UNIT-IV

Project Organizations

Line-of- business organizations, project organizations, evolution of organizations, process automation.

Project Control and process instrumentation

The seven core metrics, management indicators, quality indicators, life-cycle expectations, Pragmatic software metrics, and metrics automation.

Part-1 Project Organizations and Responsibilities

Project Organizations and Responsibilities

 Organizations engaged in software Line-of-Business need to support projects with the infrastructure necessary to use a common process.

 Project organizations need to allocate artifacts & responsibilities across project team to ensure a balance of global (architecture) & local (component) concerns.

 The organization must evolve with the WBS & Life cycle concerns. **Software lines of business &**
 product teams have different motivation. Software lines of business are motivated by return of
 investment (ROI),
new business discriminators, market diversification &profitability.

 Project teams are motivated by the cost, Schedule &quality of specific deliverables

1) Line-Of-Business Organizations:

The main features of default organization are as follows:

• Responsibility for process definition & maintenance is specific to a cohesive line of business.

• Responsibility for process automation is an organizational role & isequal in importance to the process definition role.

• Organizational role may be fulfilled by a single individual or several different teams. Fig: Default roles in a software Line-of-Business Organization. **Software Engineering Process Authority (SEPA)**

The SEPA facilities the exchange of information & process guidance both to & from project practitioners This role is accountable to General Manager for maintaining a current assessment of the organization's process maturity & its plan for future improvement

Project Review Authority (PRA)

The PRA is the single individual responsible for ensuring that a software project complies with all organizational & business unit software policies, practices & standards A software Project Manager is responsible for meeting the requirements of a contract or some other project compliance standard **Software Engineering Environment Authority(SEEA)**

The SEEA is responsible for automating the organization's process, maintaining the organization's standard environment, Training projects to use the Environment &maintaining organization-wide reusable assets The SEEA role is necessary to achieve a significant ROI for common process.

Infrastructure

An organization's infrastructure provides human resources support, project- independent research & development, &other capital software engineering assets.

2) Project organizations:

Artifacts Activities

- **Business case Customer interface, PRA interface**
- **Software development plan Planning, monitoring**
- **Status assessments Risk management**
- **Software process definition**
- **Process improvement**

Figure 11-2. Default project organization and responsibilities

Software Management

Software Development Software Architecture Software Assessment System engineering Administration

• The above figure shows a default project organization and maps project-level roles and responsibilities.

• The main features of the default organization are as follows:

• **The project management team** is an active participant, responsible for producing as well as managing.

• **The architecture team** is responsible for real artifacts and for the integration of components, not just for staff functions.

• **The development team** owns the component construction and maintenance activities.

• The assessment team is separate from development.

- **Quality** is everyone's into all activities and checkpoints.
- Each team takes responsibility for a different quality perspective.

3) EVOLUTION OF ORGANIZATIONS:

Transition Construction

Inception:

Software management: **50%**
Software Architecture: 20%
Software development: 20% Software
Assessment
(measurement/evaluation):10%

Elaboration:

Software management: 10%
Software Architecture: **50%**
Software development: 20% Software
Assessment
(measurement/evaluation):20%

Construction:

Software management: 10%
Software Architecture: 10%
Software development: **50%** Software
Assessment
(measurement/evaluation):30%

Transition:

Software management: 10%
Software Architecture: 5%
Software development: 35% Software
Assessment
(measurement/evaluation):**50% The**
Process Automation

Introductory Remarks:

The environment must be the first-class artifact of the process.
Process automation& change management is critical to an iterative process. If the change is expensive then the development organization will resist it.
Round-trip engineering& integrated environments promote change freedom & effective evolution of technical artifacts.
Metric automation is crucial to effective project control.
External stakeholders need access to environment resources to improve interaction with the development team & add value to the process.
The three levels of process which requires a certain degree of process automation for the corresponding process to be carried out efficiently.

MANAGEMENT INDICATORS:

1. There are three fundamental sets of management metrics: technical progress, financial status, and staffing progress
2. By examining these perspectives, management can generally assess whether a project is on budget and on schedule
3. Most managers know their resource expenditures in terms of costs and schedule. The problem is to assess how much technical progress has been made.
4. Following three are management indicators:

