23CS712: NR23: Software Process & Project Management: 23CS712

Dr. P. Dileep Kumar Reddy

Professor-Dean-R&D,IPR, IIC

CSE Department

Narsimha Reddy Engineering College (Autonomous)

Secunderabad, Telangana State, India- 500100.

Ph.No: 09959845657

4.12. PROJECT ORGANIZATION AND RESPONSIBILITIES

INTRODUCTION: 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.

4.12.1 LINES-OF-BUSINESS ORGANIZATIONS: Figure 11-1 maps roles and responsibilities to a default line-of-business organization. This structure can be tailored to specific circumstances.

- 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.

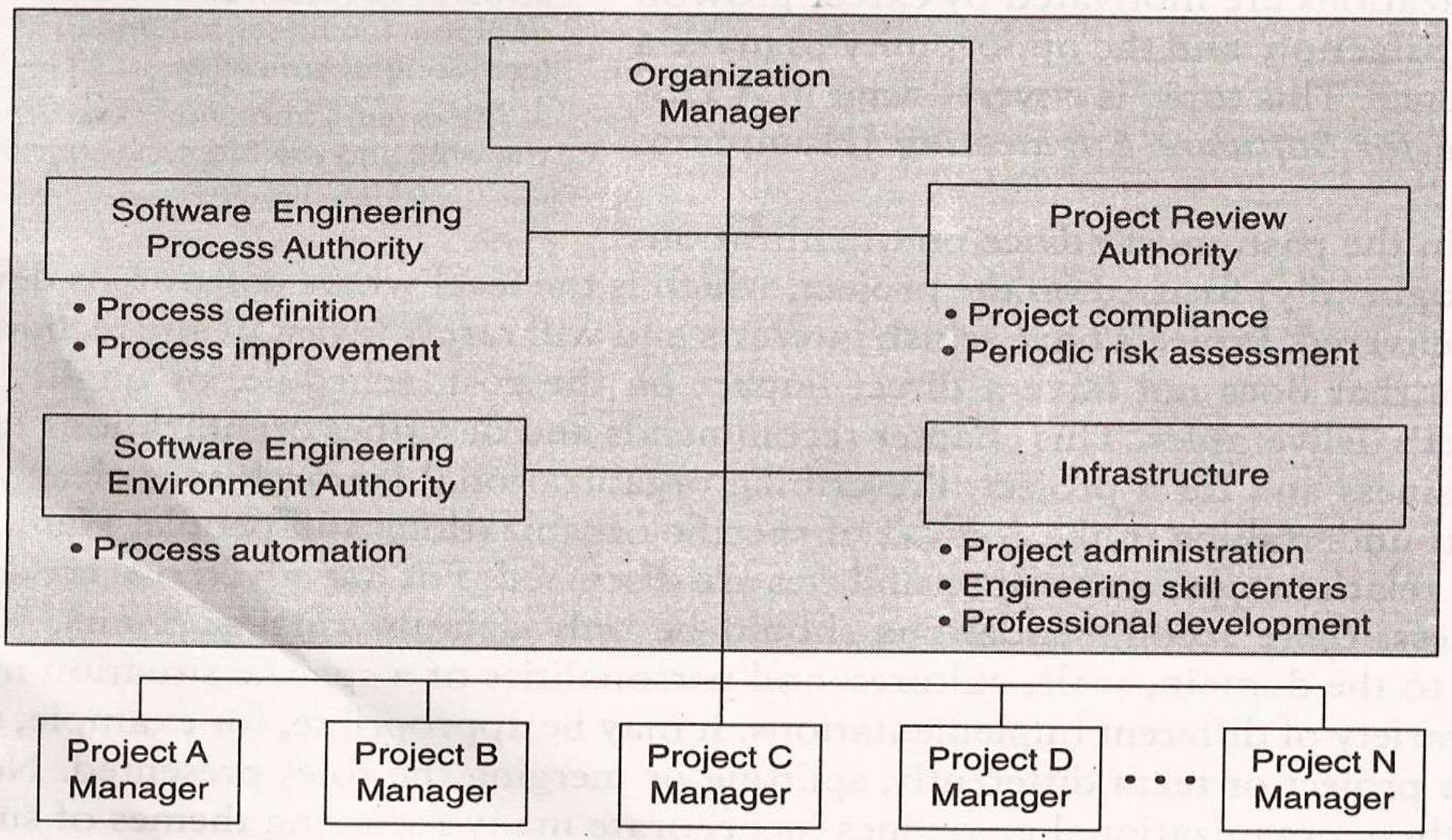


FIGURE 11-1. *Default roles in a software line-of-business organization*

- The line of business organization consists of four component teams.

- **SOFTWARE ENGINEERING PROCESS AUTHORITY**
- The software engineering process authority (SEPA) is responsible for exchanging the information and project guidance to or from the project practitioners.

- **PROJECT REVIEW AUTHORITY**
- The project review Authority (PRA) is responsible for reviewing the financial performance, customer commitments, risks and accomplishments, adherence to organizational policies by the customer etc.

- **SOFTWARE ENGINEERING ENVIRONMENT AUTHORITY**
- The software Engineering Environment Authority (SEEA) deals with the maintenance or organizations standard environment, training projects and process automation.

- **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.**

4.13 PROJECT ORGANIZATIONS

Figure 11-2 shows a 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 are 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.

***The assessment team* is separate form development.**

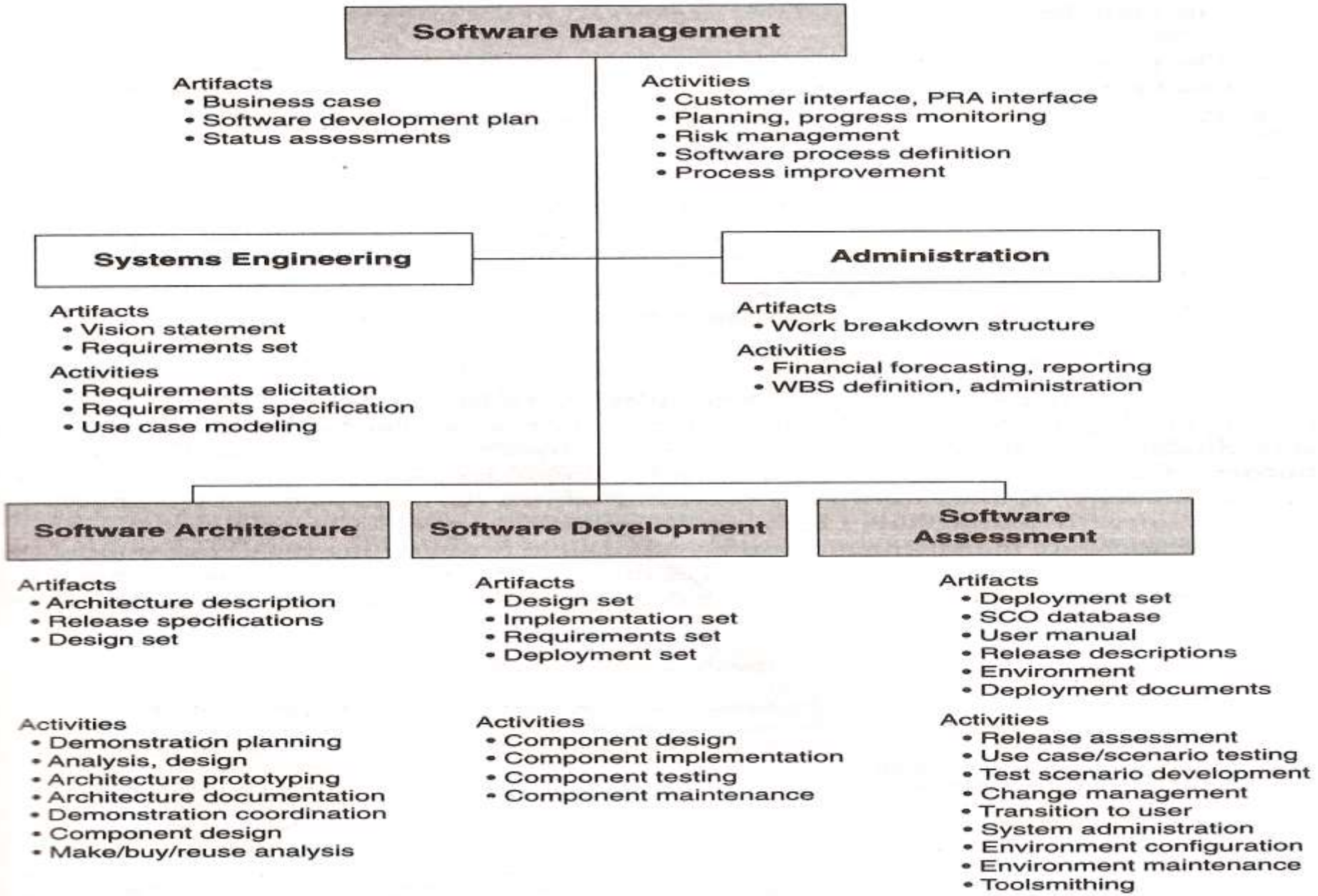


FIGURE 11-2. *Default project organization and responsibilities*

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: Software management team is responsible for:

- 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

SOFTWARE ARCHITECTURE TEAM

The software architecture team performs the tasks of integrating the components, creating real artifacts etc. The skill possessed by the architecture team is of utmost importance as 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-case view.

Software technology experience to generate an acceptable process view, component and development views.

- Responsibilities: Software architecture team is responsible for:
- System-level quality i.e., performance, reliability and maintainability.
- Requirements and design trade-offs.
- Component selection
- Technical risk solution
- Initial integration

SOFTWARE DEVELOPMENT TEAM

- The Development team is involved in the construction and maintenance activities. It is most application specific 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: Software development team is responsible for
  - Component development, testing and maintenance.
  - Component design and implementation
  - Component documentation.

- **SOFTWARE DEVELOPMENT TEAM**
  - The Development team is involved in the construction and maintenance activities. It is most application specific 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: Software development team is responsible for
    - Component development, testing and maintenance.
    - Component design and implementation
    - Component documentation.

