UNIT - IV

Project Organizations
Line-of-business organizations, project organizations, evolution of organizations, process automation.
Project Control and process instrumentation
The seven-core metrics, management indicators, quality indicators, life-cycle expectations, Pragmatic software metrics, metrics automation.

Project organization

A project organization is one, in which a project structure is created as a separate unit or division within a permanent functional structure; drawing specialists and workers from various functional departments who work under the overall leadership, control and co-ordination of a project manager to complete projects of a technical and costly nature.

Each project has its unique characteristics and the design of an organizational structure should consider the organizational environment, the project characteristics in which it will operate, and the level of authority the project manager is given. A project structure can take on various forms with each form having its own advantages and disadvantages.

One of the main objectives of the structure is to reduce uncertainty and confusion that typically occurs at the project initiation phase. The structure defines the relationships among members of the project management and the relationships with the external environment. The structure defines the authority by means of a graphical illustration called an organization chart.

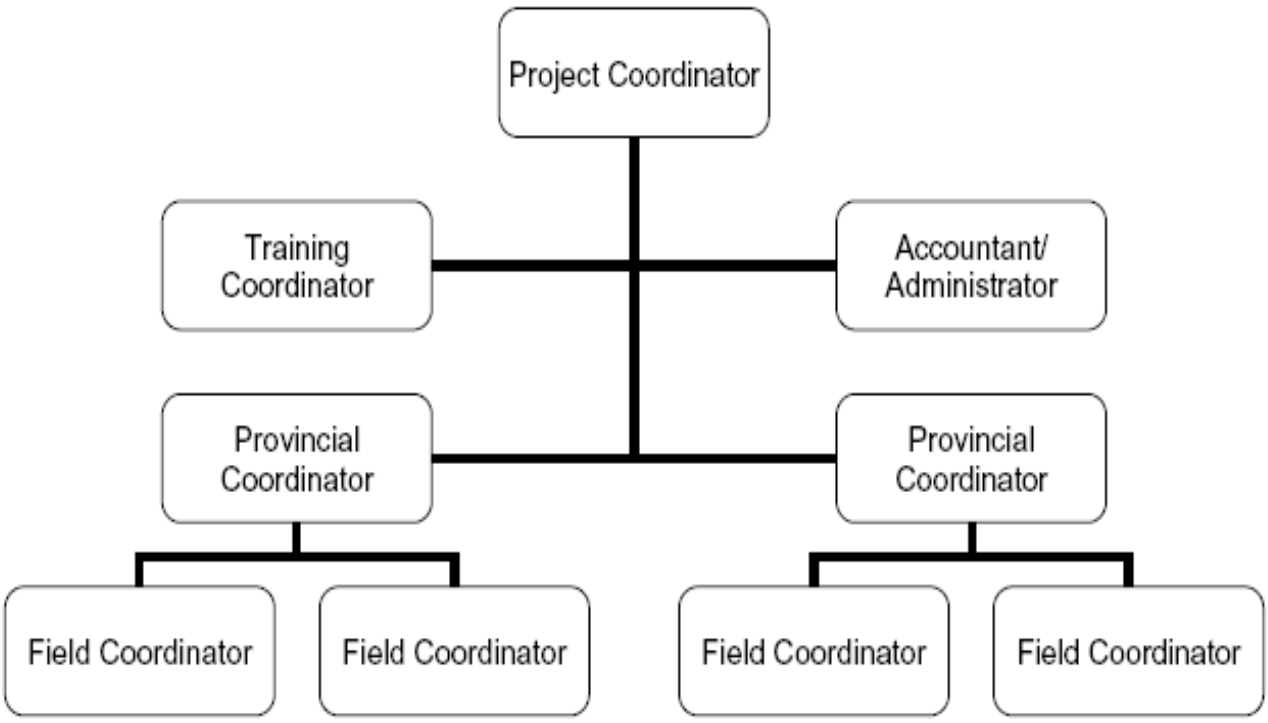


Figure 4-1a: Project Organization Chart

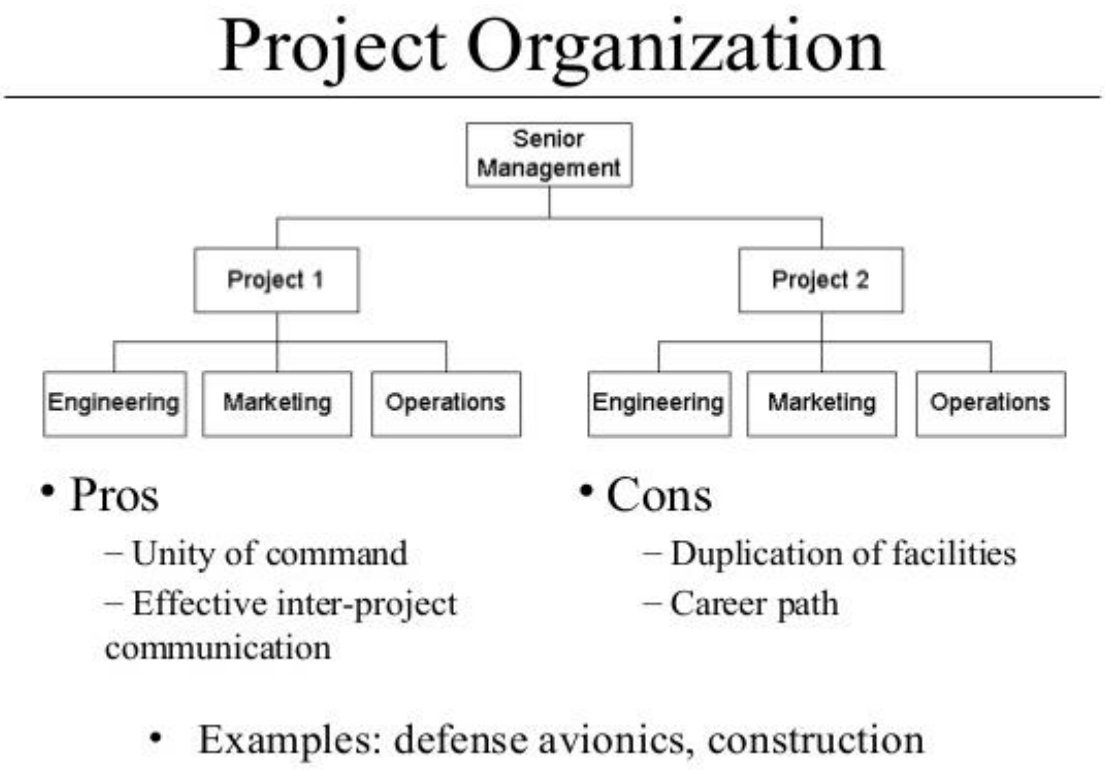


Figure 4-1b: Project Organization Chart

Line-of-business Organizations

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, where process commonality makes sense. For example, the process for developing avionics software is different from the process used to develop office applications.

• Responsibility for process automation is an organizational role and is equal in importance to the process definition role. Projects achieve process commonality primarily through the environment support of common tools.

• Organizational roles may be fulfilled by a single individual or several different teams, depending on the scale of the organization. A 20-person software product company may require only a single person to fulfill all the roles, while a 10,000-person telecommunications company may require hundreds of people to achieve an effective software organization.

