internal and external interfaces that are incorporated into the architecture.

3. Formal Inspections: There are various defect removal strategies available. Formal inspection is one of those strategies. However this is the least important strategy because the cost associated with human recourses is more and is defect detection rate for the critical architecture defects is less

Management and scheduling based on metrics: This principle is related to the model based approach and objective quality control principles. It states to use common notations fro the artifacts so that quality and progress can be easily measured.

5. Binary quality Gates at the inch-pebble level: The concept behind this practice is quite confusing. Most of the organizations have misunderstood the concept and have developed an expensive and a detailed plan during the initial phase of the lifecycle, but later found the necessity to change most of their detailed plan due to the small changes in requirements or architectural. This principle states that first start planning with an understanding of requirements and the architecture. Milestones must be established during engineering stage and inch-pebble must be followed in the production stage.

6. Plan versus visibility of progress throughout the progress: This practice involves a direct communication between different team members of a project so that, they can discuss the significant issues related to the project as well as notice the progress of the project in-comparison to their estimated progress

7.Identifying defects associated with the desired quality: This practice is similar to the architecture-first approach and objective quality control principles of software management. It involves elimination of architectural defects early in the life-cycle, thereby maintaining the architectural quality so as to successfully complete the project.

8. Configuration management: According to Airline software council, configuration management serves as a crucial element for controlling the complexity of the artifacts and for tracing the changes that occur in the artifacts. This practice is similar to the change management principle of software management and prefers automation of components so as to reduce the probability of errors that occur in the large-scale projects.

9.Disclose management accountability: The entire managerial process is disclosed to al the people dealing with the project

NEXT GENERATION SOFTWARE COST MODELS

- In comparison to the current generation software cost modes, the next generation software cost models should perform the architecture engineering and application production separately. The cost associated with designing, building, testing and maintaining the architecture is defined in terms of scale, quality, process, technology and the team employed.
- After obtaining the stable architecture, the cost of the production is an exponential function of size, quality and complexity involved.
- The architecture stage cost model should reflect certain diseconomy of scale (exponent less than 1.0) because it is based on research and development-oriented concerns. Whereas the production stage cost model should reflect economy of scale (exponent less than 1.0) for production of commodities.
- **The next generation software cost models should be designed in a way that, they can assess larger architectures with economy of scale. Thus, the process exponent will be less than 1.0 at the time of production because large systems have more automated proves components and architectures which are easily reusable.**
- **The next generation cost model developed on the basis of architecture-first approach is shown below.**
- **At architectural engineering Stage**
    - A Plan with less fidelity and risk resolution
    - It is technology or schedule-based
    - It has contracts with risk sharing
    - Team size is small but with experienced professionals.

- The architecture team, consists of small number of software engineers
- The application team consists of small number of domain engineers.
- The output will be an executable architecture, production and requirements
- The focus of the architectural engineering will be on design and integration of entities as well as host development environment.
- It contains two phases they are inspection and elaboration