4.14 EVOLUTION OF ORGANIZATIONS

- The project organization represents the architecture of the team and needs to evolve consistent with the project plan captured in the work breakdown structure. Figure 11-7 illustrates how the team's center of gravity shifts over the life cycle, with about 50% of the staff assigned to one set of activities in each phase.
- 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

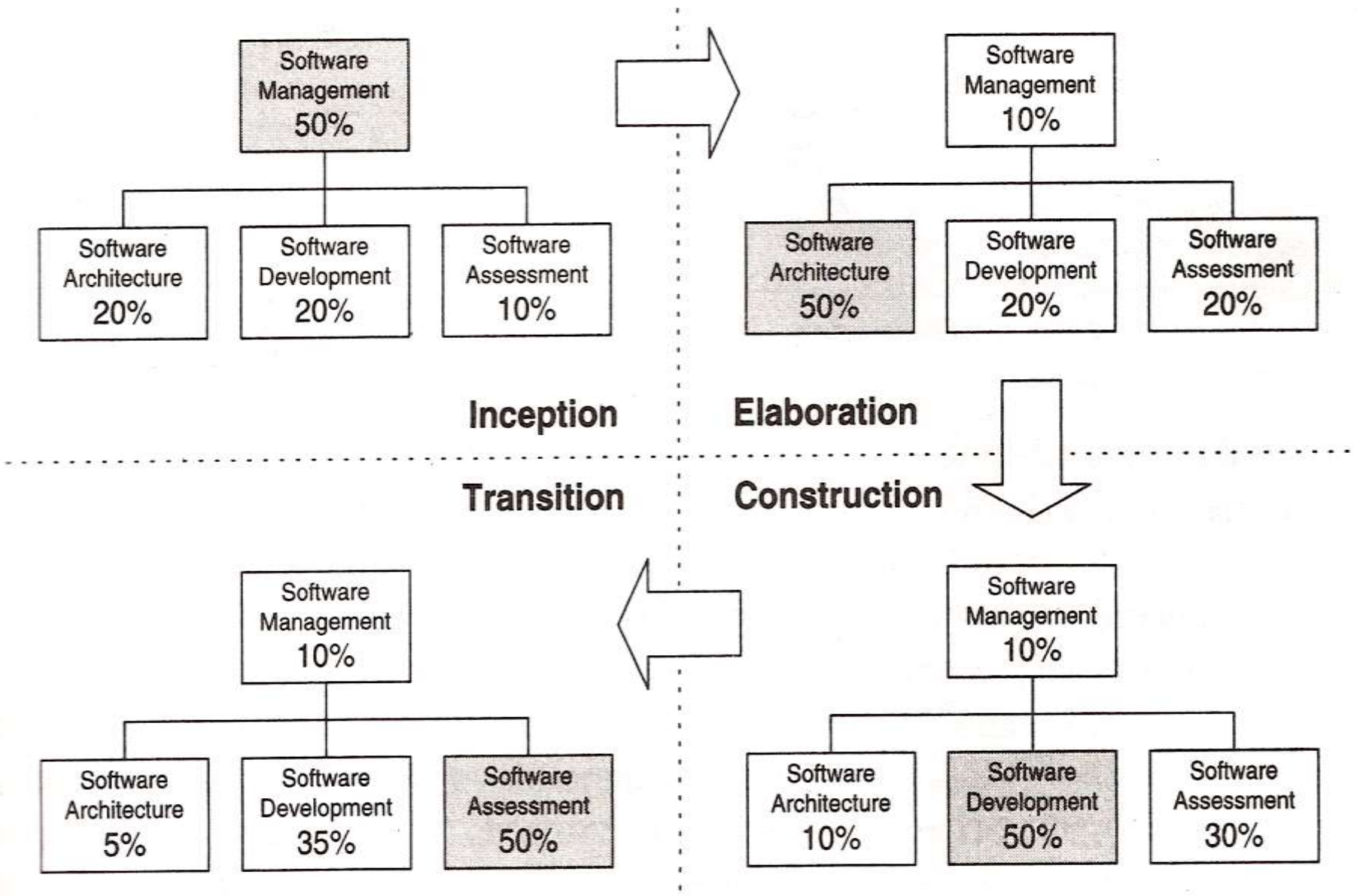


FIGURE 11-7. *Software project team evolution over the life cycle*

4.15 PROCESS AUTOMATION

- Three levels of process are
- **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.
- **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.
- **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.

4.16 TOOLS: AUTOMATION BUILDING BLOCKS

- It introduces some of the important tools that tend to be needed universally across software projects and that correlate well to the process framework. (Many other tools and process automation aids are not included.) Most of the core software development tools map closely to one of the process workflows, as illustrated ill Figure 12-1.

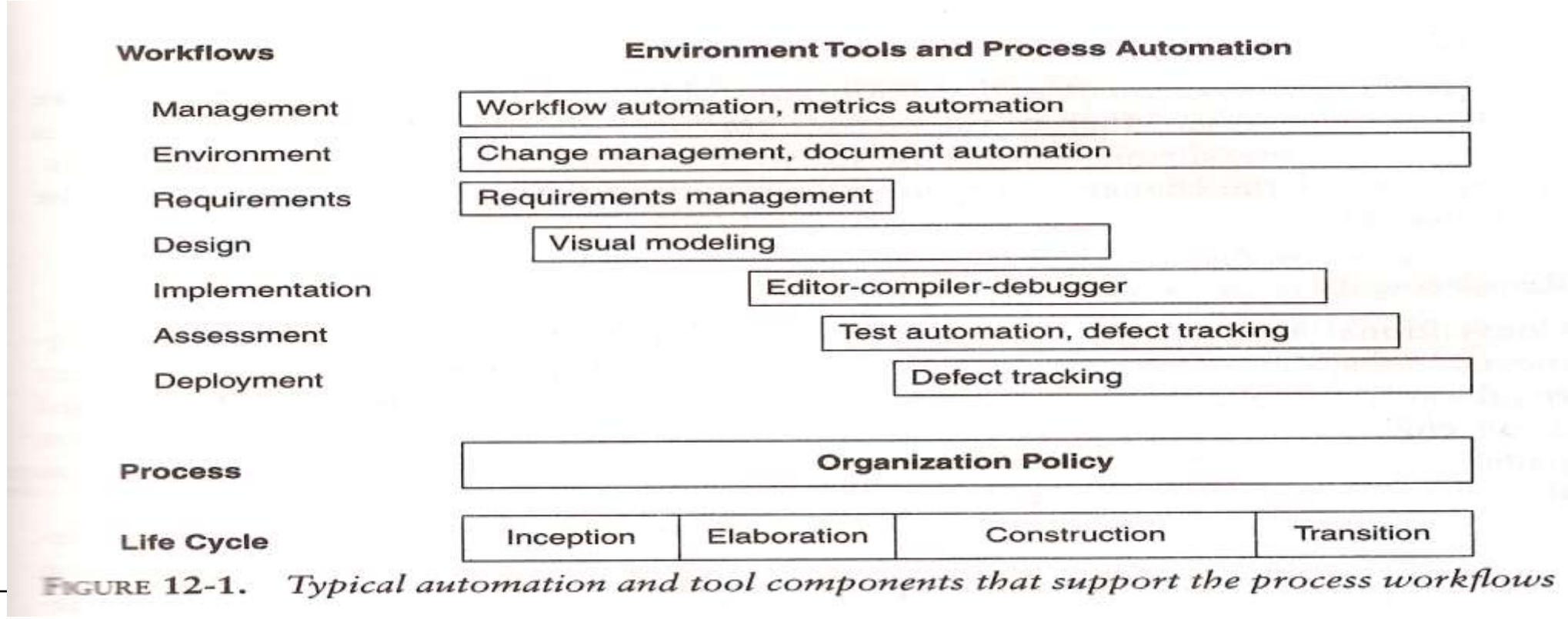


FIGURE 12-1. *Typical automation and tool components that support the process workflows*

- **MANAGEMENT**
  - There are many opportunities for automating the project planning and control activities of the management workflow. 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. This automation support can considerably improve the insight of the metrics collection and reporting concepts.
- **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 equal treatment of all requirements drained away engineering hours from the driving requirements then wasted that time on paperwork associated with detailed traceability that was inevitably discarded later as the driving requirements and subsequent design understanding evolved.
  - 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 too1s that Support the requirements, design, implementation, and assessment workflows are usually used together. 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, 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**
  - The assessment workflow requires all the tools just discussed as well as additional capabilities to support test automation and test management. 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.

4.17 THE PROJECT ENVIRONMENT

- The project environment artifacts evolve through three discrete states: the prototyping environment, the development environment, and the maintenance environment.
- 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. This informal configuration of tools should be capable of supporting the following activities:
  - **Performance trade-offs and technical risk analyses**
  - **Make /buy trade-offs and feasibility studies for commercial products**
  - **Fault tolerance/dynamic reconfiguration trade-offs**
  - **Analysis of the risks associated with transitioning to full-scale implementation**
  - **Development of test scenarios, tools, and instrumentation suitable for analyzing the requirements.**
- 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.
- The maintenance environment should typically coincide with a mature version of the development environment. In some cases, 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.

4.17.1 ROUND-TRIP ENGINEERING

- Round-trip engineering is the environment support necessary to maintain consistency among the engineering artifacts.
- Figure 12-2 depicts some important transitions between information repositories. The automated translation of design models to source code (both forward and reverse engineering) is fairly well established. The automated translation of design models to process (distribution) models is also becoming straightforward through technologies such as ActiveX and the Common Object Request Broker Architecture (CORBA).
- The primary reason for round-trip engineering is to allow freedom in changing software engineering data sources.

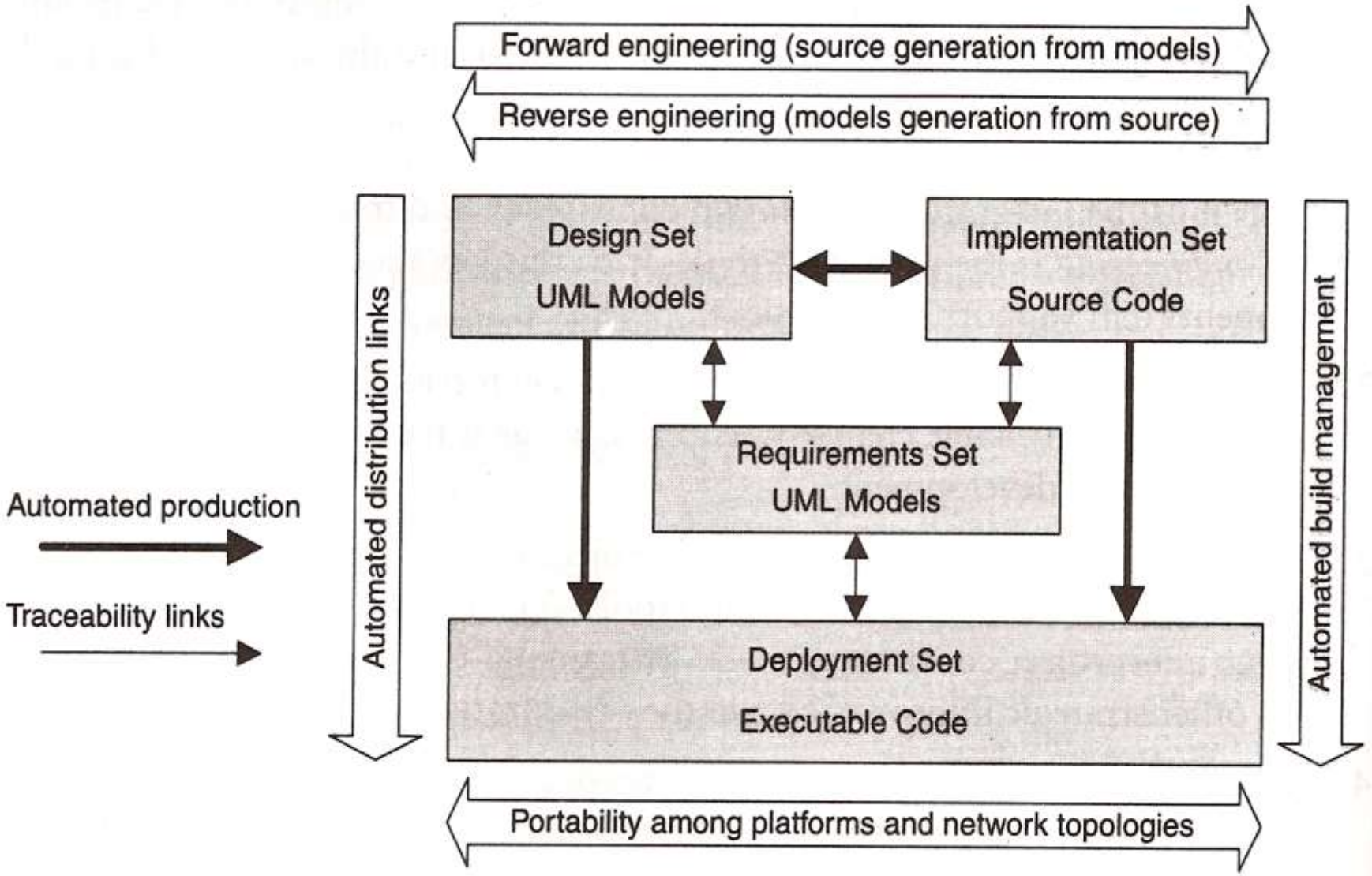


FIGURE 12-2. *Round-trip engineering*

4.17.2 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

- 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 example SCO shown in Figure 12-3 is a good starting point for describing a set of change primitives. It shows the level of detail required to achieve the metrics and change management rigor necessary for a modern software process.
- The basic fields of the SCO are title, description, metrics, resolution, assessment and disposition.
- **Title.** The title is suggested by the originator and is finalized upon acceptance by the configuration control board (CCB).