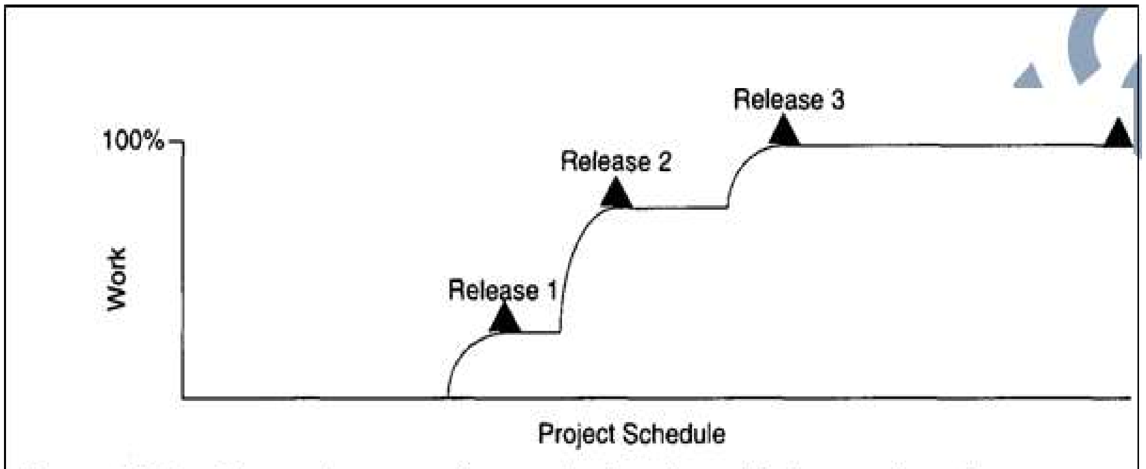


FIGURE 13-1. *Expected progress for a typical project with three major releases*

STAFFING A N D TEAM DYNAMICS:

1. Tracking actual versus planned staffing is a necessary is well-understood management metric.
2. There is one other important management indicator, the relationship between attrition and additions.
3. Increases in new staff can slow overall project progress as new people consume the more productive time.
4. Low attrition of good people is a sign of success.
5. If progress motivation is not there, good engineers will migrate elsewhere.
6. An increase in unplanned attrition-namely, people leaving a project prematurely-is one of the most obvious indicators that a project is destined for trouble.
7. The causes of such attrition can vary, but they are usually personnel dissatisfaction with management methods, lack of teamwork, or probability of failure in meeting the planned objectives.

BREAKAGE AND MODULARITY:

Breakage is defined as the average extent of change, which is the amount of software that needs rework.

1. Modularity is the average breakage trend over time.
2. For a healthy project, the trend expectation is decreasing or stable. (Figure 13
3. This indicator provides insight into the benign (not harmful) or malignant (harmful) character of

   software change.
4. In a mature iterative development process, earlier changes are expected to result in more scrap than later changes.
5. Breakage trends that are increasing with time clearly indicate that product maintainability is

REWORK AND

1. Rework is defined as the average c o s t of change.
2. Which is the effort t o analyze, resolve, and retest all changes to software b a s e l i n e s .
3. Adaptability is defined as the rework trend over time.
4. For a healthy project, the trend expectation is decreasing or stable.
5. Not all changes are treated equal. Some changes can be made in a staff-hour, while others take staff-weeks.
6. This metric provides insight into rework measurement.
7. In a mature i t e r a t i v e d e v e l o p m e n t process, e a r l i e r c h a n g e s a r e expected to require more rework than later changes.
8. Rework trends that are increasing with time clearly indicate that product maintainability is difficult.

1. **Software errors can be categorized into two types**: <u>deterministic and nondeterministic.</u> Physicists would characterize these as Bohr-bugs and
Heisen-bugs, r e s p e c t i v e l y .
2. Bohr-bugs represent a class of errors th at always result when the software i s used in a same way. These errors are predominantly caused by coding errors, and changes are typically isolated to a single component.
1. Heisen-bugs a r e software faults that are coincidental situation. These errors are almost always design errors and typically are not repeatable even when the software is used in the same apparent way.
2. Conventional software programs executing a single program on a single processor typically contained only Bohr-bugs.

3. Modern, distributed systems with numerous interoperating components executing across a network of processors are vulnerable to Heisen-bugs, which are far more complicated to detect, analyze, and resolve.
4. The best way to mature a software product is to establish an initial test infrastructure that allows execution of test early in the life cycle.

Write note on lifecycle expectations.(13.3 table)