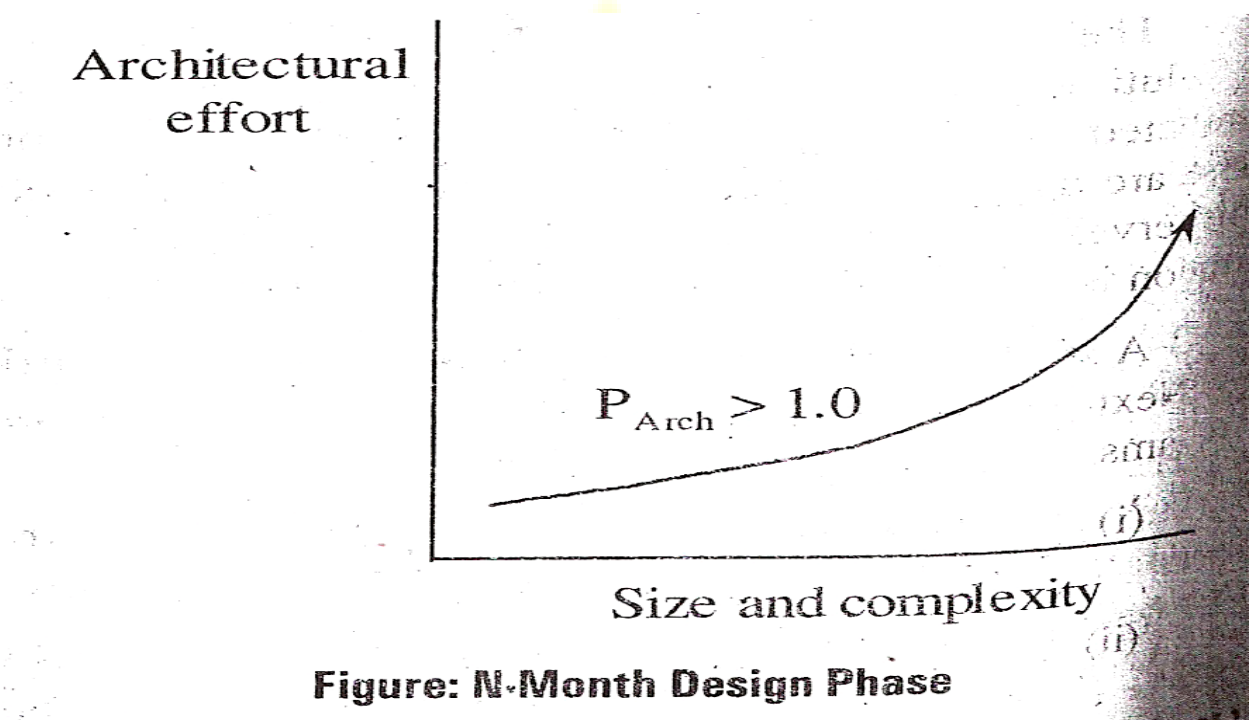

Figure: N-Month Design Phase

- At Application production stage
  - A plan with high fidelity and lower risk
  - It is cost-based
  - It has fixed-priced contracts
  - Team size is large and diverse as needed.

- Architecture team consists of a small number of software engineers.
- The Application team may have nay number of domain engineers.
- The output will be a function which is deliverable and useful, tested

baseline and warranted quality.

- The focus of the application production will be on implementing testing

and maintaining target technology.

- It contains two phases they are construction and transition

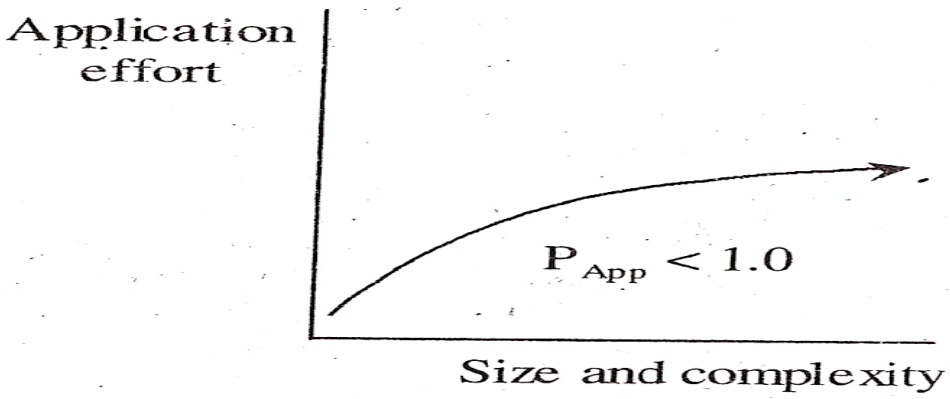

Figure: M-Month Production Increments

Total Effort = Func(TechnologyArch, ScaleArch, Quality Arch, Process Arch) + Func(TechnologyApp, ScaleApp, Quality App, Process App)

Total Time = Func(ProcessArch, EffortArch) + Func(ProcessApp, EffortApp,)

- The next generation infrastructure and environment automated various management activities with low effort. It relieves many of the sources of diseconomy of scale by reusing the common processes that are repetitive in a particular project. It also reuses the common outcomes of the project. The prior experience and matured processes utilized in these types of models eliminate the scrap rework sources. Here, the economics of scale will be affected.

- The architecture and applications of next generation cost models have difference scales and sized which represents the solution space. The size can be computed inters of SLOC or megabytes of executable code while the scale can be computed in 0-terms of components, classes, processes or nodes. The requirement or use cases of solution space are different from that of a problem space. Moreover, there can be more than one solution to a problem. Where cost serves as a key discriminator. The cost estimates must be determined to find an optimal solution. If an optional solution is not found then different solution s need to be selected or to change the problem statement**.**
- A strict notation must be applied for design artifacts so, that the prediction of a design scale can be improved. The Next-generation software cost model should automate the process of measuring design scale directly from UML diagrams. There should be two major improvements. There are,
    - Separate architectural engineering stage from application production stage. This will yield greater accuracy and more precision of lifecycle estimate.
    - The use of rigorous design notations. This will enable the automation and standardization of scale measure so that they can be easily traced which helps to determine the total cost associated with production.
- The next generation software process has two potential breakthroughs, they are,
    - Certain integrated tools would be available that automates the information transition between the requirements, design, implementation and deployment elements. These tools facilitate roundtrip engineering between various artifacts of engineering.

