ESPM UNIT-IV

Interactive Process Planning: Work Breakdown Structures, Planning Guidelines, Cost and Schedule Estimating. Interaction Planning Process. Pragmatic Planning.

Project Organizations and Responsibilities: Line-of-Business Organizations, Project Organizations, and Evolution of Organizations.

Process Automation: Automation Building Blocks, the Project Environment.

WORK BREAKDOWN STRUCTURES (WBS)

- A good work breakdown structure and its synchronization with the process framework are critical factors in software project success. Development of a work breakdown structure dependent on the project management style, organizational culture, customer preference, financial constraints and several other hard-to-define, project-specific parameters.
- A WBS is simply a hierarchy of elements that decomposes the project plan into the discrete work tasks.
- A WBS provides the following information structure:
  a) A delineation of all significant work
  b) A clear task decomposition for assignment of responsibilities
  c) A framework for scheduling, budgeting, and expenditure tracking

CONVENTIONAL WBS ISSUES

Conventional work breakdown structures frequently suffer from three fundamental flaws.

- They are prematurely structured around the product design.
- They are prematurely decomposed, planned, and budgeted in either too much or too little detail.
- They are project-specific, and cross-project comparisons are usually difficult or impossible.

```
Management
System requirements and design
Subsystem 1
    Component 11
        Requirements
        Design
        Code
        Test
        Documentation
    . . . (similar structures for other components)
    Component 1N
        Requirements
        Design
        Code
        Test
        Documentation
. . . (similar structures for other subsystems)
Subsystem M
    Component M1
        Requirements
        Design
        Code
        Test
        Documentation
    . . . (similar structures for other components)
    Component MN
        Requirements
        Design
        Code
        Test
        Documentation
Integration and test
    Test planning
    Test procedure preparation
    Testing
    Test reports
Other support areas
    Configuration control
    Quality assurance
    System administration
```

Conventional work breakdown structure, following the product hierarchy

- *Conventional work breakdown structures are prematurely structured around the product design.* The above Figure shows a typical conventional WBS that has been structured primarily around the subsystems of its

product architecture, and then further decomposed into the components of each subsystems. A WBS is the architecture for the financial plan.

- *Conventional work breakdown structures are prematurely decomposed, planned and budgeted in either too little or too much detail.* Large software projects tend to be over planned and small projects tend to be under planned. The basic problem with planning too much detail at the outset is that the detail does not evolve with the level of fidelity in the plan.
- *Conventional work breakdown structures are project-specific and cross-project comparisons are usually difficult or impossible*. With no standard WBS structure, it is extremely difficult to compare plans, financial data, schedule data, organizational efficiencies, cost trends, productivity trends, or quality trends across multiple projects.

EVOLUTIONARY WORK BREAKDOWN STRUCTURES

An evolutionary WBS should organize the planning elements around the process framework rather than the product framework. The basic recommendation for the WBS is to organize the hierarchy as follows:

a) First-level WBS elements are the workflows (management, environment, requirements, design, implementation, assessment, and deployment).

b) Second-level elements are defined for each phase of life cycle (inception, elaboration, construction, and transition).

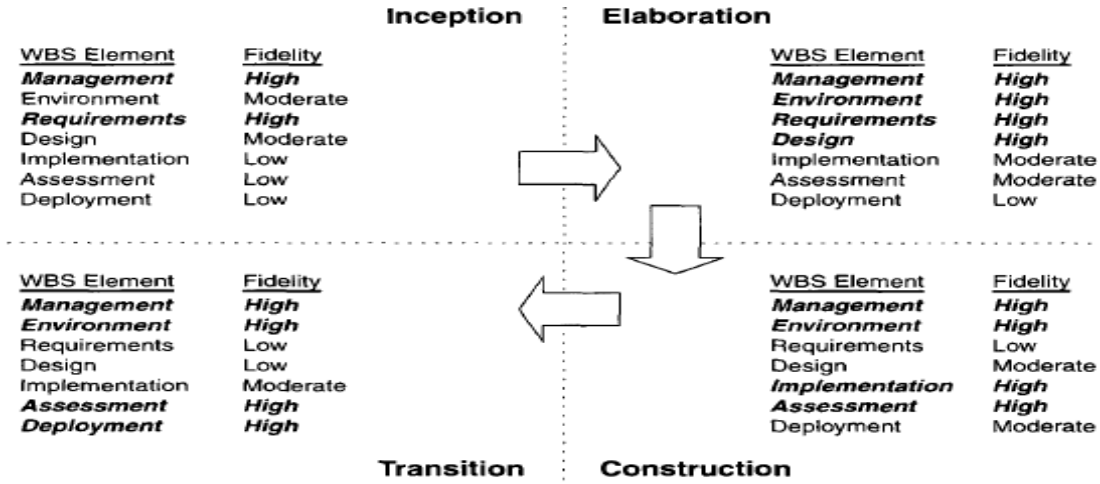
c) Third-level elements are defined for the focus of activities that produce the artifacts of each phase.

A default WBS consistent with the process framework (phases, workflows, and artifacts) is shown in Figure:

- A Management
  - AA Inception phase management
    - AAA Business case development
    - AAB Elaboration phase release specifications
    - AAC Elaboration phase WBS baselining
    - AAD Software development plan
    - AAE Inception phase project control and status assessments
  - AB Elaboration phase management
    - ABA Construction phase release specifications
    - ABB Construction phase WBS baselining
    - ABC Elaboration phase project control and status assessments
  - AC Construction phase management
    - ACA Deployment phase planning
    - ACB Deployment phase WBS baselining
    - ACC Construction phase project control and status assessments
  - AD Transition phase management
    - ADA Next generation planning
    - ADB Transition phase project control and status assessments
- B Environment
  - BA Inception phase environment specification
  - BB Elaboration phase environment baselining
    - BBA Development environment installation and administration
    - BBB Development environment integration and custom toolsmithing
    - BBC SCO database formulation
  - BC Construction phase environment maintenance
    - BCA Development environment installation and administration
    - BCB SCO database maintenance
  - BD Transition phase environment maintenance
    - BDA Development environment maintenance and administration
    - BDB SCO database maintenance
    - BDC Maintenance environment packaging and transition
- C Requirements
  - CA Inception phase requirements development
    - CAA Vision specification
    - CAB Use case modeling
  - CB Elaboration phase requirements baselining
    - CBA Vision baselining
    - CBB Use case model baselining
  - CC Construction phase requirements maintenance
  - CD Transition phase requirements maintenance
- D Design
  - DA Inception phase architecture prototyping
  - DB Elaboration phase architecture baselining
    - DBA Architecture design modeling
    - DBB Design demonstration planning and conduct
    - DBC Software architecture description
  - DC Construction phase design modeling
    - DCA Architecture design model maintenance
    - DCB Component design modeling
  - DD Transition phase design maintenance
- E Implementation
  - EA Inception phase component prototyping
  - EB Elaboration phase component implementation
    - EBA Critical component coding demonstration integration
  - EC Construction phase component implementation
    - ECA Initial release(s) component coding and stand-alone testing
    - ECB Alpha release component coding and stand-alone testing
    - ECC Beta release component coding and stand-alone testing
    - ECD Component maintenance
  - ED Transition phase component maintenance
- F Assessment
  - FA Inception phase assessment planning
  - FB Elaboration phase assessment
    - FBA Test modeling
    - FBB Architecture test scenario implementation
    - FBC Demonstration assessment and release descriptions
  - FC Construction phase assessment
    - FCA Initial release assessment and release description
    - FCB Alpha release assessment and release description
    - FCC Beta release assessment and release description
  - FD Transition phase assessment
    - FDA Product release assessment and release descriptions
- G Deployment
  - GA Inception phase deployment planning
  - GB Elaboration phase deployment planning
  - GC Construction phase deployment
    - GCA User manual baselining
  - GD Transition phase deployment
    - GDA Product transition to user