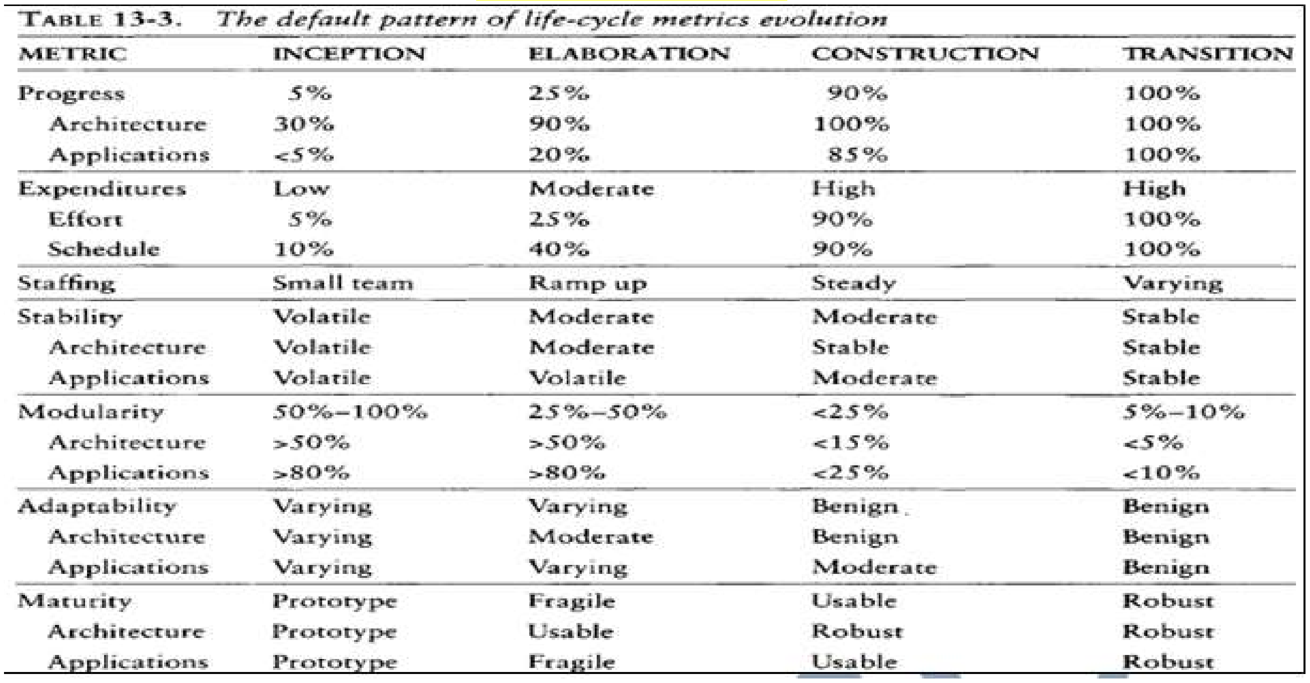


TABLE 13-3. *The default pattern of life-cycle metrics evolution*

METRIC	INCEPTION	ELABORATION	CONSTRUCTION	TRANSITION
Progress	5%	25%	90%	100%
Architecture	30%	90%	100%	100%
Applications	<5%	20%	85%	100%
Expenditures	Low	Moderate	High	High
Effort	5%	25%	90%	100%
Schedule	10%	40%	90%	100%
Staffing	Small team	Ramp up	Steady	Varying
Stability	Volatile	Moderate	Moderate	Stable
Architecture	Volatile	Moderate	Stable	Stable
Applications	Volatile	Volatile	Moderate	Stable
Modularity	50%–100%	25%–50%	<25%	5%–10%
Architecture	>50%	>50%	<15%	<5%
Applications	>80%	>80%	<25%	<10%
Adaptability	Varying	Varying	Benign	Benign
Architecture	Varying	Moderate	Benign	Benign
Applications	Varying	Varying	Moderate	Benign
Maturity	Prototype	Fragile	Usable	Robust
Architecture	Prototype	Usable	Robust	Robust
Applications	Prototype	Fragile	Usable	Robust

There is no mathematical evidence for using the seven core metrics. However, there were specific reasons for selecting them:

• The quality indicators are derived from the activity of ongoing process.
• They provide insight into the waste generated by the process. Scrap and rework metrics of manufacturing processes.

• Rather than focus on the value, they explicitly concentrate on the trends or changes with respect to time.
• The combination of insight from the current value and the current trend provides sufficient information for management.

Write note on pragmatic software metrics

Basic characteristics of good metric are as follows:

1. It is considered meaningful by the customer, manager, and performer. If any one of these stakeholders does not see the metric as meaningful, it will not be used.

2. It demonstrates quantifiable correlation between process perturbations (problems) and business performance.
3. It is objective and unambiguously defined. Objectivity should translate into some form of numeric representation (such as numbers, percentages, ratios) as opposed to textual representations (such as excellent, good, fair, poor).
Ambiguity is minimized through well-understood units of measurement (such as staff-month, SLOC, change, function point, class, scenario, and requirement),

4. It displays trends. This is an important characteristic. Understanding the change in a metric's value with respect to time, subsequent (later) projects, subsequent releases, and so forth is an extremely important perspective.

. 5. It is a natural by-product of the process. The metric does not introduce new artifacts or overhead activities; it is derived directly from the mainstream engineering and management workflows.
6. It is supported by automation.