It will reduce the four sets of fundamental technical artifacts into three sets. This is achieved by automating the activities related to human-generated source code so as to eliminate the need fro a separate implementation set

An organizational manager should strive for making the transition to a modern process'.

- The transition to a modern process should be made based on the following quotations laid by Boehm.

Identifying and solving a software problem in the design phase is almost 100 times cost effective than solving the same problem after delivery.

This quotation or metric serves as a base for most software processes. Modern processes, component-based development techniques and architectural frameworks mainly focuses on enhancing this relationship. The architectural errors are solved by implementing an architecture-first approach. Modern process plays a crucial role in identification of risks

Software Development schedules can be compressed to a Maximum of 25 percent

If we want a reduction in the scheduled time, then we must increase the personnel resources which inturn increases the management overhead. The management overhead, concurrent activities scheduling, sequential activities conservation along some resource constraints will have the flexibility limit of about 25 percent.

This metric must be acceptable by the engineering phase which consists of detailed system content if we have successfully completed the engineering then compression in the production stage will be automatically flexible. The concurrent development must be possible irrespective of whether a business organization implements the engineering phase over multiple projects or whether a project implements the engineering phase over multiple incremental stages

The maintenance cost will be almost double the development cost

Most o the experts in the software industry find it difficult to maintain the software than development. The ratio between development and maintenance can be measured by computing productivity cost. One of the interesting fact of iterative development is that the dividing line between the development and maintenance is vanishing. Moreover, a good iterative process and an architecture will cause the reduction in the scrap and rework levels so this ratio (i.e.,) 2:1 can be reduced to 1:1.

Both the software development cot and the maintenance cost are dependent on the number of lines in the source code.

This metric was applicable to the conventional cost models which were lacking in-terms of commercial components, reusing techniques, automated code generators etc. The implementation of commercial components, reusing techniques and automated code generators will make this metric inappropriate. However, the development cost is still dependent on the commercial components, reuse technique and automatic code generators and their integration.

The next-generation cost models should focus more on the number of components and their integration efforts rather than on the number of lines of code.

Software productivity mainly relies on the type of people employed

The personal skills, team work ability and the motivation of employees are the crucial factors responsible for the success and the failure of any project. The next-generation cost models failure should concentrate more on employing a highly skilled team of professionals at engineering stage

The ratio of software to hardware cost is increasing.

As the computers are becoming more and more popular, the need for software an hardware applications is also increasing. The hardware components are becoming cheaper whereas, the software applications are becoming more complicated as a result, highly skilled professionals needed for development and controlling the software applications, the in turn increases the cost. In 1955 the software to hardware cost ratio was 15:85 and in 1985 this ratio was 85:15. This ratio continuously increases with respect to the need for variety of software applications. Certain software applications have already been developed which provides automated configuration control and analysis of quality assurance. The next-generation cost models must focus on automation of production and testing.

Only 15% of the overall software development is dedicated process to programming.

- The automation and reusability of codes have lead to the reduction in programming effort. Earlier in 1960s, the programming staff was producing about 200 machine instructions per month and in 1970s and 1980s, the machine instruction count has raised to about 1000

machine instructions. Now as days, programmers are able to produce several thousand instructions without even writing few hundreds of them

Software system and products cost three times the cost associated with individual software programs per SLOC software-system products cost 9 times more than the cost of individual software program.

- In the software development, the cost of each instruction depends upon the complexity of the software. Modern processes and technologies must reduce this diseconomy of scale. The economy of the scale must be achievable under the customer specific software systems with a common architecture, common environment and common process.

60% of Errors are caught by walkthrough

- The walkthrough and other forms of human inspection catch only the surface and style issues. However, the critical issues are not caught by the walkthroughs so, this metric doesn't prove to the reliable.

Only 20% of the contributors are responsible for the 80% of the contributions.

- This metric is applicable to most of the engineering concepts such as 80:20 principles of software project management. The next generation software process must facilitate the software organizations in achieving economic scale.

MODERN PROCESS TRANSITIONS

Indications of a successful project transition to a modern culture

- Several indicators are available that can be observed in order to distinguish projects that have made a genuine cultural transition from projects that only pretends.
- The following are some rough indicators available.