PLANNING GUIDELINES

Software projects span a broad range of application domains. It is valuable but risky to make specific planning recommendations independent of project context. Project-independent planning advice is also risky. There is the risk that the guidelines may be adopted blindly without being adapted to specific project circumstance. Two simple planning guidelines should be considered when a project plan is being initiated or assessed.



Evolution of planning fidelity in the WBS over the life cycle

The below table prescribes a default allocation of costs among the first-level WBS elements.

WBS budgeting defaults

FIRST-LEVEL WBS ELEMENT	DEFAULT BUDGET
Management	10%
Environment	10%
Requirements	10%
Design	15%
Implementation	25%
Assessment	25%
Deployment	5%
Total	100%

The below table prescribes allocation of effort and schedule across the lifecycle phases.

Default distributions of effort and schedule by phase

DOMAIN	INCEPTION	ELABORATION	CONSTRUCTION	TRANSITION
Effort	5%	20%	65%	10%
Schedule	10%	30%	50%	10%

THE COST AND SCHEDULE ESTIMATING PROCESS

Project plans need to be derived from two perspectives. The first is a ***forward-looking***, top-down approach. It starts with an understanding of the general requirements and constraints, derives a macro-level budget and schedule, then decomposes these elements into lower level budgets and intermediate milestones.

From this perspective, the following planning sequence would occur:

- The software project manager (and others) develops a characterization of the overall size, process, environment, people, and quality required for the project.
- A macro-level estimate of the total effort and schedule is developed using a software cost estimation model.
- The software project manager partitions the estimate for the effort into top-level WBS using guidelines.

At this point, subproject managers are given the responsibility for decomposing each of the WBS elements into lower levels using their top-level allocation, staffing profile, and major milestone dates as constraints.

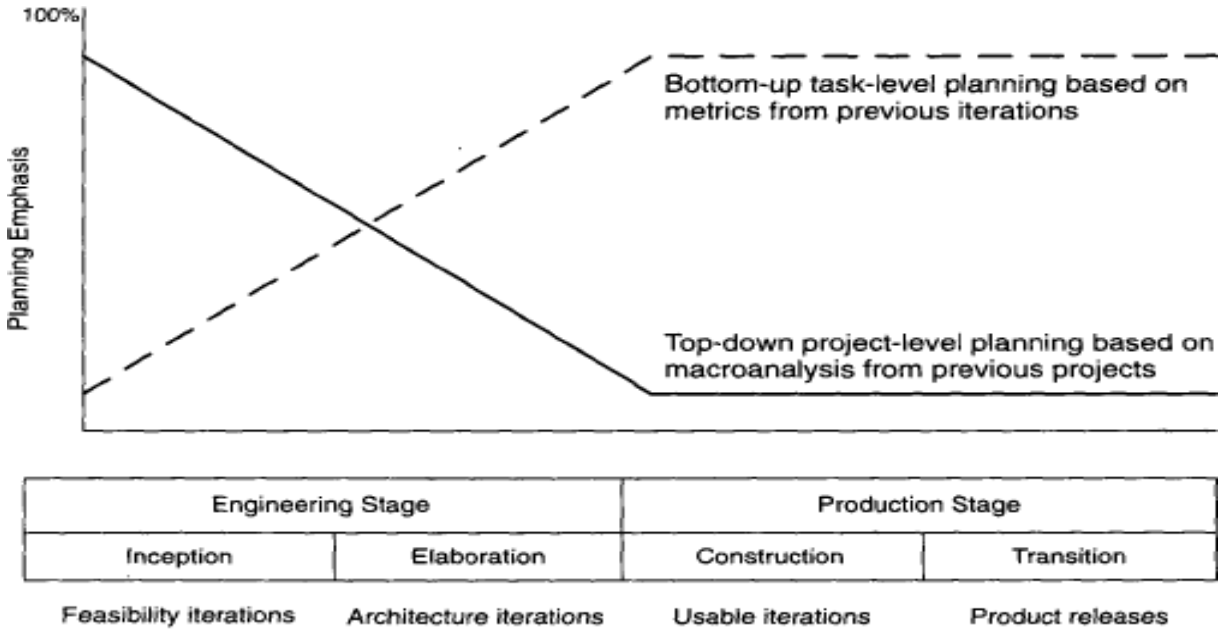
The second perspective is a ***backward-looking***, bottom-up approach. We start with the end in mind, analyze the micro-level budgets and schedules, and then sum all these elements into the higher level budgets and intermediate milestones. This approach tends to define and populate the WBS from the lowest levels upward. From this perspective, the following planning sequence would occur:

a) The lowest level WBS elements are elaborated into detailed tasks

b) Estimates are combined and integrated into higher level budgets and milestones.

c) Comparisons are made with the top-down budgets and schedule milestones.

During the engineering stage top down approach dominates bottom up approach. During the production stage bottom approach dominates top down approach.