Experience has demonstrated that the most successful metrics are those that are collected and reported b y automated tools.

Following examples shows why the values of metrics should be interpreted correctly.

- A low number of change requests to a software baseline may mean that the software is mature and error-free, or it may mean that the test team is on vacation.
- A software change order that has been open for a long time may mean that the problem was simple to diagnose and the solution required large rework, or it may mean that a problem was very time-consuming to diagnose and the solution required a simple change to a single line of code.
- A large increase in personnel in a given month may cause progress to increase proportionally if they are trained people who are productive from the outset. It may cause progress to down if they are untrained new hires who demand extensive support from productive people to get up to speed.

1. There are many opportunities to automate the project control activities of a software project.
2. To analyze the data, software project control panel (SPCP) that maintains an on-line version of the status of s/w provides the key advantage.
3. This concept was first recommended by the Airlie Software Council [Brown,1996], using the "dashboard".(displaying metrics /score)
4. The idea is to provide a display panel that integrates d a t a from multiple sources to show the current s tatu s of the project.
5. For example, the software project manager would want to see a display with overall project values, a test manager may want to see a display focused on metrics specific to an upcoming beta release, and development managers may be interested only in data concerning the subsystems and components for which they are responsible.
6. The panel can support standard features such as warning lights, thresholds, variable scales, digital formats, and analog formats to present an overview of the current situation.
7. This automation support can improve m a n a g e m e n t insight i n t o p r o g r e s s a n d quality trends and improve the acceptance of metrics by the engineering team.

To implement a complete SPCP, it is necessary to define and develop the following

- Metrics primitives: indicators, trends, comparisons, and progressions
- A graphical u s e r interface: GUI support for a software p r o j e c t m a n a g e r role and flexibility to support other roles.
- Metrics collection agents: data extraction from the environment tools.
- Metrics data management server: stores the data.
- metrics definitions: actual metrics presentations for requirements progress (extracted from requirements set artifacts), design progress (extracted from design set artifacts), implementation progress (extracted from implementation set artifacts), assessment progress (extracted from deployment set artifacts), and other progress dimensions (extracted from manual sources, financial management systems, management artifacts, etc.)
- Actors: typically, the monitor and the administrator.