The lower-level managers and the middle level managers should participate in the project development

Any organization which ha an employee count less than or equal to 25 does not need to have pure managers. The responsibility of the managers in this type of organization will be similar to that of a project manager. Pure managers are needed when personal resources exceed 25. Firstly, these managers understand the status of the project them, develop the plans and estimate the results. The manager should participate in developing the plans. This transition affects the software project managers

Tangible design and requirements

The traditional processes utilize tons of paper in order to generate the documents relevant to the desired project. Even the significant milestones of a project are expressed via documents. Thus, the traditional process spends most of their crucial time on document preparation instead of performing software development activities.

An iterative process involves the construction of systems that describe the architecture, negotiates the significant requirements, identifies and resolves the risks etc. These milestones will be focused by all the stakeholders because they show progressive deliveries of important functionalities instead of documental descriptions about the project. Engineering teams will accept this transition of environment from to less document-driven while conventional monitors will refuse this transition.

Assertive Demonstrations are prioritized

The design errors are exposed by carrying-out demonstrations in the early stages of the life cycle. The stake holders should not over-react to these design errors because overemphasis of design errors will discourage the development organizations in producing the ambitious future iterating. This does not mean that stakeholders should bare all these errors. Infact, the stakeholders must follow all the significant steps needed for resolving these issues because these errors will sometimes lead to serious down-fall in the project.

This transition will unmark all the engineering or process issues so, it is mostly refused by management team, and widely accepted by users, customers and the engineering team.

The performance of the project can be determined earlier in the life cycle.

The success and failure of any project depends on the planning and architectural phases of life cycle so, these phases must employ high-skilled professionals. However, the remaining phases may work well an average team.

Earlier increments will be adolescent

The development organizations must ensure that customers and users should not expect to have good or reliable deliveries at the initial stages. This can be done by demonstration of flexible benefits in successive increments. The demonstration is similar to that of documentation but involves measuring of changes, fixes and upgrades based on the objectives so as to highlight the process quality and future environments

Artifacts tend to be insignificant at the early stages but proves to be the most significant in the later stages : The details of the artifacts should not be considered unless a stable and a useful baseline is obtained. This transition is accepted by the development team while the conventional contract monitors refuse this transition.

Identifying and Resolving of real issues is done in a systematic order

The requirements and designs of any successful project arguments along with the continuous negotiations and trade-offs. The difference between real and apparent issued of a successful project can easily be determined. This transition may affect any team of stakeholders

Everyone should focus on quality assurance

The software project manager should ensure that quality assurance is integrated in every aspect of project that is it should be integrated into every individuals role, every artifact, and every activity performed etc. There are some organizations which maintains a separate group of individuals know as quality assurance team, this team would perform inspections, meeting and checklist inorder to measure quality assurance. However, this transition involves replacing of separate quality assurance team into an organizational teamwork with mature process, common objectives and common incentives. So, this transition is supported by engineering teams and avoided by quality assurance team and conventional managers.

Performance issues crop up earlier in the projects life cycle

Earlier performance issues are a mature design process but resembles as an immature design. This transition is accepted by development engineers because it enables the evaluation of performance tradeoffs in subsequent releases.

Automation must be done with appropriate investments

Automation is the key concept of iterative development projects and must be done with sufficient funds. Moreover, the stakeholders must select an environment that supports iterative development. This transition is mainly opposed by organizational managers.

Good software organizations should have good profit margins.

Most of the contractors for any software contracting firm focus only on obtaining their profit margins beyond the acceptable range of 5% and 15%. They don't look for the quality of finished product as a result, the customers will be affected. For the success of any software industry, the good quality and at a reasonable rate them, customer will not worry about the profit the contractor has made. The bad contractors especially in a government contracting firm will be against this transition

Characteristics of conventional and iterative software development Process

- The characteristics of the conventional software process are listed below:
    1. It evolves in the sequential order (requirement design-code-test).
    2. It gives the same preference to all the artifacts, components, requirements etc.
    3. It completes all the artifacts of a stage before moving to the other stage in the project life cycle.
    4. It achieves traceability with high-fidelity for al the artifacts present at each life cycle stage.
- The characteristics of the modern iterative development process framework are listed below:

1. It continuously performs round-trip engineering of requirements, design, coding and testing at evolving levels of abstraction.
2. It evolves the artifacts depending on the priorities of the risk management.
3. It postpones the consistency analysis and completeness of the artifacts to the later stages in the life cycle.
4. It achieves the significant drives (i.e. 20 percent) with high-fidelity during the initial stages of the life cycle.