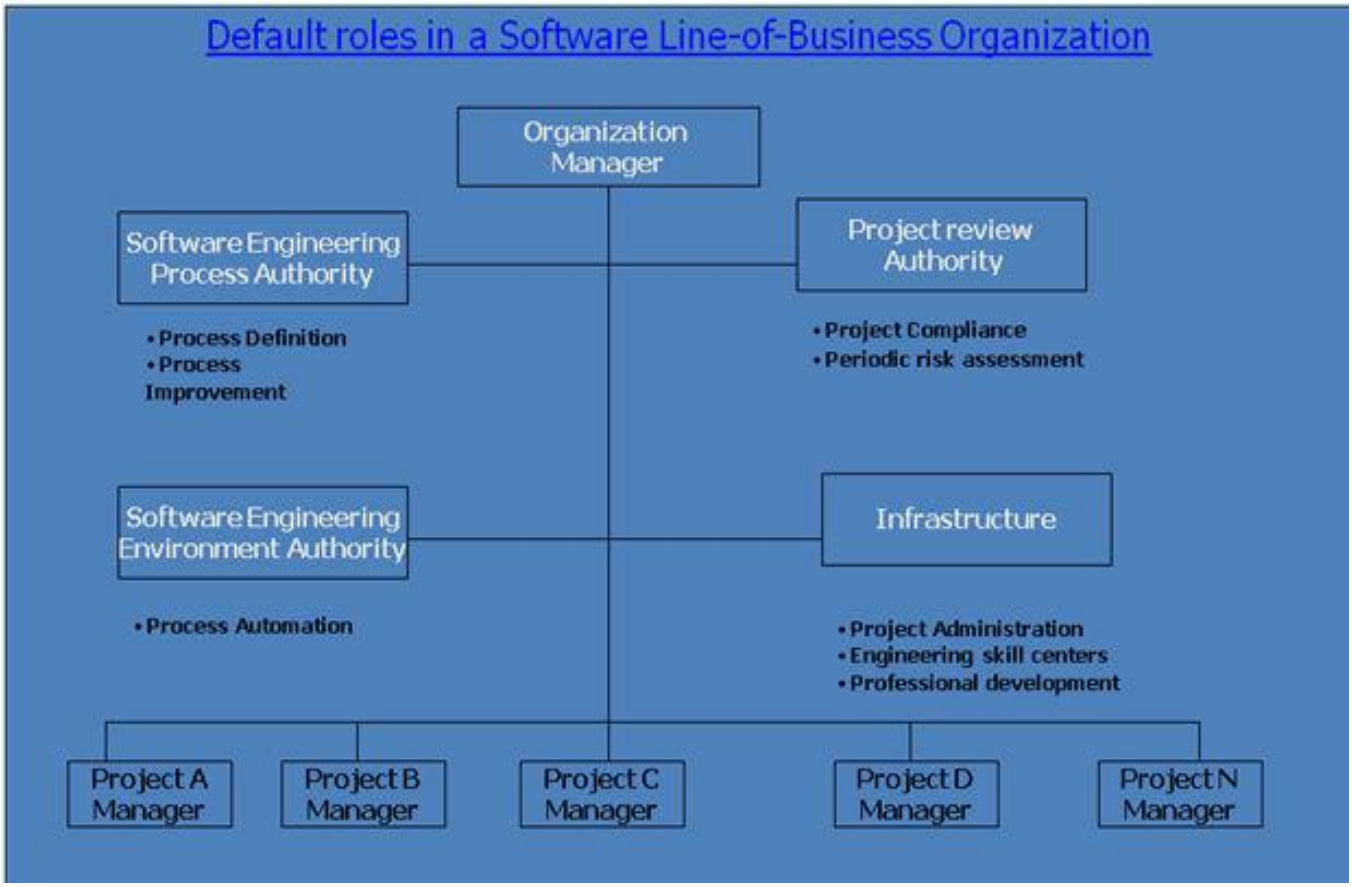


Figure 4-2: Line of Business Organizations

Software Engineering Process Authority

The Software Engineering Process Authority (SEPA) facilitates the exchange of information and process guidance both to and from project practitioners. This role is accountable to the organization general manager for maintaining a current assessment of the organization's process maturity and its plan for future process improvements. The SEPA must help initiate and periodically assess project processes. Catalyzing the capture and dissemination of software best practices can be accomplished only when the SEPA understands both the desired improvement and the project context. The SEPA is a necessary role in any organization. It takes on responsibility and accountability for the process definition and its maintenance (modification, improvement, technology insertion). The SEPA could be a single individual, the general manager, or even a team of representatives. The SEPA must truly be an authority, competent and powerful, not a staff position rendered impotent by ineffective bureaucracy.

Project Review Authority

The Project Review Authority (PRA) is the single individual responsible for ensuring that a software project complies with all organizational and business unit software policies, practices, and standards. A software project manager is responsible for meeting the requirements of a contract or some other project compliance standard, and is also accountable to the PRA. The PRA reviews both the project's conformance to contractual obligations and the project's organizational policy obligations. The customer monitors contract requirements, contract milestones, contract deliverables, monthly management reviews,

progress, quality, cost, schedule, and risk. The PRA reviews customer commitments as well as adherence to organizational policies, organizational deliverables, financial performance, and other risks and accomplishments.

Software Engineering Environment Authority

The Software Engineering Environment Authority (SEEA) is responsible for automating the organization's process, maintaining the organization's standard environment, training projects to use the environment, and maintaining organization-wide reusable assets. The SEEA role is necessary to achieve a significant return on investment for a common process. Tools, techniques, and training can be amortized effectively across multiple projects only if someone in the organization (the SEEA) is responsible for supporting and administering a standard environment. In many cases, the environment may be augmented, customized, or modified, but the existence of an 80% default solution for each project is critical to achieving institutionalization of the organization's process and a good ROI on capital tool investments.