- **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.
- **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)
- **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.

- **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.
- **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.

Title: _____

Description

Name: _____ Date: _____

Project: _____

Metrics

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

Initial Estimate

Breakage: _____

Rework: _____

Actual Rework Expended

Analysis: _____ Test: _____

Implement: _____ Document: _____

Resolution

Analyst: _____

Software Component: _____

Assessment

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

Tester: _____ Platforms: _____ Date: _____

Disposition

State: _____ Release: _____ Priority _____

Acceptance: _____ Date: _____

Closure: _____ Date: _____

FIGURE 12-3. *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.

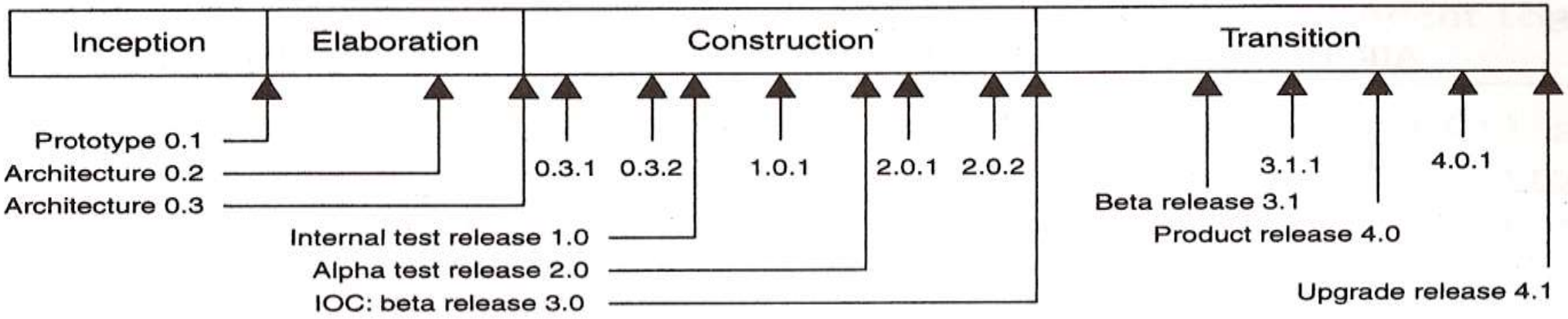
There are general1y two classes of baselines: external product releases and internal testing releases.

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.

Generally, three levels of baseline releases arc required for most systems: major, minor, and interim. Each level corresponds to a numbered identifier such as N.M.X, where N is the major release number, M is the minor release number, and X is the interim release identifier. A major release represents a new generation of the product or project, while a minor release represents the same basic product but with enhanced features, performance, or quality. Major and minor releases are intended to be external product releases that are persistent and supported for a period of time. An interim release corresponds to a developmental configuration that is intended to be transient. The shorter its life cycle, the better. Figure 12-4 shows examples of some release name histories for two different situations.

- **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.**
- **Table 12-1 provides examples of these changes in the context of two different project domains: a large-scale, reliable air traffic control system and a packaged software development tool**

Typical project release sequence for a large-scale, one-of-a-kind project



Typical project release sequence for a small commercial product

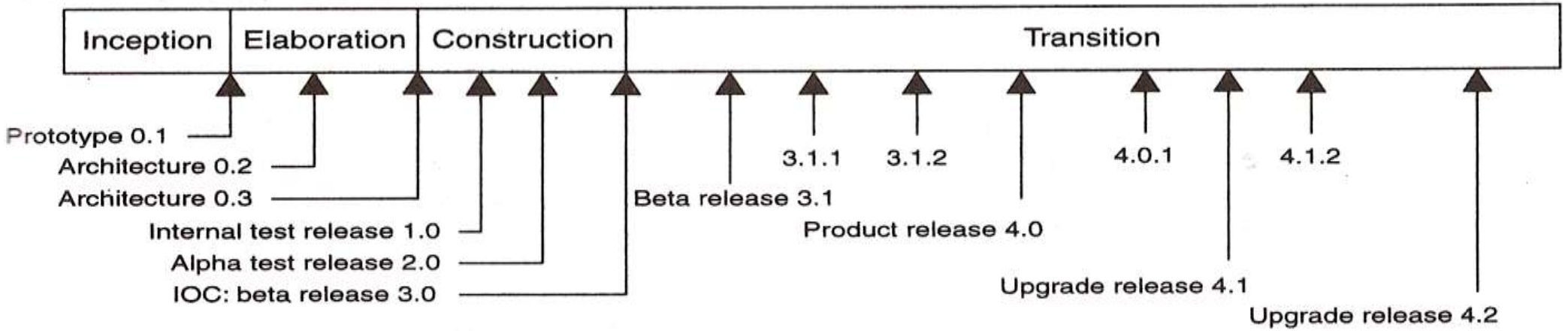


FIGURE 12-4. *Example release histories for a typical project and a typical product*

Change Type	Air Traffic control Project	Packaged visual Modeling Tool
Type 0	Control deadlock and loss of flight data	Loss of user data
Type 1	Display response time that exceeds the requirement by 0.5 second	Browser expands but does not collapse displayed entries
Type 2	Add internal message field for response time instrumentation	Use of color to differentiate updates from previous version of visual model
Type 3	Increase air traffic management capacity from 1,200 to 2,400 simultaneous flights	Port to new platform such as WinNT
Type 4	Upgrade from Oracle 7 to Oracle 8 to improve query performance	Exception raised when interfacing to MS Excel 5.0 due to windows resource management bug.

CONFIGURATION CONTROL BOARD

- 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 the process.
- [Proposed]: A proposed change is drafted and submitted to the CCB. The proposed change must include a 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.

4.17.3 INFRASTRUCTURES

- From a process automation perspective, the organization's infrastructure provides the organization capital assets, including two key artifacts: a policy that captures the standards for project software development processes, and 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).

- The need for balance is an important consideration in defining organizational policy. Effective organizational policies have several recurring themes:
    - They are concise and avoid policy statements that fill 6-inch-thick documents.
    - They confine the policies to the real shalls, then enforce them.
    - They avoid using the word should in policy statements. Rather than a menu of options (shoulds), policies need a concise set of mandatory standards (shalls).
    - Waivers are the exception, not the rule.
- Appropriate policy is written at the appropriate level.
- The organization policy is the defining document for the organization's software policies. In any process assessment, this is the tangible artifact that says what you do. From this document, reviewers should be able to question and review projects and personnel and determine whether the organization does what it says. Figure 12-5 shows a general outline for an organizational policy.

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. **Organization 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. **Walver policy**

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

FIGURE 12-5: Organization policy outline

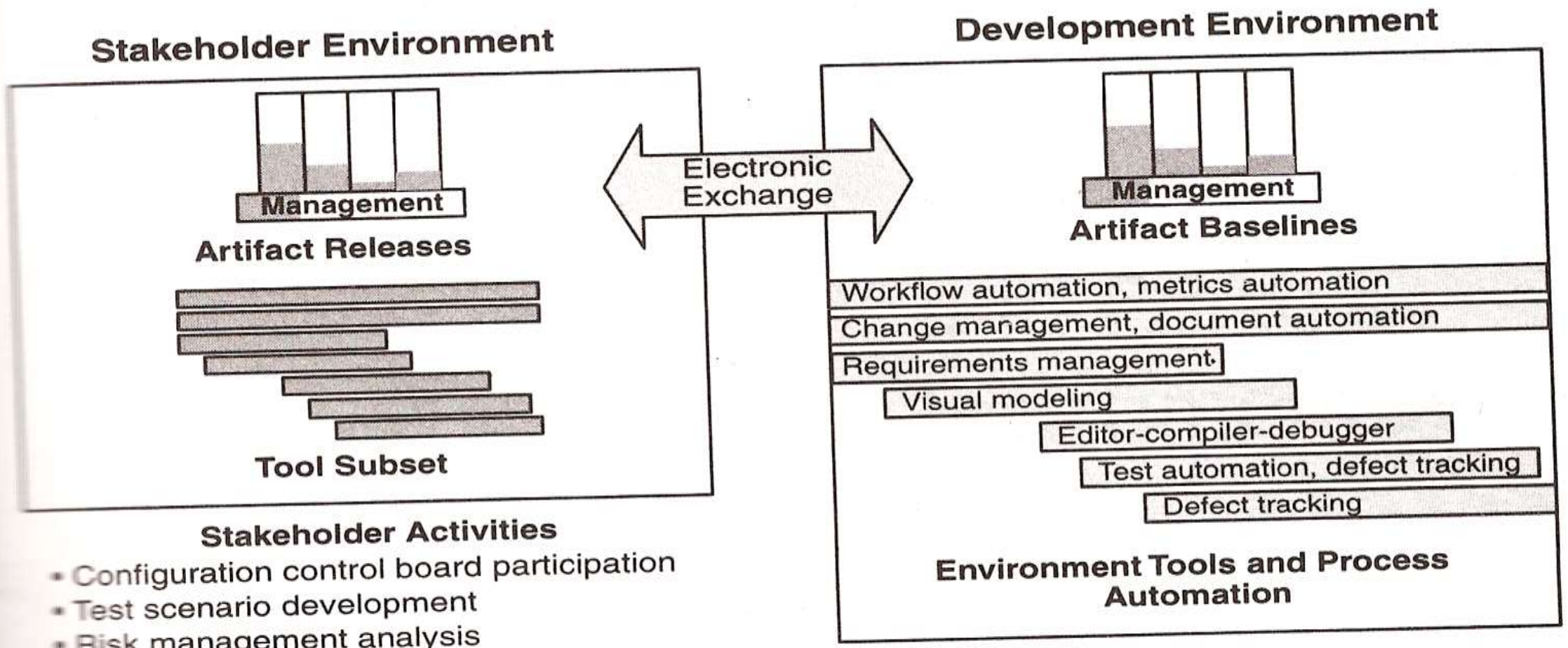
- **ORGANIZATION ENVIRONMENT**
  - Some of the typical components of an organization's automation building blocks are as follows:
    - Standardized tool selections (through investment by the organization in a site license or negotiation of a favorable discount with a tool vendor so that project teams are motivated economically to use that tool), 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).