What are four graphical objects of top level display

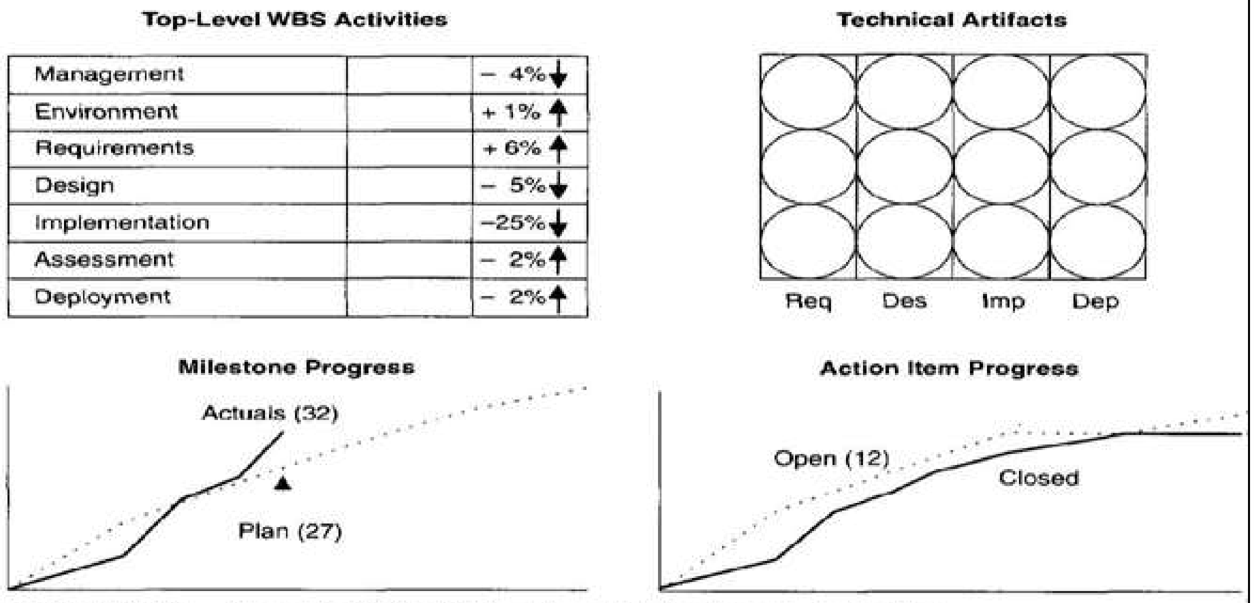


FIGURE 13-10. *Example SPCP display for a top-level project situation*

1. Project activity status. The graphical o b j e c t in the upper left provides an overview of the status of the top-level WBS elements. The seven elements could be coded red, yellow, and green to reflect the current e a r n e d v a l u e status. (In Figure 13-10, they are coded with white and shades of gray.) For example, green would represent ahead of plan, yellow would indicate within 10% of plan, and red would identify elements that have a greater than 10% cost or schedule variance.
2. Technical artifact status . The graphical ob j e ct in the upper right provides an overview of the status of technical artifacts. The Req light would display current state of requirements specifications. The Des light would do the same for the design models, the Imp light for the source code baseline, and the Dep light for the test program.
3. Milestone progress. The graphical object in the lower left provides a progress assessment of the achievement of milestones against plan.
4. Action item progress. The graphical object in the lower right provides a different perspective of progress, showing t h e current n u m b e r o f open and closed issues.

UNIT V

CCPDS-R Case Study and Future Software Project Management Practices

Modern Project Profiles, Next-Generation software Economics, Modern Process Transitions.

CCPDS-R Case Study and Future Software Project Management Practices

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 downstream integration, late patches and shoe-horned software fixes by implementing sequential or continuous integration rather than implementing large-scale integration during the project completion.

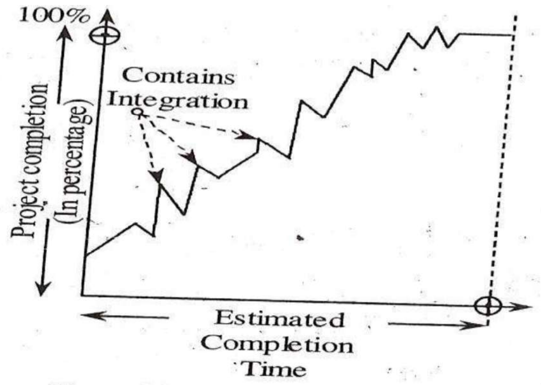


Figure (a): Modern Project Profile

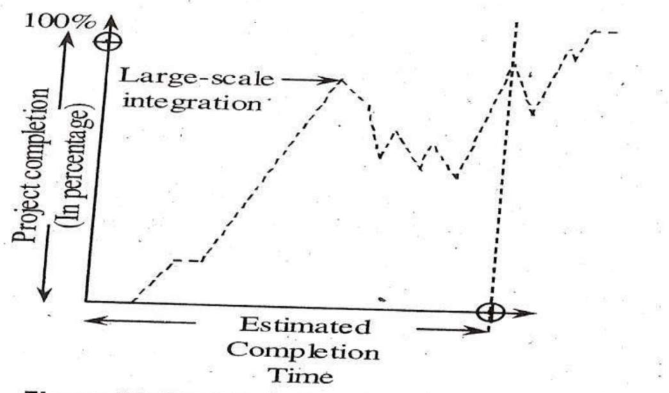


Figure (b): Traditional Project Profile

- 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 Ex**
Management	**5%**	**10%**
Environment	**5%**	**10%**
Requirements	**5%**	**10%**
Design	**10%**	**15%**
Implementation	**30%**	**25%**
Assessment	**40%**	**25%**
Deployment	**5%**	**5%**

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 requirements.
- Before 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.
- 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.