Engineering stage planning emphasis:

- Macro-level task estimation for production-stage artifacts
- Micro-level task estimation for engineering artifacts
- Stakeholder concurrence
- Coarse-grained variance analysis of actual vs. planned expenditures
- Tuning the top-down project-independent planning guidelines into project-specific planning guidelines
- WBS definition and elaboration

Production stage planning emphasis:

- Micro-level task estimation for production-stage artifacts
- Macro-level task estimation for maintenance of engineering artifacts
- Stakeholder concurrence
- Fine-grained variance analysis of actual vs. planned expenditures

Planning balance throughout the life cycle

THE ITERATION PLANNING PROCESS

- Planning is concerned with defining the actual sequence of intermediate results.
- Iteration is used to mean a complete synchronization across the project, with a well-orchestrated global assessment of the entire project baseline.
  Inception Iterations: the early prototyping activities integrate the foundation components of candidate architecture and provide an executable framework for elaborating the critical use cases of eth system.
  Elaboration Iteration: These iterations result in architecture, including a complete framework and infrastructure for execution. Upon completion of the architecture iteration, a few critical use cases should be demonstrable: (1) initializing the architecture (2) injecting a scenario to drive the worst-case data processing flow through the system (3) injecting a scenario to drive the worst-case control flow through the system (for example, orchestrating the fault-tolerance use cases).
  Construction Iterations: Most projects require at least two major construction iterations: an alpha release and a beta release.
  Transition Iterations: Most projects use a single iteration to transition a beta release into the final product.
- The general guideline is that most projects will use between four and nine iteration. The typical project would have the following six-iteration profile:
  - **One iteration in inception:** an architecture prototype
  - **Two iterations in elaboration:** architecture prototype and architecture baseline
  - **Two iterations in construction:** alpha and beta releases
  - **One iteration in transition:** product release

PRAGMATIC PLANNING

Even though good planning is more dynamic in an iterative process, doing it accurately is far easier. While executing iteration N of any phase, the software project manager must be monitoring and controlling against a plan that was initiated in iteration N-1 and must be planning iteration N+1. the art of good project management is to make trade-offs in the current iteration plan and the next iteration plan based on objective results in the current iteration and previous iterations. Aside form bad architectures and misunderstood requirement, inadequate planning (and subsequent bad management) is one of the most common reasons for project failures. Conversely, the success of every successful project can be attributed in part to good planning.

A project's plan is a definition of how the project requirements will be transformed into a product within the business constraints. It must be realistic, it must be current, it must be a team product, it must be understood by the stake holders, and it must be used. Plans are not just for mangers. The more open and visible the planning process and results, the more ownership there is among the team members who need to execute it. Bad, closely held plans cause attrition. Good, open plans can shape cultures and encourage teamwork.

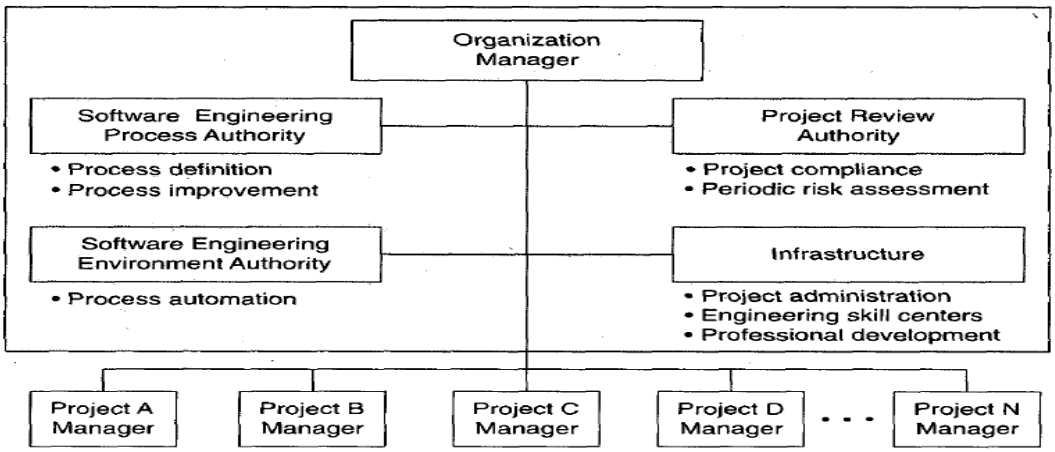
Software lines of business and project teams have different motivations. Software lines of business are motivated by return on investment, new business discriminators, market diversification and profitability.

Software professionals in both types of organizations are motivated by career growth, job satisfaction and the opportunity to make a difference.

LINE -OF-BUSINESS ORGANIZATIONS

The main features of the default organization are as follows:

- Responsibility for process definition and maintenance is specific to a cohesive line of business.
- Responsibility for process automation is an organizational role and is equal in importance to the process definition role.
- Organization roles may be fulfilled by a single individual or several different teams, depending on the scale of the organization.



Default roles in a software line-of-business organization

The line of business organization consists of 4 component teams.

Software Engineering Process Authority (SEPA):

- Responsible for exchanging the information and project guidance to or from the project practitioners.
- Maintains current assessment of organization process maturity.
- Help in initiate and periodically assess project processes.
- Responsible for process definition and maintainence.

Project Review Authority (PRA):

- Responsible for reviewing the financial performance, customer commitments, risks & accomplishments, adherence to organizational policies by customer etc.
- Reviews both project‟s conformance, customer commitments as well as organizational polices, deliverables, financial performances and other risks.

Software Engineering Environment Authority (SEEA):

- SEEA deals with the maintenance or organizations standard environment, training projects and process automation.
- Maintains organization‟s standard environment.
- Training projects to use environment.
- Maintain organization wide resources support.

Infrastructure:

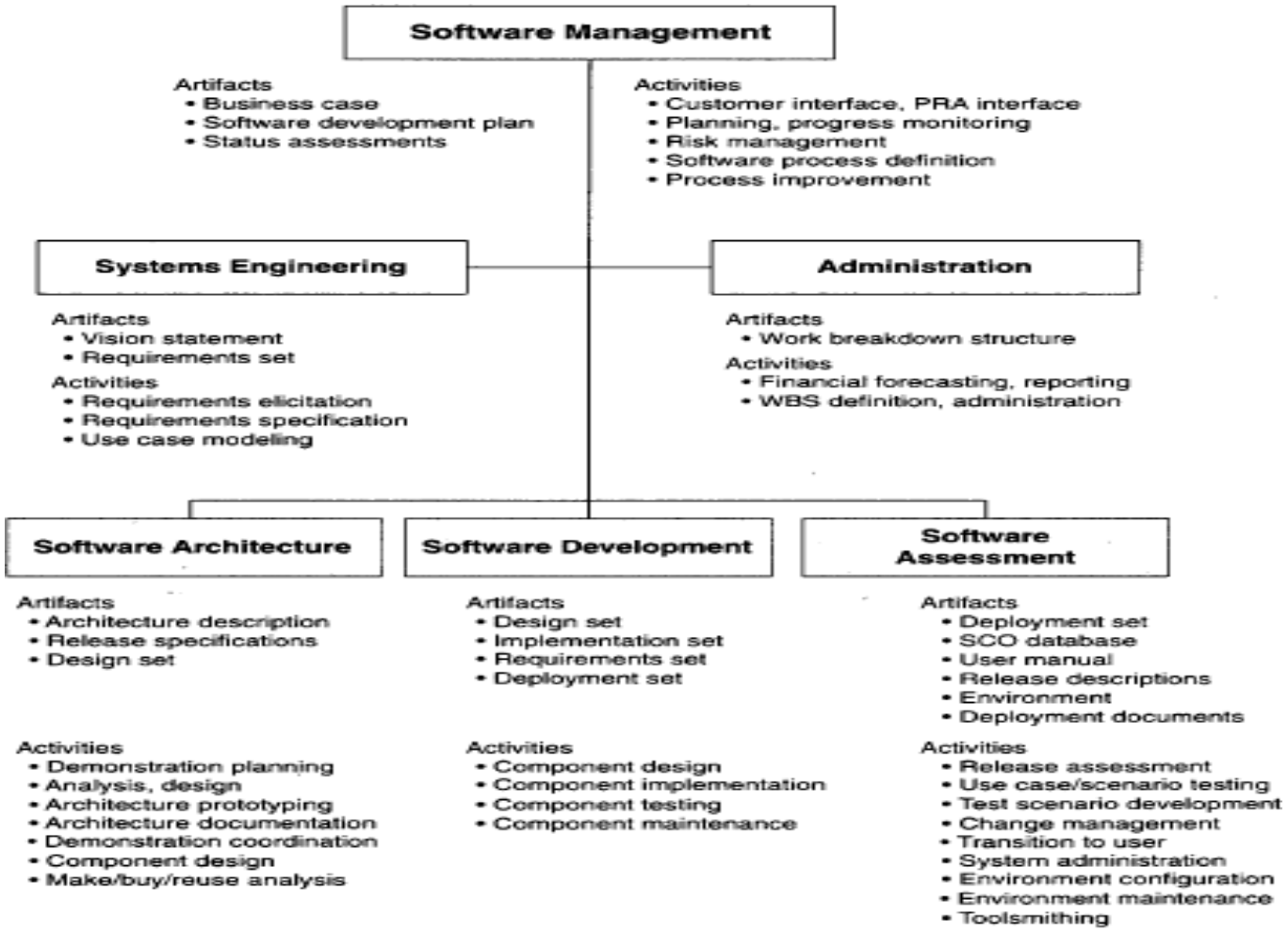
- An organization‟s infrastructure provides human resources support, project-independent research and development other capital software engineering assets. The typical components of the organizational infrastructure are as follows:
  - **Project Administration:** Time accounting system; contracts, pricing, 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.

PROJECT ORGANIZATIONS

The default project organization and maps project-level roles and responsibilities. This structure can be tailored to the size and circumstance of the specific project organization.

The main feature of the default organization is as follows:

- *The project management team* is an active participant, responsible for producing as well as managing. Project management is not a spectator sport.
- *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.
- *Quality is every one job. Each team takes responsibility for a different quality perspective.*



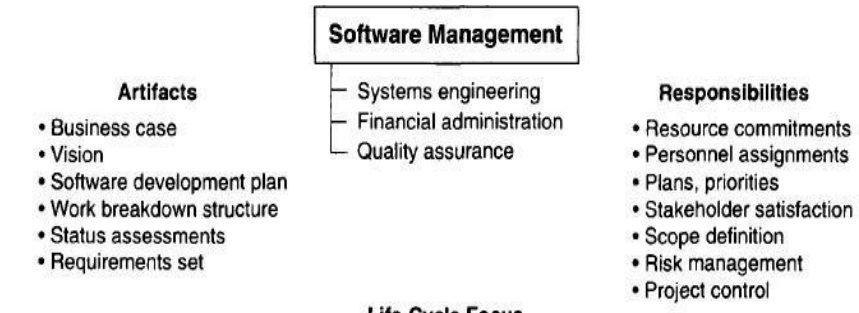
Default project organization and responsibilities

Software Management Team:

- This is active participant in an organization and is incharge of producing as well as managing.
- As the software attributes, such as Schedules, costs, functionality and quality are interrelated to each other, negotiation among multiple stakeholders is required and these are carried out by the software management team.

Responsibilities:

- Effort planning
- Conducting the plan
- Adapting the plan according to the changes in requirements and design
- Resource management
- Stakeholders satisfaction
- Risk management
- Assignment or personnel
- Project controls and scope definition
- Quality assurance



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

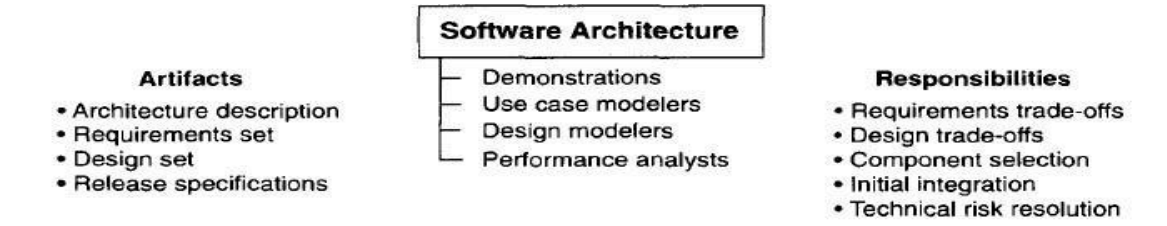
Software management team activities

Software Architecture Team:

- ❑ The software architecture team performs the tasks of integrating the components, creating real artifacts etc.
- ❑ It promotes team communications and implements the applications with a system-wide quality.
- ❑ The success of the development team is depends on the effectiveness of the architecture team along with the software management team controls the inception and elaboration phases of a life-cycle.
- ❑ The architecture team must have:
  - ❖ Domain experience to generate an acceptable design and use-caseview.
  - ❖ Software technology experience to generate an acceptable process view, component and development views.

Responsibilities:

- ❖ System-level quality i.e., performance, reliability and maintainability.
- ❖ Requirements and design trade-offs.
- ❖ Component selection
- ❖ Technical risk solution
- ❖ Initial integration



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 maintenance *Multiple-component issue resolution* Performance tuning Quality improvements

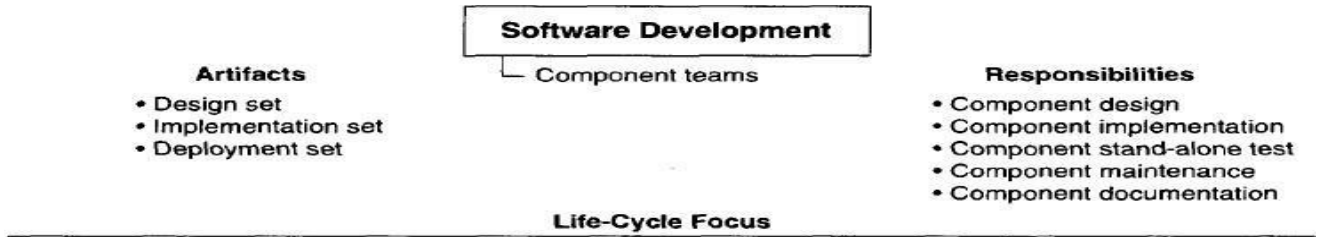
Software architecture team activities

Software Development Team:

- The Development team is involved in the construction and maintenance activities. It is most applicationspecific team. It consists of several sub teams assigned to the groups of components requiring a common skill set.
- The skill set include the following:
- ***Commercial component***: specialists with detailed knowledge of commercial components central to a system's architecture.
- ***Database***: specialists with experience in the organization, storage, and retrieval of data.
- ***Graphical user interfaces***: specialists with experience in the display organization; data presentation, and user interaction.
- ***Operating systems and networking***: specialists with experience in various control issues arises due to synchronization, resource sharing, reconfiguration, inter object communications, name space management etc.
- ***Domain applications***: Specialists with experience in the algorithms, application processing, or business rules specific to the system.