- Other indirectly useful components of an organization's infrastructure
  - A reference library of precedent experience for planning, assessing and improving process performance parameters; answers for how well? How much? Why?
  - Existing case studies, including objective benchmarks of performance for successful projects that followed the organization process.
  - A library of project artifact examples such as software development plans, architecture descriptions and status assessment histories.
  - Mock audits and compliance traceability for external process assessment frameworks.
- Such as the software Engineering Institute's Capability Maturity Model (SEI CMM)

4.17.4 STAKEHOLDER ENVIRONMENTS

- The transition to a modern iterative development process with supporting automation should not be restricted to the development team. many large scale contractual projects include people in external organization that represent other stakeholders participating in the development process.
- 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 manage and monitor the project.
  - Avoid excessive travel, paper interchange delays, format translations, paper and shipping costs and other overhead costs.

- **FIGURE 12-6:** Illustrates some of the new opportunities for value-added activities by external stakeholders in large contractual efforts. There are several important reasons for extending development environment resources into certain stakeholder domains.
  - Technical artifacts are not just paper. Electronic artifacts in rigorous notations such as visual models and source code are viewed far more efficiently by using tools with smart browsers.
  - Independent assessments of the evolving artifacts are encouraged by electronic read-only access to on-line data such as configuration baseline libraries and the change management database. Reviews and inspections, breakage/rework assessments, metrics analyses and even beta testing can be performed independently of the development team.
  - Even paper documents should be delivered electronically to reduce production costs and turn around time.



Stakeholder Activities

- Configuration control board participation
- Test scenario development
- Risk management analysis
- Metrics trend analysis
- Artifact reviews, analyses, audits
- Independent alpha and beta testing

FIGURE 12-6. *Extending environments into stakeholder domains*

1. MODEL QUESTIONS

1. What are the main features of the default line-of-business organization?
2. What are the four component teams in a default line-of-business organization and their responsibility?
3. What are the four component teams in a default project organization and their responsibility?
4. How does the emphasis in the four teams evolve over the course of the entire project?
5. What is the reason for looking at organizations from project as well as line-of-business perspective?
6. What are the steps in identifying project roles? Name any five project roles and the skills needed for them.
7. What are the benefits of matching people to roles?
8. Explain the process of critical path method on the following PERT chart with requirements, coding and system taking 4 weeks and each others taking 2 weeks each. Shown in figure.
9. Discuss the evolution of software project team over the software life cycle.
10. Discuss the three levels of process along with their automation support.

11. ‘Many automation tools are available for software development process’. Support your answer.
12. Discuss the major critical concerns associated with the software workflows.
13. What are the three discrete states of the project environment? Also discuss the four environment disciplines that are critical to the management.
14. Explain briefly about Round-trip engineering.
15. What are software change orders? Explain the various fields of SCO.
16. What are the sources of change? Why should change be made in a controlled way?
17. Define a configuration baseline.
18. Write short notes on
    - Configuration Control Board (CCB)
    - Organization’s Infrastructure.
19. Discuss briefly about the stakeholder environments.

_____

Thank You..

PROJECT CONTROL AND PROCESS INSTRUMENTATION

Dr. P. Dileep Kumar Reddy

Professor-Dean-R&D,IPR, IIC

CSE Department

Narsimha Reddy Engineering College (Autonomous)

Secunderabad, Telangana State, India- 500100.

Ph.No: 09959845657

5.0 INTRODUCTION

- The primary themes of a modern software development process tackle the central management issues of complex software
  - Getting the design right by focusing on the architecture first
    - Managing risk through iterative development
    - Reducing the complexity with component based techniques
    - Making software progress and quality tangible through instrumented change management
    - Automating the overhead and bookkeeping activities through the use of round-trip engineering and integrated environments
      - The goals of software metrics are to provide the development team and the management team with the following:
        - An accurate assessment of progress to date
        - Insight into the quality of the evolving software product
- A basis for estimating the cost and schedule for completing the product with increasing accuracy over time.

5.1 THE SEVEN CORE METRICS

Seven core metrics are used in all software projects. Three are management indicators and four are quality indicators.

MANAGEMENT INDICATORS

Work and progress (work performed over time)

Budgeted cost and expenditures (cost incurred over time)

Staffing and team dynamics (personnel changes over time)

QUALITY INDICATORS

Change traffic and stability (change traffic over time)

Breakage and modularity (average breakage per change over time)

Rework and adaptability (average rework per change overtime)

Mean time between failures (MTBF) and maturity (defect rate over time)

- Table 13-1 describes the core software metrics. Each metric has two dimensions: a static *value* used as an objective, and the dynamic *trend* used to manage the achievement of that objective. While metrics values provide one dimension of insight, metrics trends provide a more important perspective for managing the process. Metrics trends with respect to time provide insight into how the process and product are evolving.

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

- The seven core metrics are based on common sense and field experience with both successful and unsuccessful metrics programs. Their attributes include the following:
  - They are simple, objective, easy to collect, easy to interpret and hard to misinterpret.
  - Collection can be automated and non intrusive.
  - They provide for consistent assessment throughout the life cycle and are derived from the evolving product baselines rather than from a subjective assessment.
  - They are useful to both management and engineering personnel for communicating progress and quality in a consistent format.
  - Their fidelity improves across the life cycle.

5.2 MANAGEMENT INDICATORS

- There are three fundamental sets of management metrics; technical progress, financial status staffing progress. By examining these perspectives, management can generally assess whether a project is on budget and on schedule. The management indicators recommended here include standard financial status based on an earned value system, objective technical progress metrics tailored to the primary measurement criteria for each major team of the organization and staff metrics that provide insight into team dynamics.

5.2.1 Work & Progress

- The various activities of an iterative development project can be measured by defining a planned estimate of the work in an objective measure, then tracking progress (work completed over time) against that plan

(Figure 13-1), the default perspectives of this metric would be as follows:

- Software architecture team: use cases demonstrated
- Software development team: SLOC under baseline change management, SCOs closed.
- Software assessment team: SCOs opened, test hours executed, evaluation criteria met
- Software management team: milestones completed

5.2.2 BUDGETED COST AND EXPENDITURES

- To maintain management control, measuring cost expenditures over the project life cycle is always necessary. One common approach to financial performance measurement is use of an earned value system, which provides highly detailed cost and schedule insight.
- Modern software processes are amenable to financial performance measurement through an earned value approach. The basic parameters of an earned value system, usually expressed in units of dollars, are as follows:
  - **Expenditure Plan**: the planned spending profile for a project over its planned schedule. For most software projects (and other labor-intensive projects), this profile generally tracks the staffing profile.
  - **Actual Progress**: the technical accomplishment relative to the planned progress underlying the spending profile. In a healthy project, the actual progress tracks planned progress closely.
  - **Actual Cost**: the actual spending profile for a project over its actual schedule. In a healthy project, this profile tracks the planned profile closely.

Earned Value: the value that represents the planned cost of the actual progress.

Cost variance: the difference between the actual cost and the earned value.

Positive values correspond to over - budget situations; negative values correspond to under budget situations.

Schedule Variance: the difference between the planned cost and the earned value. Positive values correspond to behind-schedule situations; negative values correspond to ahead-of-schedule situations

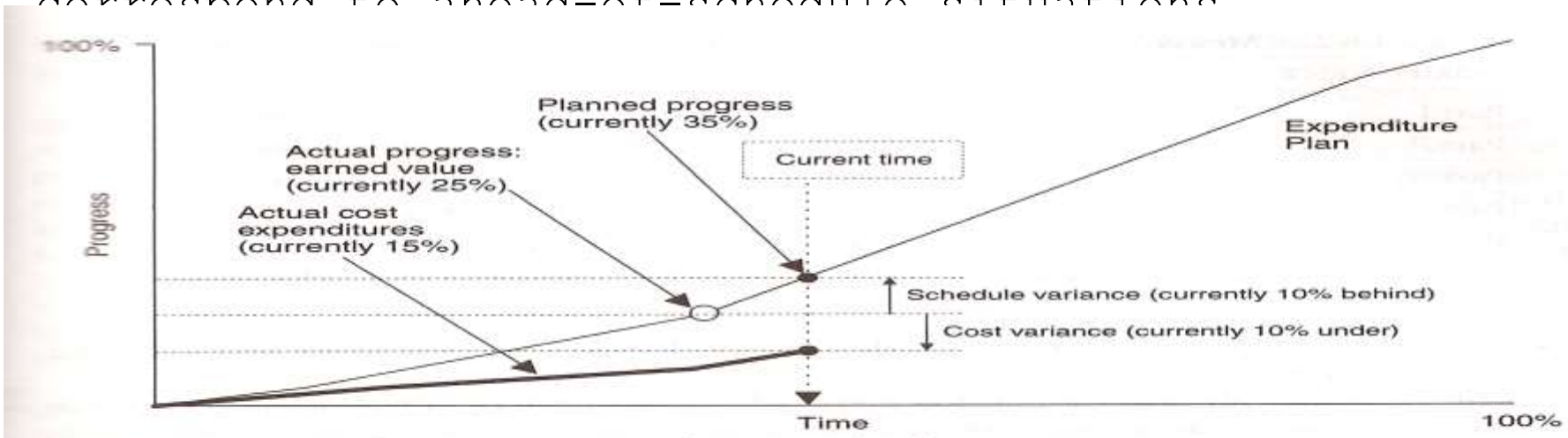


FIGURE 13-2. *The basic parameters of an earned value system*

5.2.3 STAFFING AND TEAM DYNAMICS

- An iterative development should start with a small team until the risks in the requirements and architecture have been suitably resolved. Depending on the overlap of iterations and other project specific circumstance, staffing can vary. For discrete, one of-a-kind development efforts (such as building a corporate information system), the staffing profile in figure 13-4 would be typical. It is reasonable to expect the maintenance team to be smaller than the development team for these sorts of developments. For a commercial product and

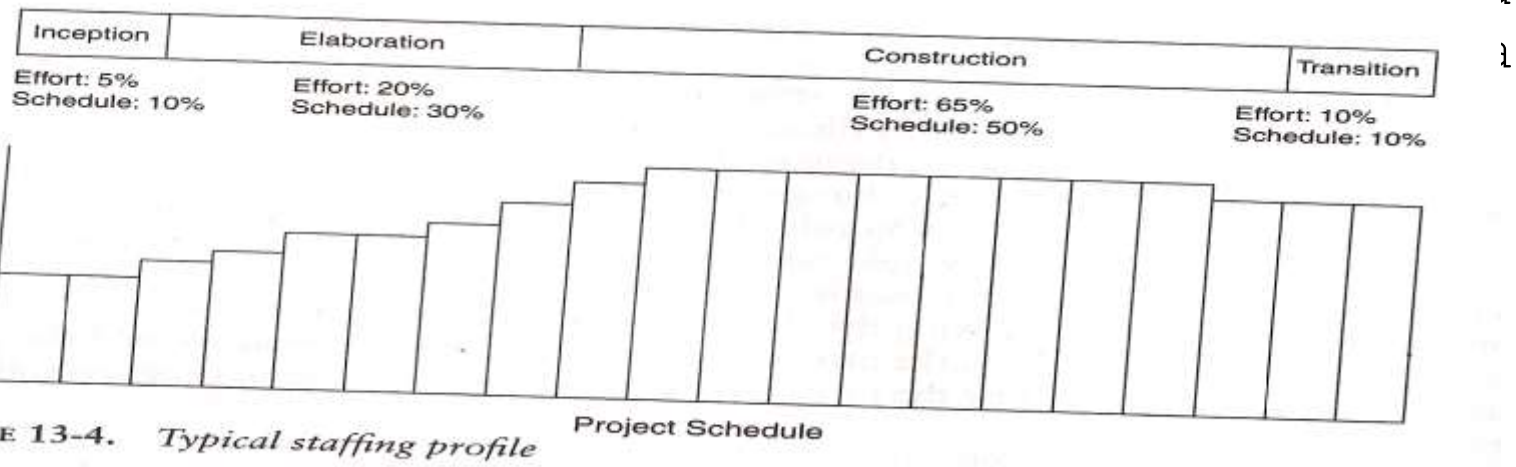


FIGURE 13-4. *Typical staffing profile*

5.3 QUALITY INDICATORS

- The four quality indicators are based primarily on the measurement of software change across evolving baselines of engineering data (such as design models and source code).