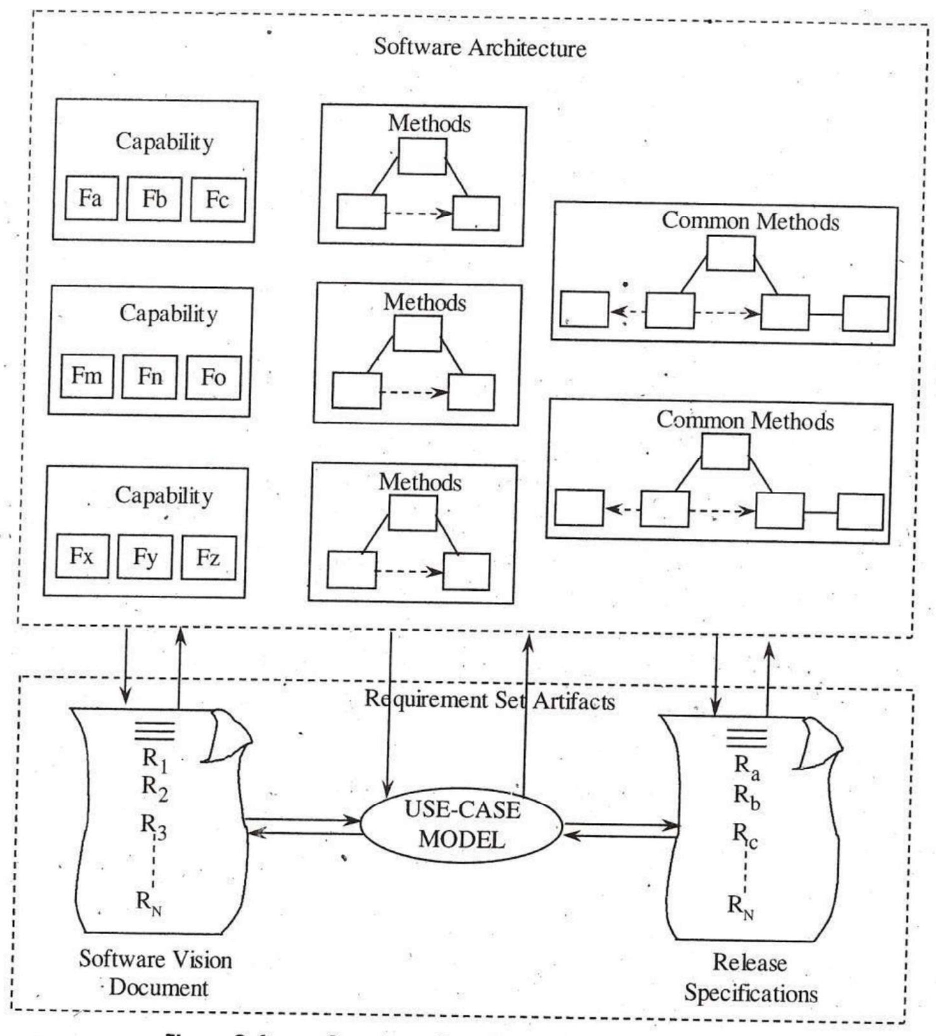


Figure: Software Component Organization of a Modern Process

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.

8.1.4 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 has 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.
- 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 tradeoffs 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.

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.

NEXT GENERATION SOFTWARE ECONOMICS

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.

- 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.

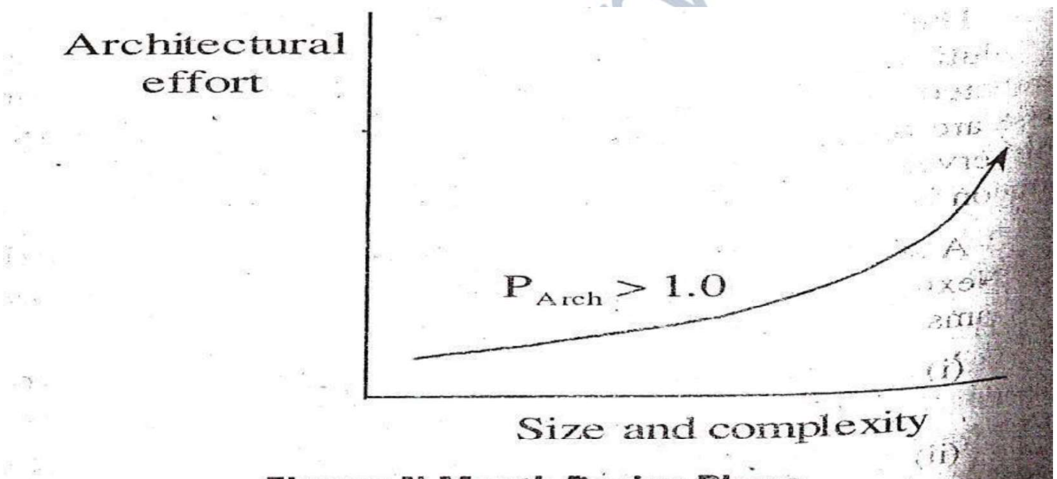


Figure: N-Month Design Phase

- 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.

3. 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 has 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 in order 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.

CASE STUDIES:

COCOMO MODEL

The best known and most transparent cost model COCOMO (Constructive costmodel) was developed by Boehm, which was derived from the analysis of 63 software projects. Boehm proposed three levels of the model: basic, intermediate and detailed. COCOMO focuses mainly upon the intermediate mode.
The COCOMO model is based on the relationships between:
Equation 1:- Development effort is related to system size

$$MM = a.KDSI.b$$

Equation 2:- Effort and development time

$$TDEV = c.MM.d$$

where MM is the effort in man-months.