Responsibilities:

- The exposure of the quality issues that affect the customer‟s expectations.
- Metric analysis.
- Verifying the requirements.
- Independent testing.
- Configuration control and user development.
- Building project infrastructure.



Life-Cycle Focus

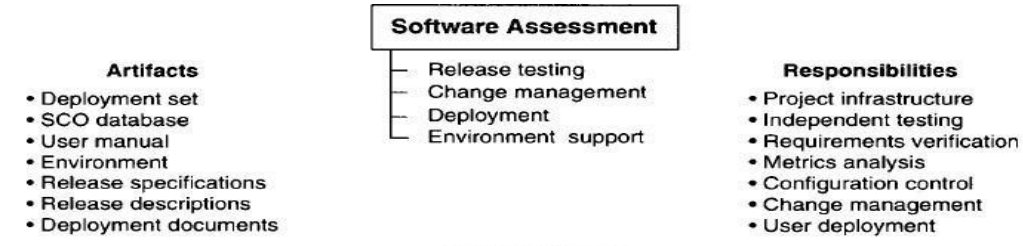
Inception	Elaboration	Construction	Transition
Prototyping support Make/buy trade-offs	Critical component design Critical component implementation and test Critical component baseline	Component design Component implementation Component stand-alone test Component maintenance	Component maintenance Component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

Software development team activities

Software Assessment Team:

- This team is involved in testing and product activities in parallel with the ongoingdevelopment.
- It is an independent team for utilizing the concurrency of activities.
- The use-case oriented and capability-based testing of a process is done by using two artifacts:
  - Release specification (the plan and evaluation criteria for a release)
  - Release description (the results of a release)

  Responsibilities:
  - The exposure of the quality issues that affect the customer‟s expectations.
  - Metric analysis.
  - Verifying the requirements.
  - Independent testing.
  - Configuration control and user development.
  - Building project infrastructure.



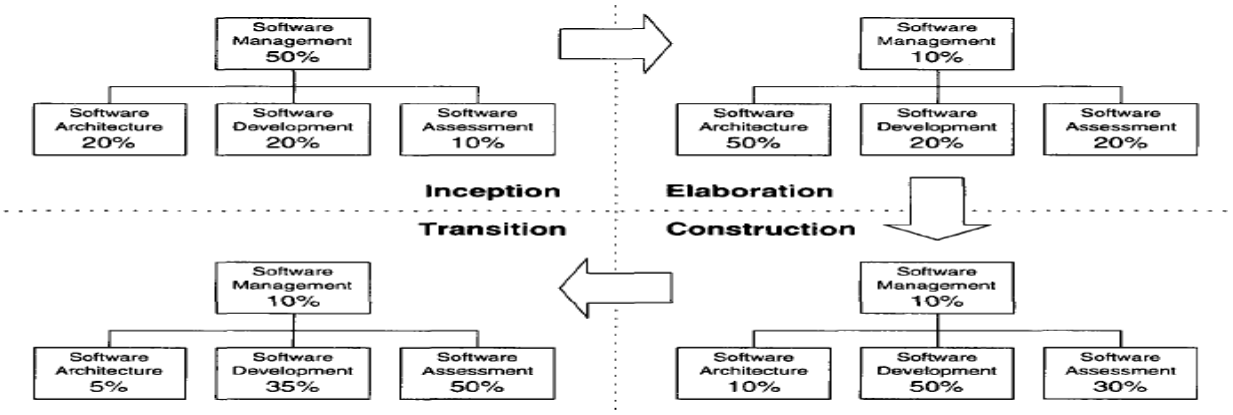
Life-Cycle Focus

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

Software assessment team activities

EVOLUTION OF ORGANIZATIONS

- ❑ The project organization represents the architecture of the teamand needs to evolve consistent with the project plan captured in the work breakdown structure.
- ❑ A different set of activities is emphasized in each phase, as follows:
- ❖ **Inception team:** An organization focused on planning, with enough support from the other teams to ensure that the plans represent a consensus of all perspectives.
- ❖ **Elaboration team:** An architecture-focused organization in which the driving forces of the project reside in the software architecture team and are supported, by the software development and software assessment teams as necessary to achieve a stable architecture baseline.
- ❖ **Construction team:** A fairly balanced organization in which most of the activity resides in the software development and software assessment teams.
- ❖ **Transition team:** A customer-focused organization in which usage feedback drives the deployment activities



Software project team evolution over the life cycle

PROCESS AUTOMATION

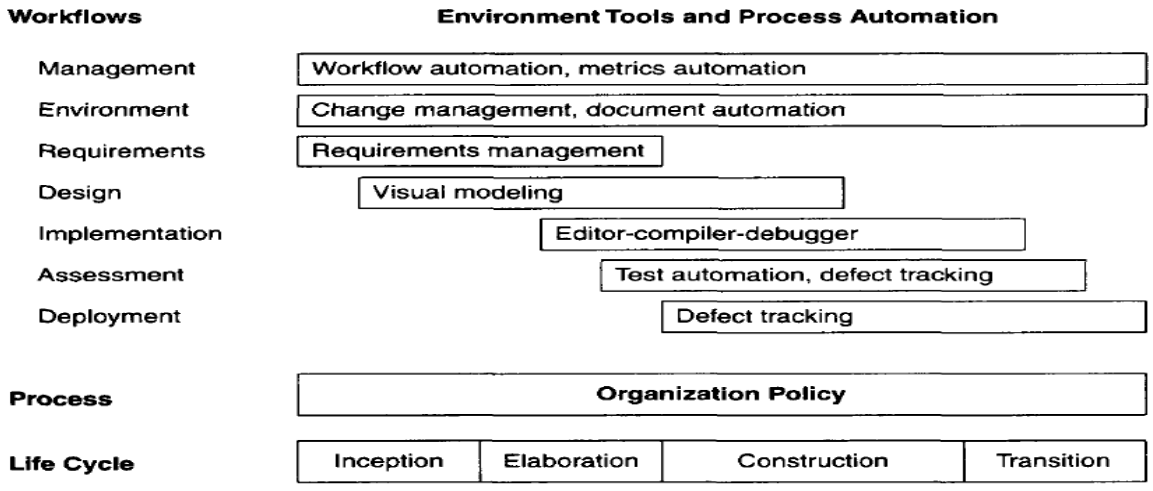
There are 3 levels of process:

1. Metaprocess: An organization‟s policies, procedures, and practices for managing a software intensive line of business.The automation support for this level is called an ***infrastructure***. An infrastructure is an inventory of preferred tools, artifact templates, microprocess guidelines, macroprocess guidelines, project performance repository, database of organizational skill sets, and library of precedent examples of past project plans and results.