5.3.1 CHANGE TRAFFIC AND STABILITY

- Overall change traffic is one specific indicator of progress and quality. Change traffic is defined as the number of software change orders opened and closed over the life cycle (Figure 13-5). This metric can be collected by change type, by release, across all releases, by team, by components, by subsystem, and so fort
  betw

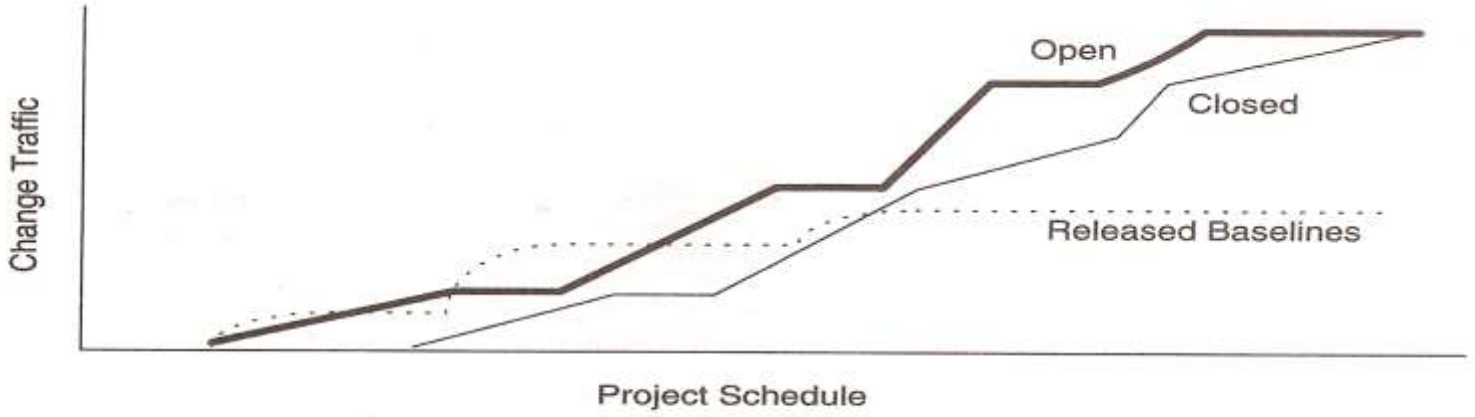


FIGURE 13-5. *Stability expectation over a healthy project's life cycle*

5.3.2 BREAKAGE AND MODULARITY

- Breakage is defined as the average extent of change, which is the amount of software baseline that needs rework (in SLOC, function points, components, subsystems, files, etc). Modularity is the average breakage trend over time [illegible] end expe[illegible] 13-6).

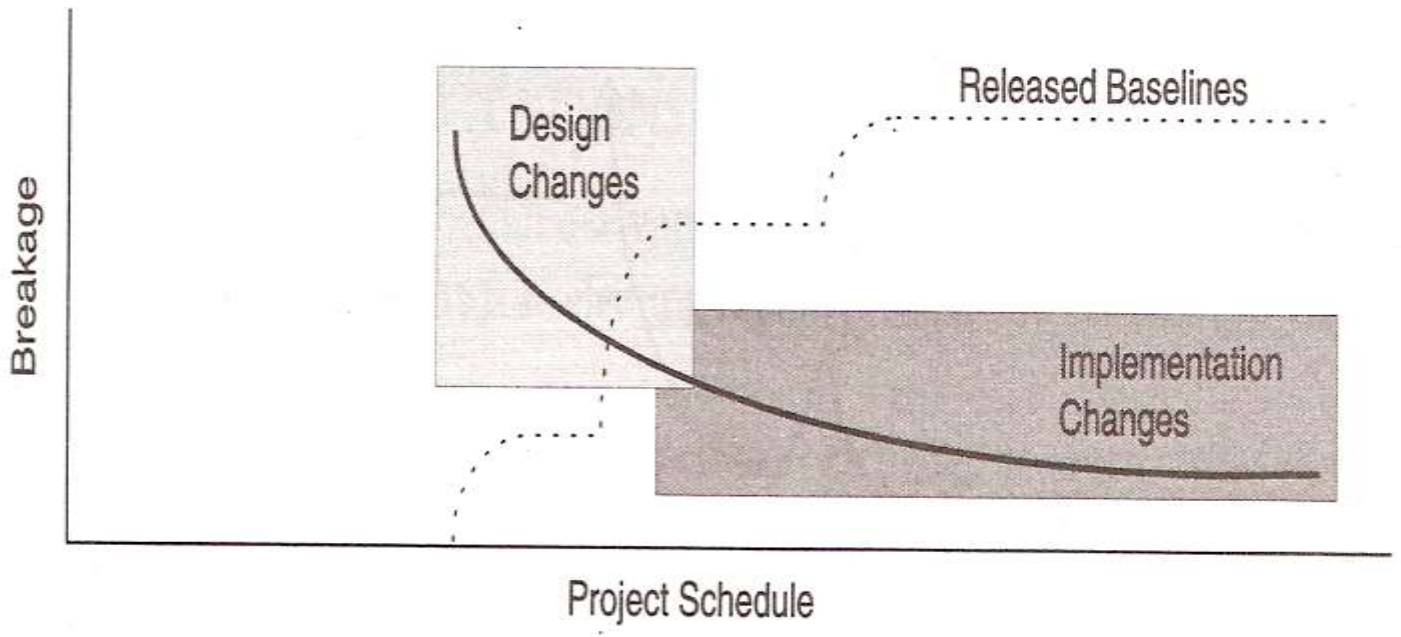


FIGURE 13-6. *Modularity expectation over a healthy project's life cycle*

5.3.3 REWORK AND ADAPTABILITY

- Rework is defined as the average cost of change, which is the effort to analyze, resolve and retest all changes to software baselines. Adaptability is defined as the rework trend over time. For a health project, the trend expectation is decreasing or stable (Figure 13-7).

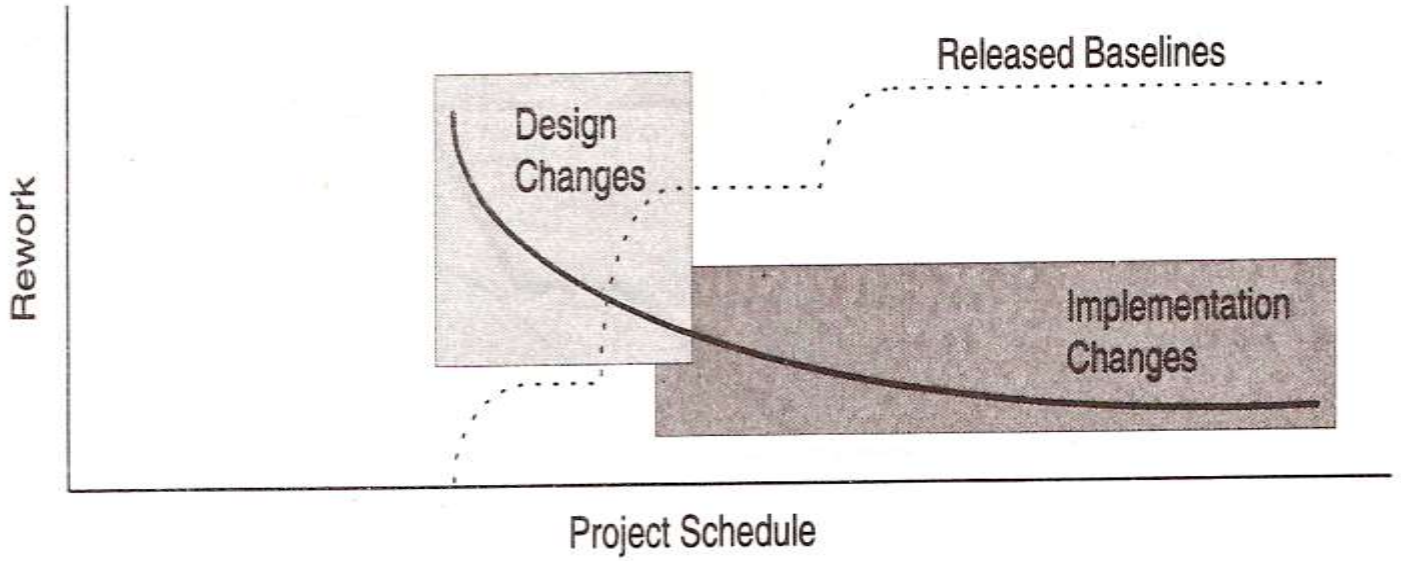
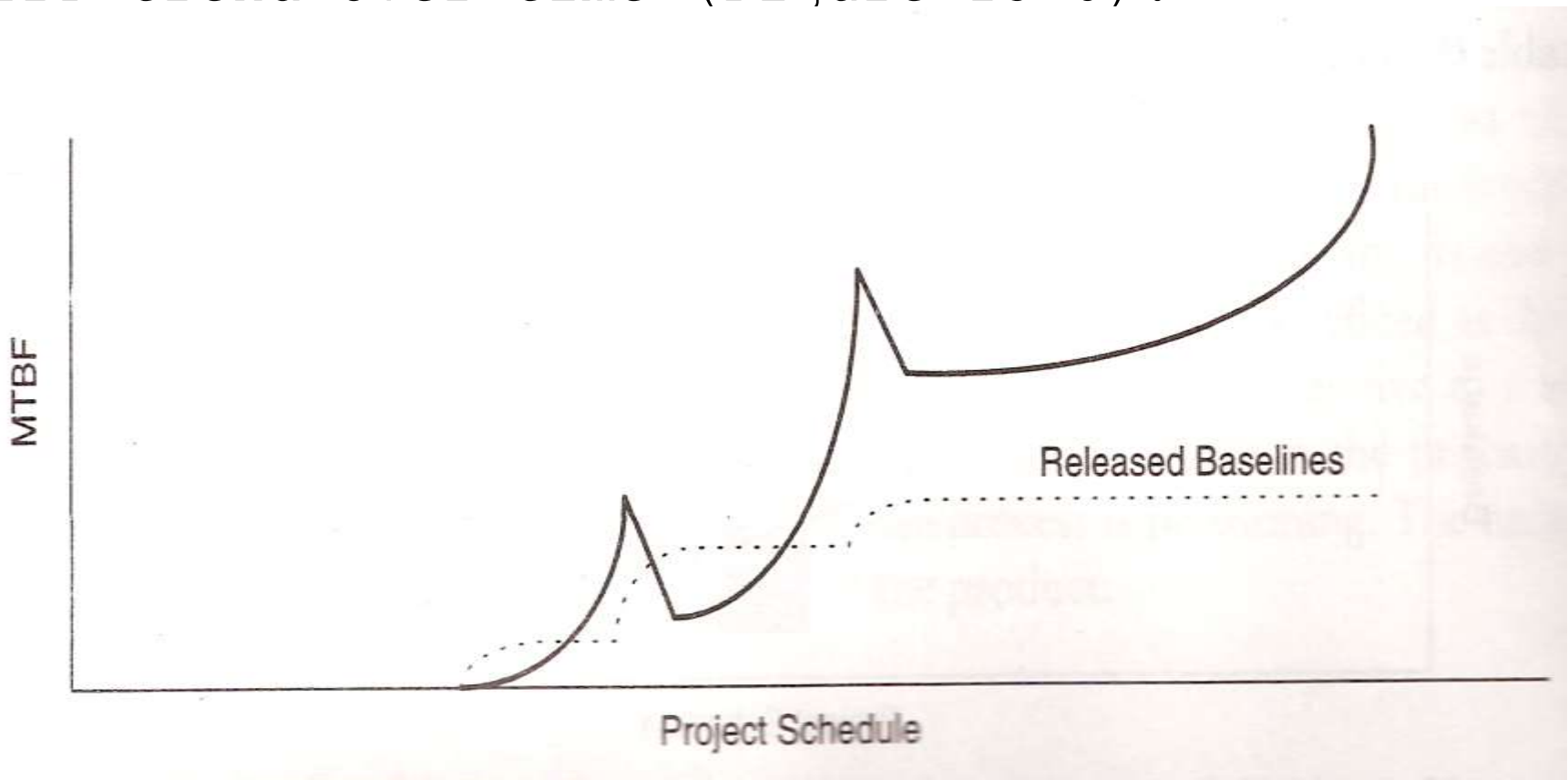