KDSI is the number of thousand delivered source instructions.
TDEV is the development time.
The coefficients a, b, c and d are dependent upon the 'mode of development which Boehm classified into 3 distinct modes:

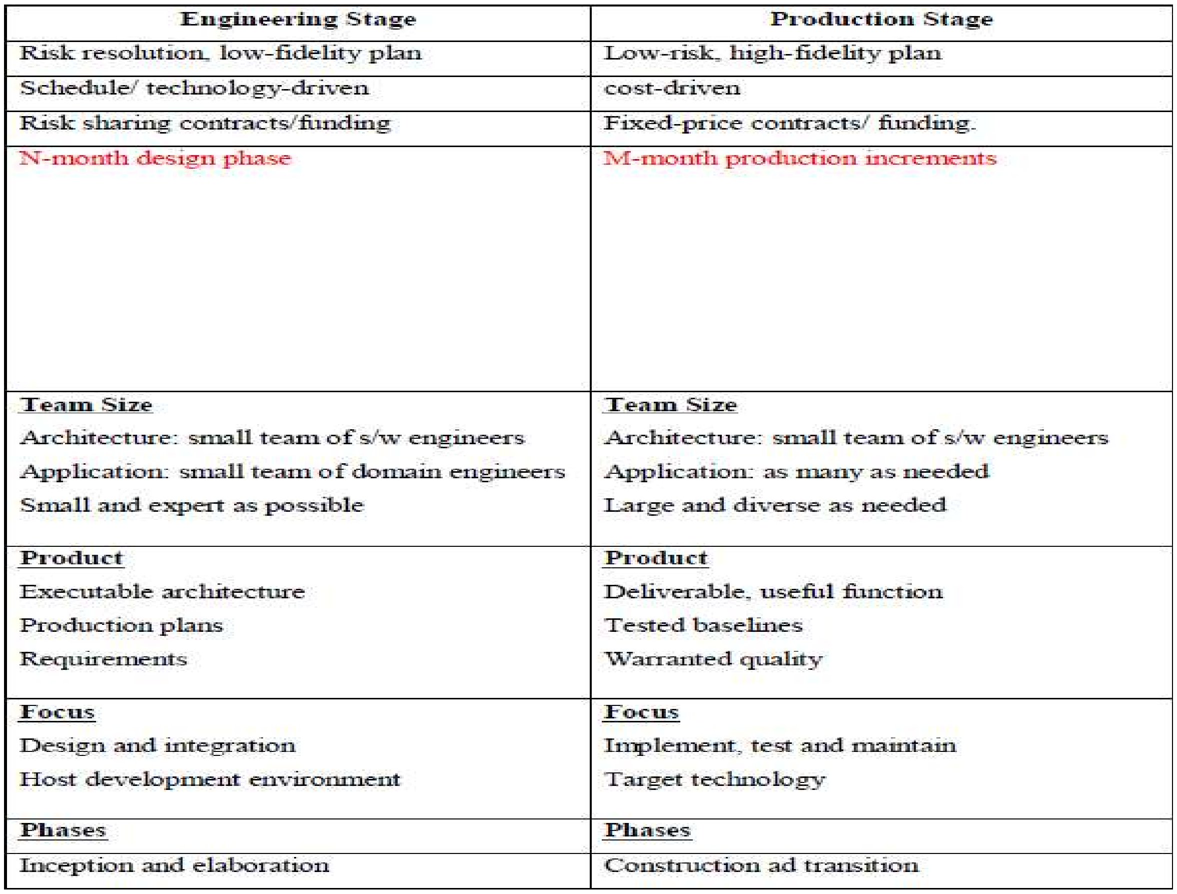
1. Organic - Projects involve small teams working in familiar and stable environments.
Eg: - Payroll systems.
2. Semi - Detached - Mixture of experience within project teams. This lies in between organic and embedded modes.
Eg: Interactive banking system.
3. Embedded: - Projects that are developed under tight constraints, innovative, complex and have volatility of requirements.
Eg: - nuclear reactor control systems.
Development mode A B C D
Organic 3.2 1.05 2.5 0.38

Semi-detached 3.0 1.12 2.5 0.35
Embedded 2.8 1.20 2.5 0.32

In the intermediate mode it is possible to adjust the nominal effort obtained from the model by the influence of 15 cost drivers. These drivers deviate from the nominal figures, where particular project differ from the average project. For example, if the reliability of the software is very high, a factor rating of 1.4 can be assigned to that driver. Once all the factors for each driver have been chosen they are multiplied to arrive at an Effort Adjustment Factor (EAF).
The actual steps in producing an estimate using the intermediate COCOMO model are:
1. Identify the 'mode' of development for the new project.
2. Estimate the size of the project in KDSI to derive a nominal effort prediction.
3. Adjust the 15 cost drivers to reflect your project, producing an error adjustment factor.
4. Calculate the predicted project effort using equation 1 and the effort adjustment factor.
5. Calculate the project duration using equation 2.

Drawbacks:
1. It is hard to accurately estimate KDSI early on in the project, when most effort estimates are required.
2. Extremely vulnerable to mis-classification of the development mode.
3. Success depends largely on tuning the model to the needs of the organization, using historical data which is not always available.