2. Macroprocess: A project's policies, procedures, and practices for producing a complete software product within certain cost, schedule, and quality constraints. The automation support for a project's process is called an ***environment.*** An environment is a specific collection of tools to produce a specific set of artifacts as governed by a specific project plan.

3. Microprocess: A project team's policies, procedures, and practices for achieving an artifact of the software process. The automation support for generating an artifact is generally called a ***tool***. Typical tools include requirements management, visual modeling, compilers, editors, debuggers, change management, metrics automation, document automation, test automation, cost estimation, and workflow automation.

Automation Buliding Blocks



Typical automation and tool components that support the process workflows

Management: Software cost estimation tools and WBS tools are useful for generating the planning artifacts. For managing against a plan, workflow management tools and a software project control panel that can maintain an on-line version of the status assessment are advantageous.

Environment: Configuration management and version control are essential in a modern iterative development process. (change management automation that must be supported by the environment.

Requirements: Conventional approaches decomposed system requirements into subsystem requirements, subsystem requirements into component requirements, and component requirements into unit requirements.

The ramifications of this approach on the environment‟s support for requirements management are twofold:

1. The recommended requirements approach is dependent on both textual and model-based representations
2. Traceability between requirements and other artifacts needs to be automated.

Design: The primary support required for the design workflow is visual modeling, which is used for capturing design models, presenting them in human-readable format, and translating them into source code. Architecture-first and demonstration-based process is enabled by existing architecture components and middleware.

Implementation: The implementation workflow relies primarily on a programming environment (editor, compiler, debugger, and linker, run time) but must also include substantial integration with the change management tools, visual modeling tools, and test automation tools to support productive iteration.

Assessment and Deployment: To increase change freedom, testing and document production must be mostly automated. Defect tracking is another important tool that supports assessment: It provides the change management instrumentation necessary to automate metrics and control release baselines. It is also needed to support the deployment workflow throughout the life cycle.

THE PROJECT ENVIRONMENT

The project environment artifacts evolve through three discrete states:

1. The proto typing environment includes an architecture tested for prototyping project architectures to evaluate trade-offs during the inception and elaboration phases of the life cycle. It should be capable of supporting the following activities:
   - technical risk analyses
   - feasibility studies for commercial products
   - Fault tolerance/dynamic reconfiguration trade-offs
   - Analysis of the risks associated implementation
   - Development of test scenarios, tools, and instrumentation suitable for analyzing the requirements.

2. The development environment should include a full suite of development tools needed to support the various process workflows and to support round-trip engineering to the maximum extent possible.

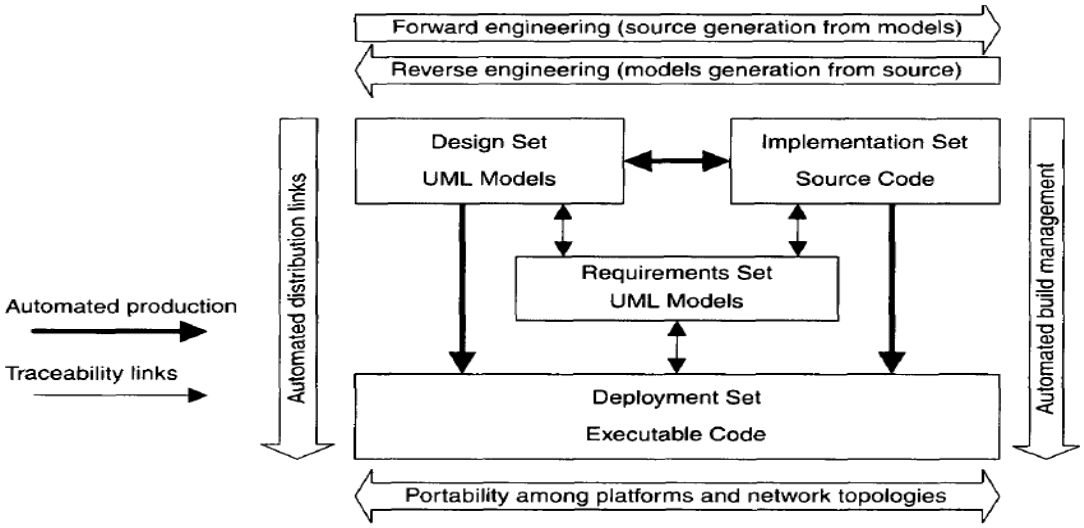
3. The maintenance environment may be a subset of the development environment delivered as one of the project's end products.

Four important environment disciplines that is critical to the management context and the success of a modern iterative development process:

- Tools must be integrated to maintain consistency and traceability. Roundtrip Engineering is the term used to describe this key requirement for environments that support iterative development.
- Change management must be automated and enforced to manage multiple, iterations and to enable change freedom. Change is the fundamental primitive of iterative development.
- Organizational infrastructures A common infrastructure promotes interproject consistency, reuse of training, reuse of lessons learned, and other strategic improvements to the organization's metaprocess.
- Extending automation support for stakeholder environments enables further support for paperless exchange of information and more effective review of engineering artifacts.

Round-Trip Engineering

- Round-trip engineering is the environment support necessary to maintain consistency among theengineering artifacts.
- The primary reason for round-trip engineering is to allow freedom in changing software engineering data sources.



Round-trip engineering

Change Management

- Change management is as critical to iterative processes as planning.
- Tracking changes in the technical artifacts is crucial to understanding the true technical progress trends and quality trends toward delivering an acceptable end product or interim release.
- In a modern process-in which requirements, design, and implementation set artifacts are captured in rigorous notations early in the life cycle and are evolved through multiple generations-change management has become fundamental to all phases and almost all activities.

Software Change Orders (SCO)

- The atomic unit of software work that is authorized to create, modify, or obsolesce components within a configuration baseline is called a software change order (SCO).
- Software change orders are a key mechanism for partitioning, allocating, and scheduling software work against an established software baseline and for assessing progress and quality.

The basic fields of the SCO are title, description, metrics, resolution, assessment and disposition.

a) Title. The title is suggested by the originator and is finalized upon acceptance by the configuration control board.

b) Description: The problem description includes the name of the originator, date of origination, CCB-assigned SCO identifier, and relevant version identifiers of related support software.