FIGURE 13-7. *Adaptability expectation over a healthy project's life cycle*

5.3.4 MTBF AND MATURITY

- MTBF is the average usage time between software faults. In rough terms, MTBF is computed by dividing the test hours by the number of type 0 and type 1 SCOs. Maturity is defined as the MTBF trend over time (Figure 13-8).



5.4 LIFE-CYCLE EXPECTATIONS

- There is no mathematical or formal derivation for using the seven core metrics. However, there were specific reasons for selecting them:
  - The quality indicators are derived form the evolving product rather than from the artifacts.
  - They provide insight into the waster generated by the process. Scrap and rework metrics are a standard measurement perspective of most manufacturing processes.
  - They recognize the inherently dynamic nature of an iterative development process. 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 tangible indicators for management action.
- The actual values of these metrics can vary widely across projects, organizations and domains. The relative trends across the

Metric	Inception	Elaboration	Construction	Transition
Progress Architecture Applications	5% 30% <5%	25% 90% 20%	90% 100% 85%	100% 100% 100%
Expenditures Effort Schedule	Low 5% 10%	Moderate 25% 40%	High 90% 90%	High 100% 100%
Staffing	Small team	Ramp up	Steady	Varying
Stability Architecture Applications	Volatile Volatile Volatile	Moderate Moderate Volatile	Moderate Stable Moderate	Stable Stable Stable
Modularity Architecture Applications	50% - 100% >50% >80%	25%-50% >50% >80%	<25% <15% <25%	5%-10% <5% <10%
Adaptability Architecture Applications	Varying Varying Varying	Varying Moderate Varying	Benign Benign Moderate	Benign Benign Benign
Maturity Architecture Applications	Prototype Prototype Prototype	Fragile Usable Fragile	Usable Robust Usable	Robust Robust Robust

5.5 PRAGMATIC SOFTWARE METRICS

- Measuring is useful, but it doesn't do any thinking for the decision makers. It only provides data to help them ask the right questions, understand the context, and make objective decisions.
- The basic characteristics of a good metric are as follows:

1. It is considered meaningful by the customer, manager and performer. Customers come to software engineering providers because the providers are more expert than they are at developing and managing software. Customers will accept metrics that are demonstrated to be meaningful to the developer.
2. It demonstrates quantifiable correlation between process perturbations and business performance. The only real organizational goals and objectives are financial: cost reduction, revenue increase and margin

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, requirement), which are surprisingly hard to define precisely in the software engineering world.

4. It displays trends: This is an important characteristic. Understanding the change in a metric's value with respect to time, subsequent projects, subsequent releases, and so forth is an extremely important perspective, especially for today's iterative development models. It is very rare that a given metric drives the appropriate action directly.

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 by automated tools, in part because software tools require rigorous definitions of the data they process

5.6 METRICS AUTOMATION

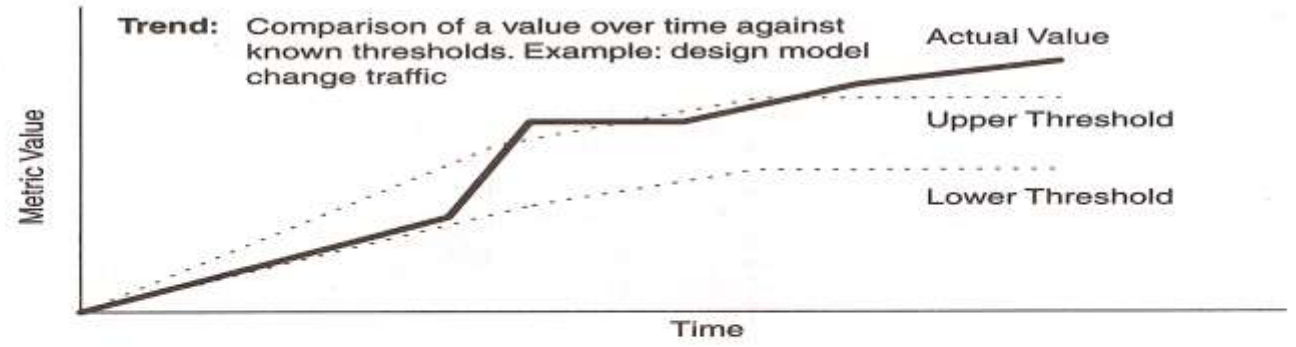
- There are many opportunities to automate the project control activities of a software project. For managing against a plan, a software project control panel (SPCP) that maintains an on-line version of the status of evolving artifacts provides a key advantage.
- To implement a complete SPCP, it is necessary to define and develop the following:
  - Metrics primitives: indicators, trends, comparisons, and progressions.
  - A graphical user interface: GUI support for a software project manager role and flexibility to support other roles
  - Metric collection agents: data extraction from the environment tools that maintain the engineering notations .for the various artifact sets

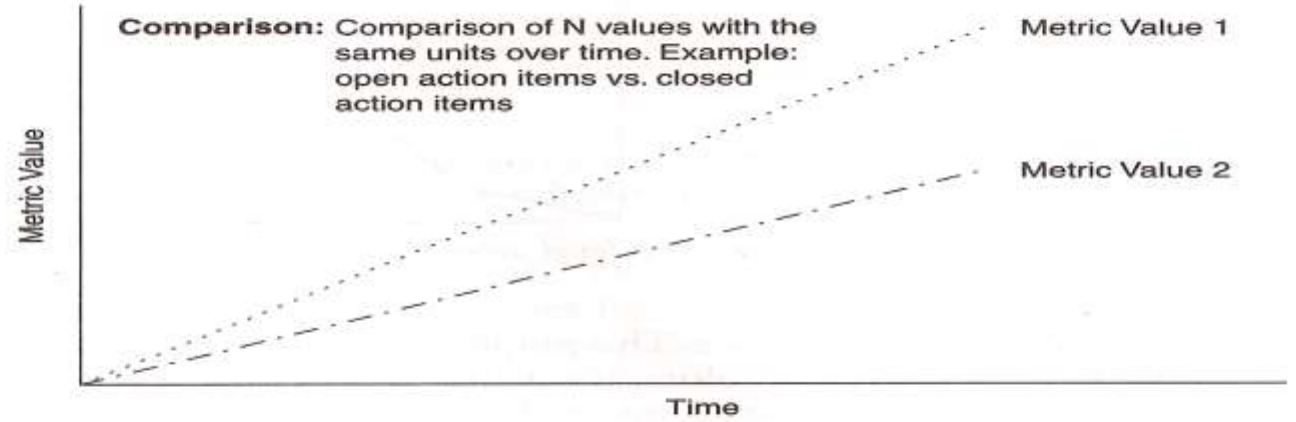
- Metrics data management server: data management support for populating the metric displays of the GUI and storing the data extracted by the agents.
- Metrics definitions: actual metrics presentations for requirements progress (extracted from requirements set artifacts), design progress (extracted from 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

- Specific monitors (called roles) include software project managers, software development team leads, software architects, and customers.

- Monitor: defines panel layouts from existing mechanisms, graphical objects, and linkages to project data; queries data to be displayed at different levels of abstraction
- Administrator: installs the system; defines new mechanisms, graphical objects, and linkages; archiving functions; defines composition and decomposition structures for displaying multiple levels of abstraction.

- Trends support user-selected time increments (such as day, week, month, quarter, year). A comparison graph presents multiple values together, over time. Convergence or divergence among values may be linked to an indicator. A progression graph presents percent complete, where elements of progress are shown as transitions between states and an earned value is associated with each state. Trends, comparisons and progressions are illustrated in Figure 13-9.





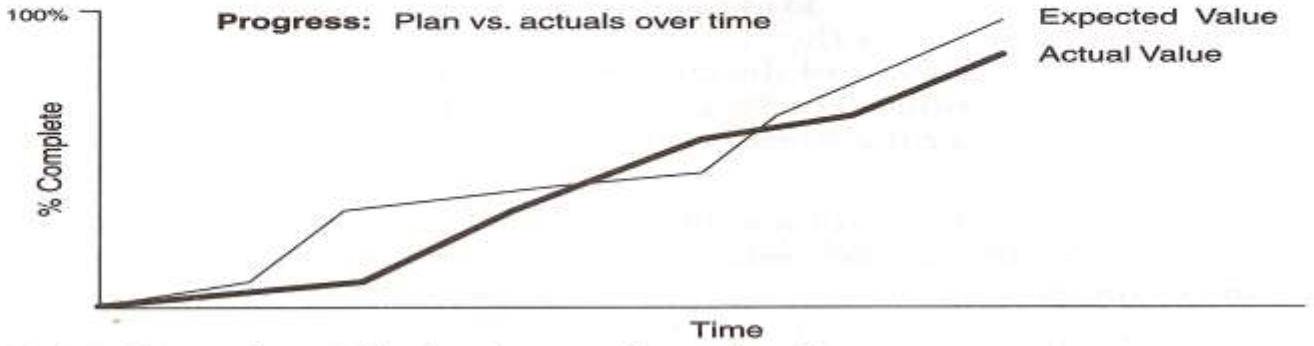


FIGURE 13-9. *Examples of the fundamental metrics classes*

- Figure 13-10 illustrates a simple example of an SPCP for a project. In this case, the software project manager role has defined a top-level display with four graphical objects.