Advantages:
1. COCOMO is transparent. It can be seen how it works.
2. Drivers are particularly helpful to the operator to understand the impact of different factors that affect project costs.

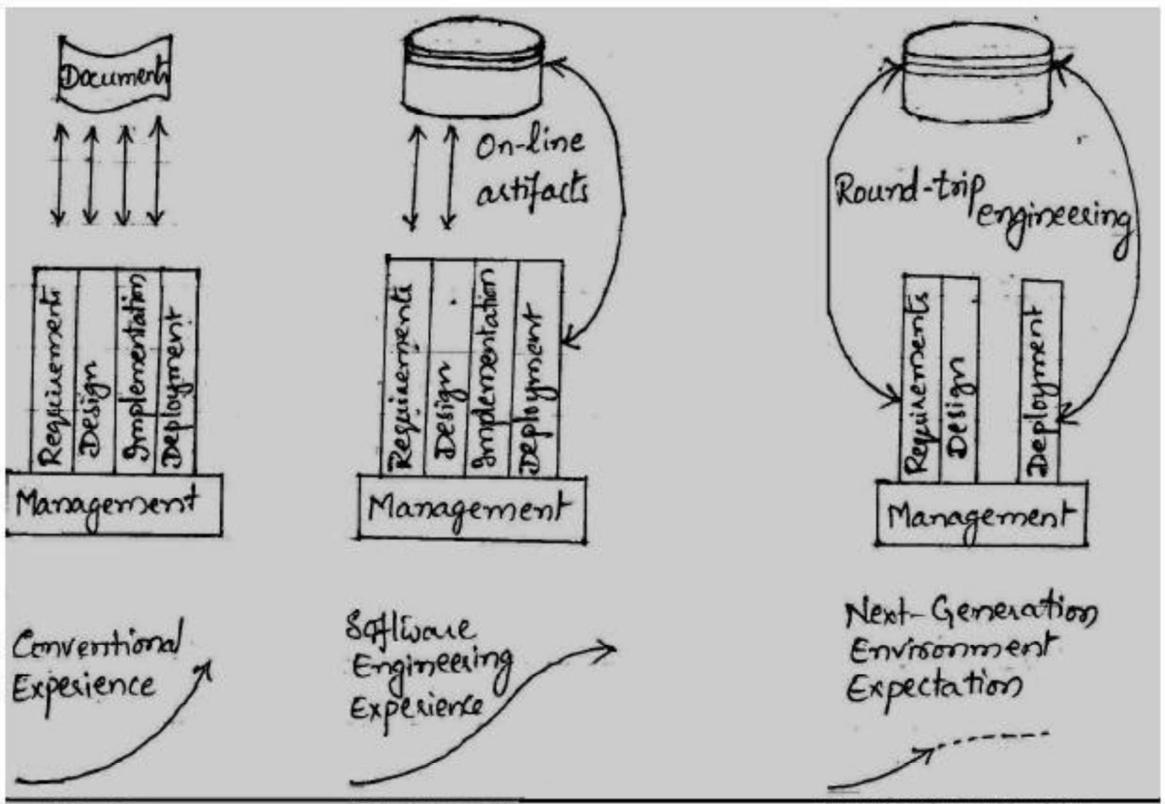


Engineering Stage	Production Stage
Risk resolution, low-fidelity plan	Low-risk, high-fidelity plan
Schedule/ technology-driven	cost-driven
Risk sharing contracts/funding	Fixed-price contracts/ funding.
N-month design phase	M-month production increments
Team Size Architecture: small team of s/w engineers Application: small team of domain engineers Small and expert as possible	**Team Size** Architecture: small team of s/w engineers Application: as many as needed Large and diverse as needed
Product Executable architecture Production plans Requirements	**Product** Deliverable, useful function Tested baselines Warranted quality
Focus Design and integration Host development environment	**Focus** Implement, test and maintain Target technology
Phases	**Phases**
Inception and elaboration	Construction ad transition

Phases Phases

Inception and elaboration Construction ad transitionArchitecture and applications have different units of mass-scale and size. Scale ismeasured in terms of architecturally significant elements such as classes, components, processes and nodes. Size is measured in SLOC or megabyte of executable code. Next generation environments and infrastructures are moving to automate and standardize many of the management activities, thereby requiring a lower percentage of effort for overhead activities as scale increases. The two major improvements in next-generation cost estimation models are.

1. Separation of the engineering stage from the production stage to differentiate between architectural scale and implementation size.
2. Rigorous design notations such as UML to be more standardized. The automation of the construction process in next-generation environments is shown below.



All Engineering

Engineering Seperate from Production

Engineering with automated production