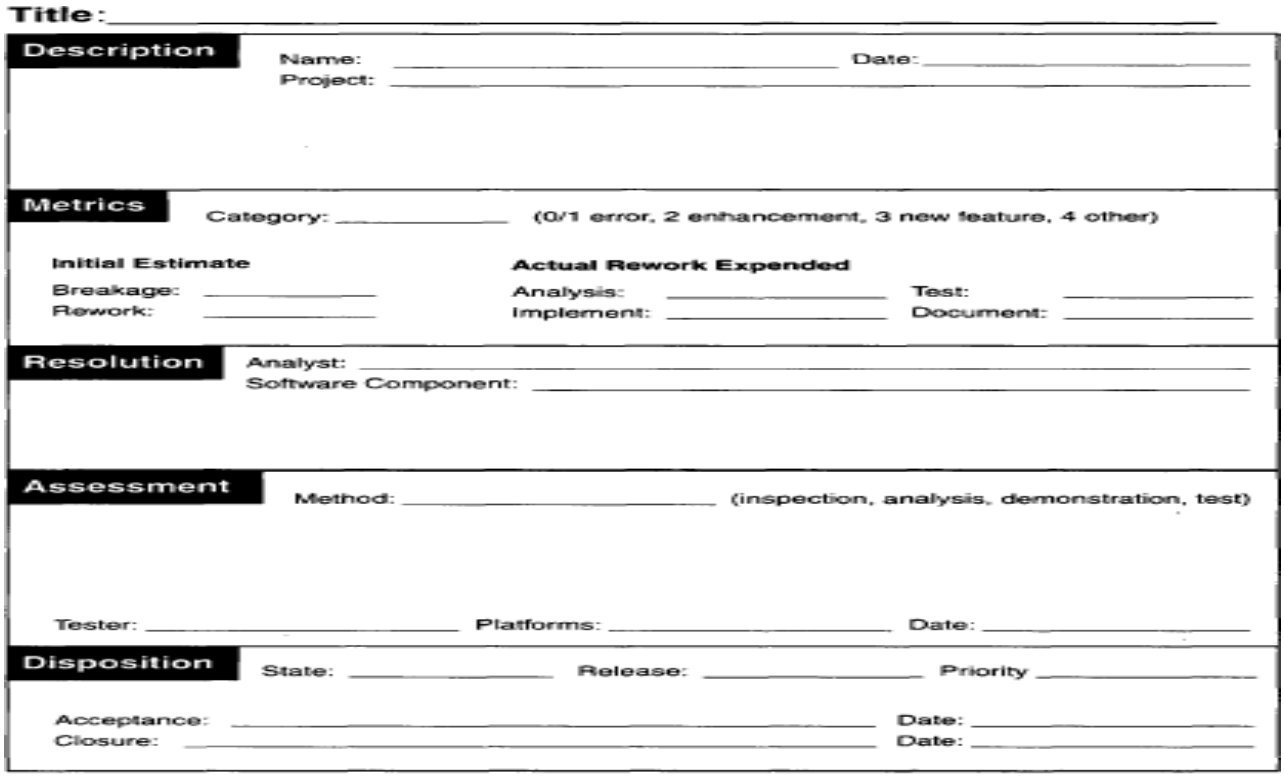
c) Metrics: The metrics collected for each sea are important for planning, for scheduling, and for assessing quality improvement. Change categories are type 0 (critical bug), type 1 (bug), type 2 (enhancement), type 3 (new feature), and type 4 (other)

d) Resolution: This field includes the name of the person responsible for implementing the change, the components changed, the actual metrics, and a description of the change.

e) Assessment: This field describes the assessment technique as either inspection, analysis, demonstration, or test. Where applicable, it should also reference all existing test cases and new test cases executed, and it should identify all different test configurations, such as platforms, topologies, and compilers.

f) Disposition: The SCO is assigned one of the following states by the CCB:

- **Proposed:** written, pending CCB review
- **Accepted:** CCB-approved for resolution
- **Rejected:** closed, with rationale, 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.



The primitive components of a software change order

Configuration Baseline

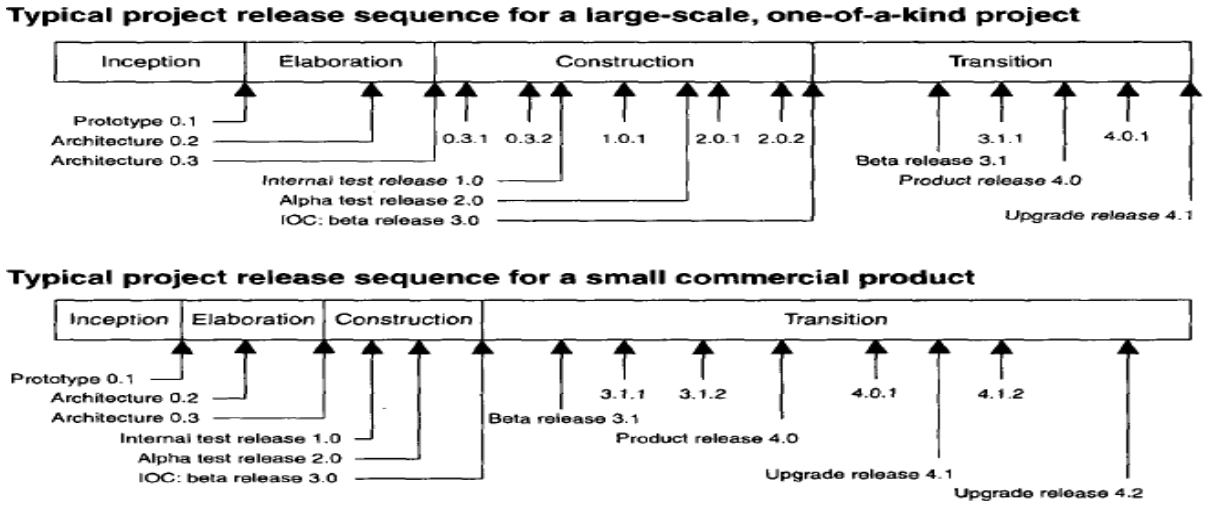
A configuration baseline is a named collection of software components and supporting documentation that is subject to change management and is upgraded, maintained, tested, statused and obsolesced as a unit.

There are general1y two classes of baselines:

1. external product releases and
2. internal testing releases.

A configuration baseline is a named collection of components that is treated as a unit. It is controlled formally because it is a packaged exchange between groups. A project may release a configuration baseline to the user community for beta testing. Once software is placed in a controlled baseline, all changes are tracked. A distinction must be made for the cause of a change. Change categories are as follows:

- Type 0: Critical failures, which are defects that are nearly always fixed before any external release.
- Type 1: A bug or defect that either does not impair the usefulness of the system or can be worked around.
- Type 2: A change that is an enhancement rather than a response to a defect.
- Type 3: A change that is necessitated by an update to the requirements.
- Type 4: changes that are not accommodated by the other categories.