1. **Project activity Status:** the graphical object 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 earned value 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. This graphical object provides several examples of indicators: tertiary colors, the actual percentage, and the current first derivative (up arrow means getting better, down arrow means getting worse).
2. **Technical artifact status:** the graphical object in the upper right provides an overview of the status of the evolving technical artifacts. The Req light would display an assessment of the current state of the use case models and 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 and provides indicators of the current values.

4. **Action item progress:** the graphical object in the lower right provides a different perspective of progress, showing the current number of open and close issues.

- Figure 13-10 in one example of a progress metric implementation.

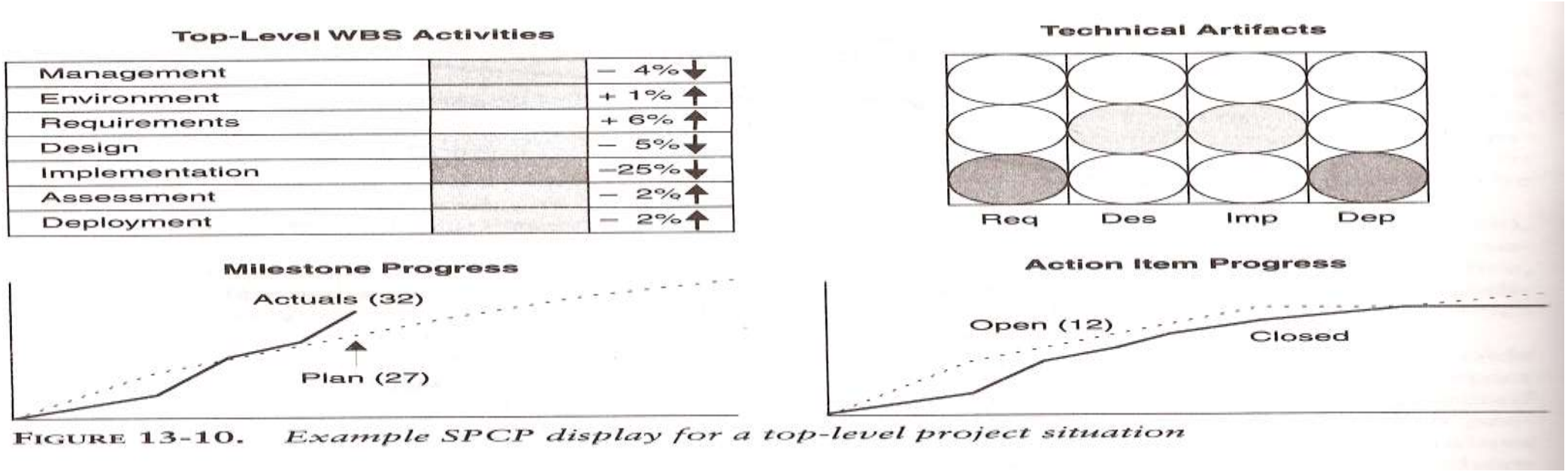


FIGURE 13-10. *Example SPCP display for a top-level project situation*

- The following top-level use case, which describes the basic operational concept of an SPCP, corresponds to a monitor interacting with the control panel:
  - **Start the SPCP.** The SPCP starts and shows the most current information that was saved when the user last used the SPCP.
  - Select a panel preference. The user selects from a list of previously defined default panel preference. The SPCP displays the preference selected.
  - **Select a value or graph metric.** The user selects whether the metric should be displayed for a given point in time or in a graph, as a trend. The default for trends is monthly.
  - **Select to superimpose controls.** The user points to a graphical object and requests that the control values for that metric and point in time be displayed.
  - **Drill down to trend.** The user points to a graphical object displaying a point in time and drills down to view the trend for the metric.
  - **Drill down to point in time.** The user points to a graphical object displaying a trend and drills down to view the values for the metric.
  - **Drill down to lower levels of information.** The user points to a graphical object displaying a point in time and drills down to view the next level of information.
  - **Drill down to lower level of indicators.** The user points to a graphical object displaying an indicator and drills down to view the breakdown of the next level of indicators.
- The SPCP is one example of a metrics automation approach that collects, organizes and reports values and trends extracted directly form the evolving engineering artifacts

5.7 PROCESS DISCRIMINATES

- In tailoring the management process to a specific domain or project, there are tow dimensions of discriminating factors: technical complexity and management complexity. Figure 14-1 illustrates discriminating these tow dimensions of process variability and shows some example project applications. The formality of reviews, the quality control of artifacts, the priorities f concerns and numerous other process instantiation parameters are governed by the point a project occupies in these two dimensions. Figure 14-2 summarizes the different priorities along the two dimensions.

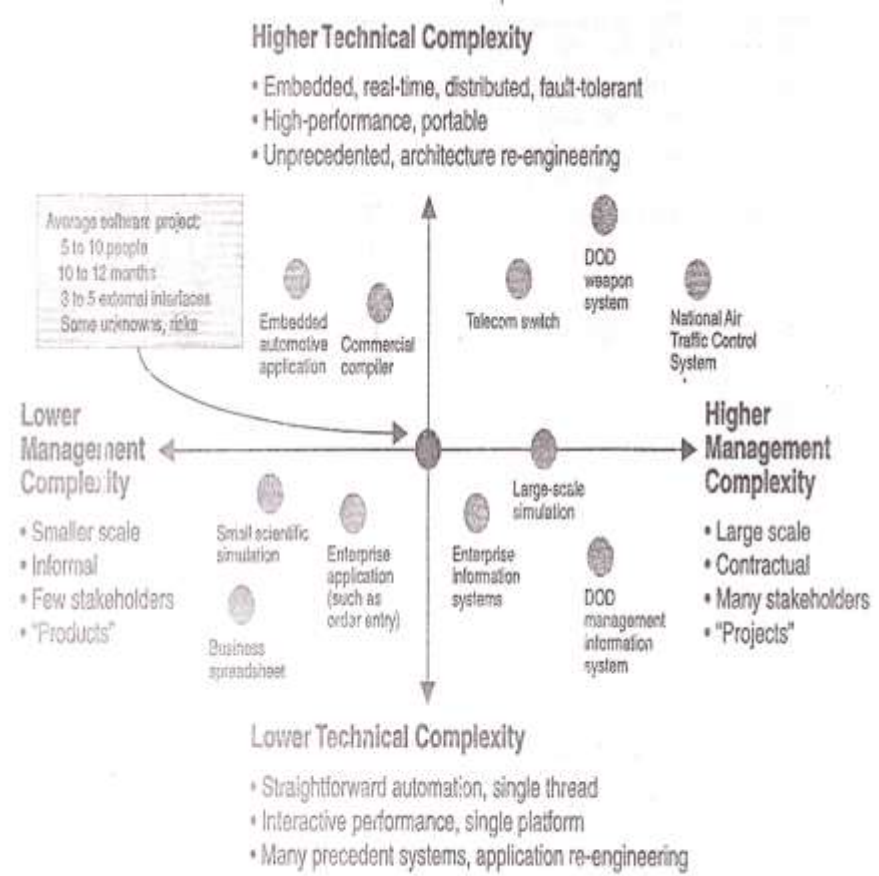


FIGURE 14-1. *The two primary dimensions of process variability*

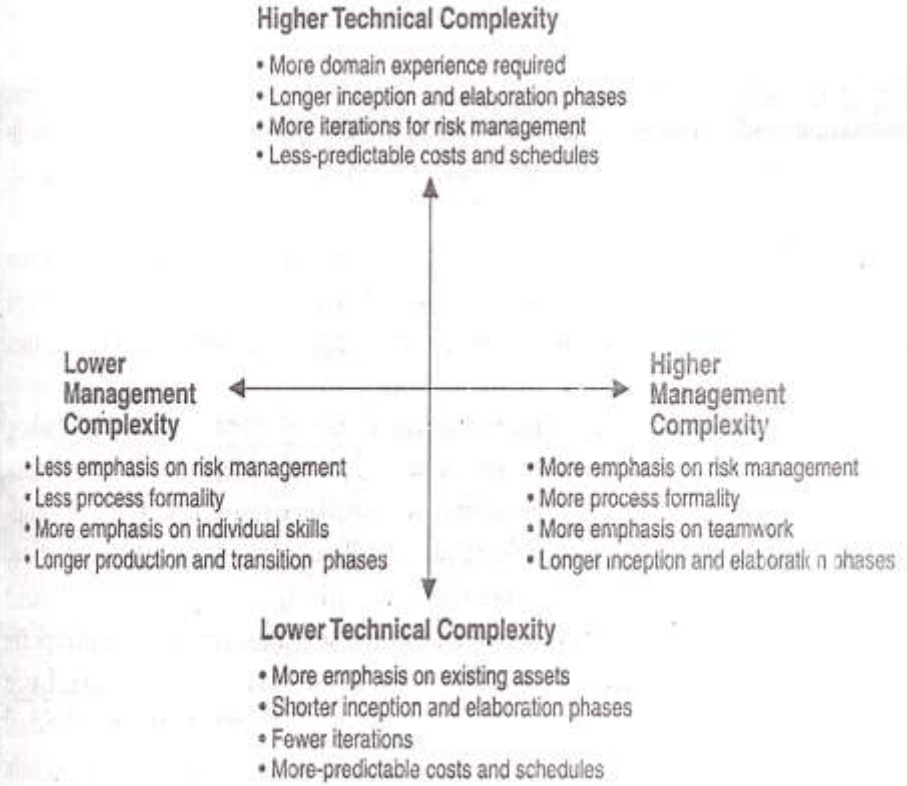


FIGURE 14-2. *Priorities for tailoring the process framework*

5.7.1 SCALE

- There are many ways to measure scale, including number of source lines of code, number of function points, number of use cases, and number of dollars. From a process tailoring perspective, the primary measure of scale is the size of the team. As the headcount increases, the importance of consistent interpersonal communications becomes paramount. Otherwise, the diseconomies of scale can have a serious impact on achievement of the project objectives.
- A team of 1 (trivial), a team of 5 (small), a team of 25 (moderate), a team of 125 (large), a team of 625 (huge), and so on. As team size grows, a new level of personnel management ins introduced at roughly each factor of 5. This model can be sued to describe some of the process differences among projects of different sizes.
- Trivial-sized projects require almost no management overhead (planning, communication, coordination, progress assessment, review, administration).