Example release histories for a typical project and a typical product

Configuration Control Board (CCB)

- A CCB is a team of people that functions as the decision authority on the content of configuration baselines.
- A CCB usually includes the software manager, software architecture manager, software development manager, software assessment manager and other stakeholders (customer, software project manager, systems engineer, user) who are integral to the maintenance of a controlled software delivery system.
- The [bracketed] words constitute the state of an SCO transitioning through theprocess.
- [Proposed]: A proposed change is drafted and submitted to the CCB. The proposed change must includea technical description of the problem and an estimate of the resolution effort.
- [Accepted, archived or rejected]: The CCB assigns a unique identifier and accepts, archives, or rejects each proposed change. Acceptance includes the change for resolution in the next release; archiving accepts the change but postpones it for resolution in a future release; and rejection judges the change to be without merit, redundant with other proposed changes, or out of scope.
- [In progress]: the responsible person analyzes, implements and tests a solution to satisfy the SCQ. This task includes updating documentation, release notes and SCO metrics actuals and submitting new SCOs.
- [In assessment]: The independent test assesses whether the SCO is completely resolved. When the independent test team deems the change to be satisfactorily resolved, the SCO is submitted to the CCB for final disposition and closure.
- [Closed]: when the development organization, independent test organization and CCB concur that the SCO is resolved, it is transitioned to a closed status. „

Infrastructures

Organization‟s infrastructure provides the organization capital assets, including two key artifacts:
a) a policy that captures the standards for project software development processes, and
b) an environment that captures an inventory of tools.

Organization Policy

- The organization policy is usually packaged as a handbook that defines the life cycle and the process primitives (major milestones, intermediate artifacts, engineering repositories, metrics, roles and responsibilities). The handbook provides a general framework for answering the following questions:
  - What gets done? (activities and artifacts)
  - When does it get done? (mapping to the life-cycle phases and milestones)
  - Who does it? (team roles and responsibilities)
- How do we know that it is adequate? (Checkpoints, metrics and standards of performance).

I. **Process-primitive definitions**
   A. Life-cycle phases (inception, elaboration, construction, transition)
   B. *Checkpoints (major milestones, minor milestones, status assessments)*
   C. Artifacts (requirements, design, implementation, deployment, management sets)
   D. Roles and responsibilities (PRA, SEPA, SEEA, project teams)

II. **Organizational software policies**
   A. Work breakdown structure
   B. Software development plan
   C. *Baseline change management*
   D. Software metrics
   E. Development environment
   F. Evaluation criteria and acceptance criteria
   G. *Risk management*
   H. Testing and assessment

III. **Waiver policy**

IV. **Appendixes**
   A. Current process assessment
   B. Software process improvement plan

Organization policy outline

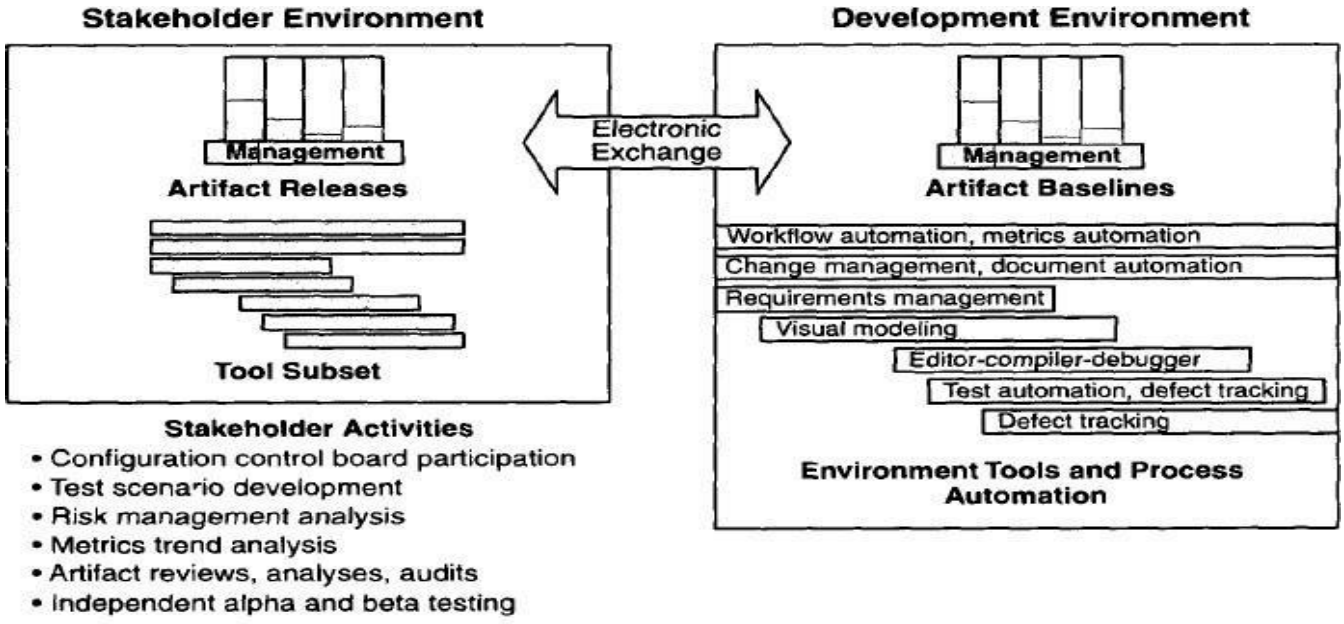
Organization Environment

Some of the typical components of an organization‟s automation building blocks are as follows:

- Standardized tool selections, which promote common workflows and a higher ROI on training.
- Standard notations for artifacts, such as UML for all design models, or Ada 95 for all custom-developed, reliability-critical implementation artifacts.
- Tool adjuncts such as existing artifact templates (architecture description, evaluation criteria, release descriptions, status assessment) or customizations.
- Activity templates (iteration planning, major milestone activities, configuration control boards).

Stakeholder Environments

- An on-line environment accessible by the external stakeholders allows them to participate in the process as follows:
  - Accept and use executable increments for hands-on evaluation.
  - Use the same on-line tools, data and reports that the software development organization uses to manageand monitor the project.
  - Avoid excessive travel, paper interchange delays, format translations, paper and shipping costs and other overhead costs.
- There are several important reasons for extending development environment resources into certainstakeholder domains.
  - Technical artifacts are not just paper.
  - Reviews and inspections, breakage/rework assessments, metrics analyses and even beta testing canbe performed independently of the development team.
  - Even paper documents should be delivered electronically to reduce production costs and turn around time.



Extending environments into stakeholder domains