- Small projects (5 people) require very little management overhead, but team leadership toward a common objective is crucial. There is some need to communicate the intermediate artifacts among team member.
- Moderate-sized projects (25 people) require moderate management overhead, including a dedicated software project manager to synchronize team workflows and balance resources.
- Large projects (125 people) require substantial management overhead including a dedicated software project manager and several subproject managers to synchronize project-level and subproject-level workflows and to balance resources,
- Project performance is dependent on average people, for two reasons:
  1. There are numerous mundane jobs in any large project, especially in the overhead workflows.
  2. The probability of recruiting, maintaining and retaining a large umber of exceptional people is small.
- Huge projects (625 people) require substantial management overhead, including multiple software project managers an many subproject managers to

PROCESS PRIMITIVE	SMALLER TEAM	LARGER TEAM
Life –cycle phases	Weak boundaries between phases	Well-defined phase transitions to synchronize progress among concurrent activities
Artifacts	Focus on technical artifacts few discrete baselines Very few management artifacts required	Change management of technical artifacts, which may result in numerous baselines Management artifacts important
Workflow effort allocations	More need for generalists, people who perform roles in multiple workflows	Higher percentage of specialists more people and teams focused on a specific workflow
Checkpoints	Many informal events for maintaining technical consistency No schedule disruption	A few formal events synchronization among teams, which can take days
Management discipline	Informal planning, project control and organization	Formal planning, project control and organization
Automation discipline	More ad hoc environments, managed by individuals	Infrastructure to ensure a consistent, up-to-date environment available across all teams Additional tool integration to support project control and change control.

5.7.2 STAKEHOLDER COHESION OR CONTENTION

- The degree of cooperation and coordination among stakeholders (buyers, developers, users, subcontractors and maintainers, among others) can significantly drive the specifies of how a process is defined. This process parameter can range from cohesive to adversarial. Cohesive teams have common goals, complementary skills and close communications. Adversarial teams have conflicting goals, completing or incomplete skills and less-than-open communications.
- Table 14-2 Summarizes key difference s in the process primitives for varying levels of stakeholder cohesion.

PROCESS PRIMITIVE	FEW STAKEHOLDERS, COHESIVE TEAMS	MULTIPLE STAKEHOLDERS, ADVERSARIAL RELATIONSHIPS
Life-cycle phases	Weak boundaries between phases	Well-defined phase transitions to synchronize progress among concurrent activities
Artifacts	Fewer and less detailed management artifacts required	Management artifacts Paramount, especially the business case, vision and status assessment
Workflow effort allocations	Less overhead in assessment	High assessment overhead to ensure stakeholder concurrence
Checkpoints	Many informal events	3 or 4 formal events many informal technical walkthroughs necessary to synchronize technical decisions. Synchronization among stakeholder teams, which c an impede progress for weeks.
Management discipline	Informal planning, project control and organization	Formal planning, project control and organization
Automation discipline	(insignificant)	On-line stakeholder environments necessary

5.7.3 PROCESS FLEXIBILITY OR RIGOR

- The degree of rigor, formality and change freedom inherent in a specific project's "contract" (vision document, business case and development plan) will have a substantial impact on the implementation of the project's process. For very loose contracts such as building a commercial product within a business unit of a software company (such as a Microsoft application or a rational software corporation development tool), management complexity is minimal. In these sorts of development processes, feature set, time to market, budget and quality can all be freely traded off and changed with very little overhead.
- Table 14-3 summarized key difference sin the process primitives for varying levels of process flexibility.

PROCESS PRIMITVIE	FLEXIBLE PROESS	INFLEXICBLE PROCESS
Life-cycle phases	Tolerant of cavalier phase commitments	More credible basis required for inception phase commitments
Artifacts	Changeable business case and vision	Carefully controlled changes to business case and vision
Workflow effort allocations	(insignificant)	Increased levels of management and assessment workflows
Checkpoints	Many informal events for maintaining technical consistency	3 or 4 formal events synchronization among stakeholder teams, which can impede progress for days or weeks
Management discipline	(insignificant)	More fidelity required for planning and project control
Automation discipline	(insignificant)	(insignificant)

The process maturity level of the development organization, as defined by the software engineering Institute's capability maturity model is another key driver of management complexity. Managing a mature process (level 3 or higher) is far simpler than managing an immature process (level 1 and 2). Organizations with a mature process typically have a high level of precedent experience in developing software and a high level of existing process collateral that enables predictable planning and execution of the process. Tailoring a mature organization's process for a specific project is generally a straight forward task. Table 14-4 summarizes key difference in the process primitives for varying levels of process maturity.

TABLE 14-4: *Process discriminators that result form differences in process maturity*

PROCESS PRIMITIVE	MATURE, LEVEL 3 OR 4 ORGANIZATIONS	LEVEL 1 ORGANIZATION
Life-cycle phases	Well-established criteria for phase transitions.	(insignificant)
Artifacts	Well-established format, content and production methods.	Free-form
Workflow effort allocations	Well-established basis	No basis
Checkpoints	Well-defined combination of formal and informal events	(insignificant)
Management discipline	Predictable planning objective status assessments	Informal planning and project control
Automation discipline	Requires high levels of automation for round-trip engineering, change management and process instrumentation	Little automation or disconnected islands of automation

5.7.5 ARCHITECTURAL RISK

- The degree of technical feasibility demonstrated before commitment to full-scale production is an important dimension of defining a specific project's process. There are many sources of architectural risk. Some of the most important and recurring sources are system performance (resource utilization, response time, throughput, accuracy), robustness to change (addition of new features, incorporation of new technology, adaptation to dynamic operational conditions) and system reliability (predictable behavior, fault tolerance). The degree to which these risks can bed eliminated before construction begins can have dramatic ramifications in the process tailoring. Table 14-5 summarizes key difference in the process primitives for varying levels of architectural risk.

PROCESS PRIMITIVE	COMPLETE ARCHITECTURE FEASIBILITY DEMONSTRATION	NO ARCHITECTURE FEASIBILITY DEMONSTRATION
Life-cycle phases	More inception and elaboration phase iterations	Fewer early iterations More construction iterations
Artifacts	Earlier breadth and depth across technical artifacts	(insignificant)
Workflow effort allocations	Higher level of design effort Lower levels of implementation and assessment	Higher levels of implementation and assessment to deal with increased scrap and rework.
Checkpoints	More emphasis on executable demonstrations	More emphasis on briefings, documents and simulations
Management discipline	(insignificant)	(insignificant)
Automation discipline	More environment resources required earlier in the life cycle	Less environment demand early in the life cycle.

5.7.6 DOMAIN EXPERIENCE

- The development organization's domain experience governs its ability to converge on an acceptable architecture in a minimum number of iterations. An organization that has built five generations of radar control switches may be able to converge on adequate baseline architecture for a new radar application in two or three prototype release iterations. A skilled software organization building its first radar application may require four or five prototype releases before converging on an adequate baseline.
- Table 14-6 summarizes key differences in the process primitives for varying levels of domain experience.
- **TABLE 14-6: *Process discriminators that result from differences in domain experience***

PROCESS PRIMITIVE	EXPERIENCED TEAM	INEXPERIENCED TEAM
Life-cycle phases	Shorter engineering Stage	Longer engineering stage
Artifacts	Less scrap and rework in requirements and design sets	More scrap and rework in requirements and design sets
Workflow effort allocations	Lower levels of requirements and design	Higher levels of requirements and design
Checkpoints	(insignificant)	(insignificant)
Management discipline	Less emphasis on risk management Less-frequent status assessments needed	More-frequent status assessments required
Automation discipline	(insignificant)	(insignificant)

5.8 EXAMPLE: SMALL-SCALE PROJECT VERSUS LARGE-SCALE PROJECT

- An analysis of the differences between the phases, workflows and artifacts of two projects on opposite ends of the management complexity spectrum shows how different two software project processes can be Table 14-7 illustrates the differences in schedule distribution for large and small project across the life-cycle phases. A small commercial project (for example, a 50,000 source-line visual basic windows application, built by a team of five) may require only 1 month of inception, 2 months of elaboration, 5 months of construction and 2 months of transition. A large, complex project (for example, a 300,000 source-line embedded avionics program, built by a team of 40) could require 8 months of inception, 14 months of elaboration, 20 months of construction, and 8 months of transition. Comparing the ratios of the life cycle spend in each phase highlights the obvious differences.
- One key aspect of the differences between the two projects is the leverage of the various process

- The following list elaborates some of the key differences in discriminators of success.
  - Design is key in both domains. Good design of a commercial product is a key differentiator in the marketplace and is the foundation for efficient new product releases. Good design of a large, complex project is the foundation for predictable, cost-efficient construction.
  - Management is paramount in large projects, where the consequences of planning errors, resource allocation errors, inconsistent stakeholder expectations and other out-of-balance factors can have catastrophic consequences for the overall team dynamics. Management is far less important in a small team, where opportunities for miscommunications are fewer and their consequences less significant.
  - Deployment plays a far greater role for a small commercial product because there is a broad user base of diverse individuals and environments.

ENGINEERING			PRODUCTION	
DOMAIN	**INCEPTION**	**ELABORATION**	**CONSTRUCTION**	**TRANSITION**
Small commercial project	10%	20%	50%	2%
Large, complex Project	15%	30%	40%	15%

RANK	SMALL COMMERCIAL PROJECT	LARGE, COMPLEX PROJECT
1	Design	Management
2	Implementation	Design
3	Deployment	Requirements
4	Requirements	Assessment
5	Assessment	Environment
6	Management	Implementation
7	Environment	Deployment

ARTIFACT	SMALL COMMERCIAL PROJECT	LARGE, COMPLEX PROJECT
Work breakdown structure	1-page spreadsheet with 2 levels of WBS elements	Financial management system with 5 or6 levels of WBS elements.
Business case	Spreadsheet and short memo	3-volumen proposal including technical volume, cost volume and related experience.
Vision statement	10-page concept paper	200-page subsystem specification
Development Plan	10-page plan	200-page development plan.
Release specifications and number of releases	3 interim release specifications	8 to 10 interim release specifications
Architecture description	5 critical use cases, 50 UML diagrams, 20 pages of text, other graphics	25 critical sue cases, 200 UML diagrams, 100 pages of text, other graphics
Software	50,000 lines of visual basic code	300,000 lines of C++ code.
Release description	10-page release notes	100-page summary
Deployment	User training course sales rollout kit	Transition plan installation plan
Use manual	On-line help and 100-page user manual	200- page user manual
Status assessment	Quarterly project reviews	Monthly project management reviews.

MODEL QUESTIONS

1. What is the need for metrics? What do you mean by indicators?
2. List the seven core metrics, their purpose and perspectives.
3. Identify examples of each of the seven core metrics and state their purpose.
4. Why are the metrics divided into management and quality indicators? Name the core metrics under each category.
5. List the attributes of seven core metrics.
6. Discuss in detail about the three fundamental sets of management metrics.
7. Write notes on any (two).
    1. Earned value analysis.
    2. Backup plan.
    3. GANTT chart.
8. Explain in detail about the four quality indicators in project control.
9. Define the following terms.
    1. Change Traffic
    2. Stability
    3. Breakage
    4. Modularity

11. Give the reasons fro selecting the seven core metrics in the software life cycle. Also discuss the evolutionary pattern of life cycle metrics.
12. List and explain the basic characteristics of a good metric.
13. 'Reasoning is required to interpret some of the situations correctly'. Justify your answer.
14. Explain briefly about the metrics automation.
15. Explain the significance of Software Project Control Panel (SPCP) in Metrics Automation.
16. What are the two primary dimensions of process variability? Explain.
17. "The size of a project is an important concern in assessing the management overhead". Comment on this statement.
18. Write short notes on stakeholder cohesion or contention.
19. Write short notes on domain experience.
20. Define the SEI-CMM maturity levels of organizations. How do processes differ because of process flexibility and process maturity?
21. What are the W5H questions to be answered for a software measure?
22. Name metrics for reliability. SW cost, effort, SW complexity with examples.

Thank You..