Infrastructure

An organization's infrastructure provides human resources support, project-independent research and development, and other capital software engineering assets. The infrastructure for any given software line of business can range from trivial to highly entrenched bureaucracies. 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

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

It is important that organization managers treat software development environments just as hardware development environments are treated—namely, as capital equipment. There is resistance to this approach in most small-scale or immature organizations, where specific process development and tooling are included as direct project expenses. For most mature software organizations, the process and tooling should be organizational assets, just as they are in other engineering disciplines. As such, they should be funded with capital resources. Financing models can include absorption into overhead or general and administrative costs, or project billing based on usage. In today's software industry, characterized by ingrained accounting methods, project-funded tooling, and software licensing methods, relatively few organizations have transitioned to such a capital investment model for their software environments. These

organizations tend to be mature, large-scale software developers that have achieved stable process definitions and have established long-term partnerships with software tool vendors.

Project Organizations

Figure shows a default project organization and maps project-level roles and responsibilities. This structure can be tailored to the size and circumstances of the specific project organization.

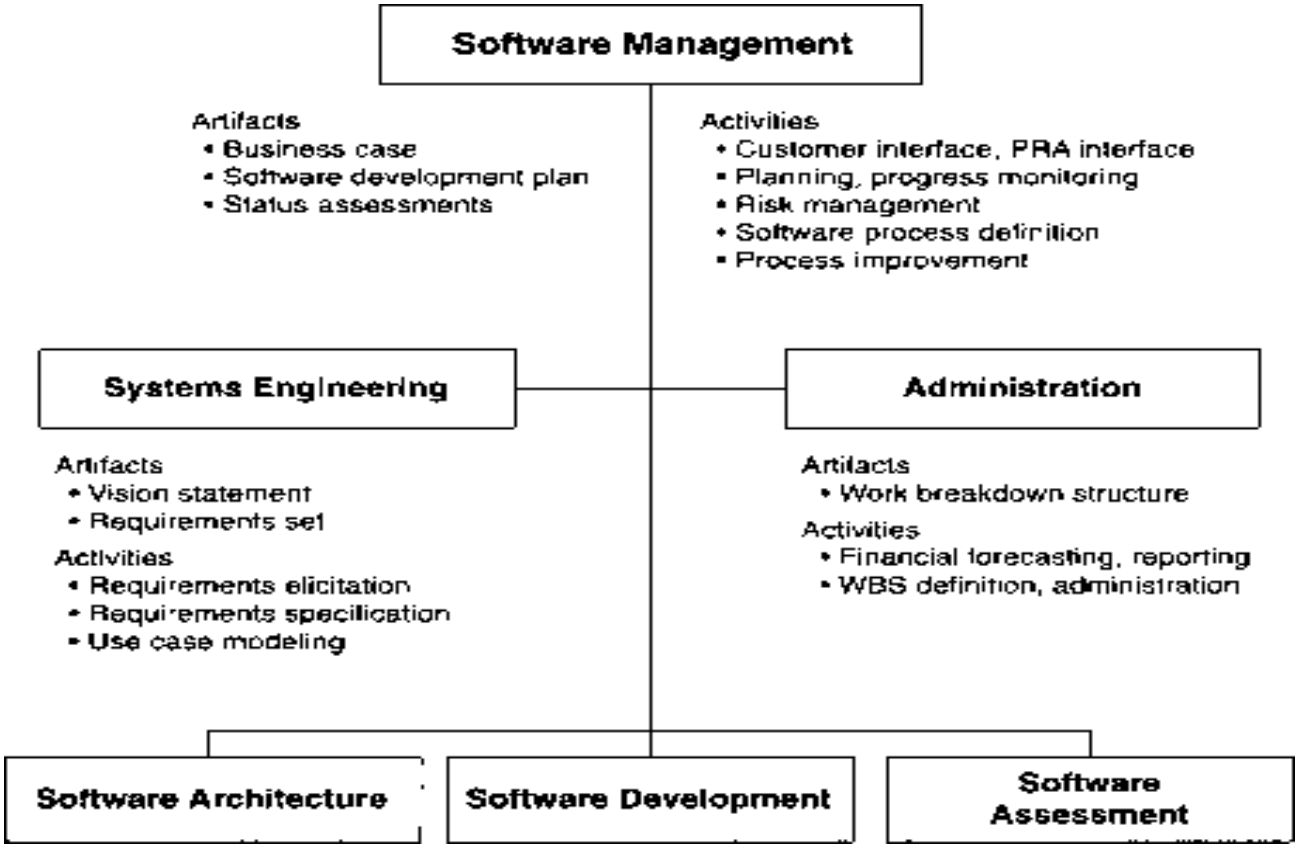


Figure 4-3: Default Project Organization

The main features of the default organization are as follows:

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

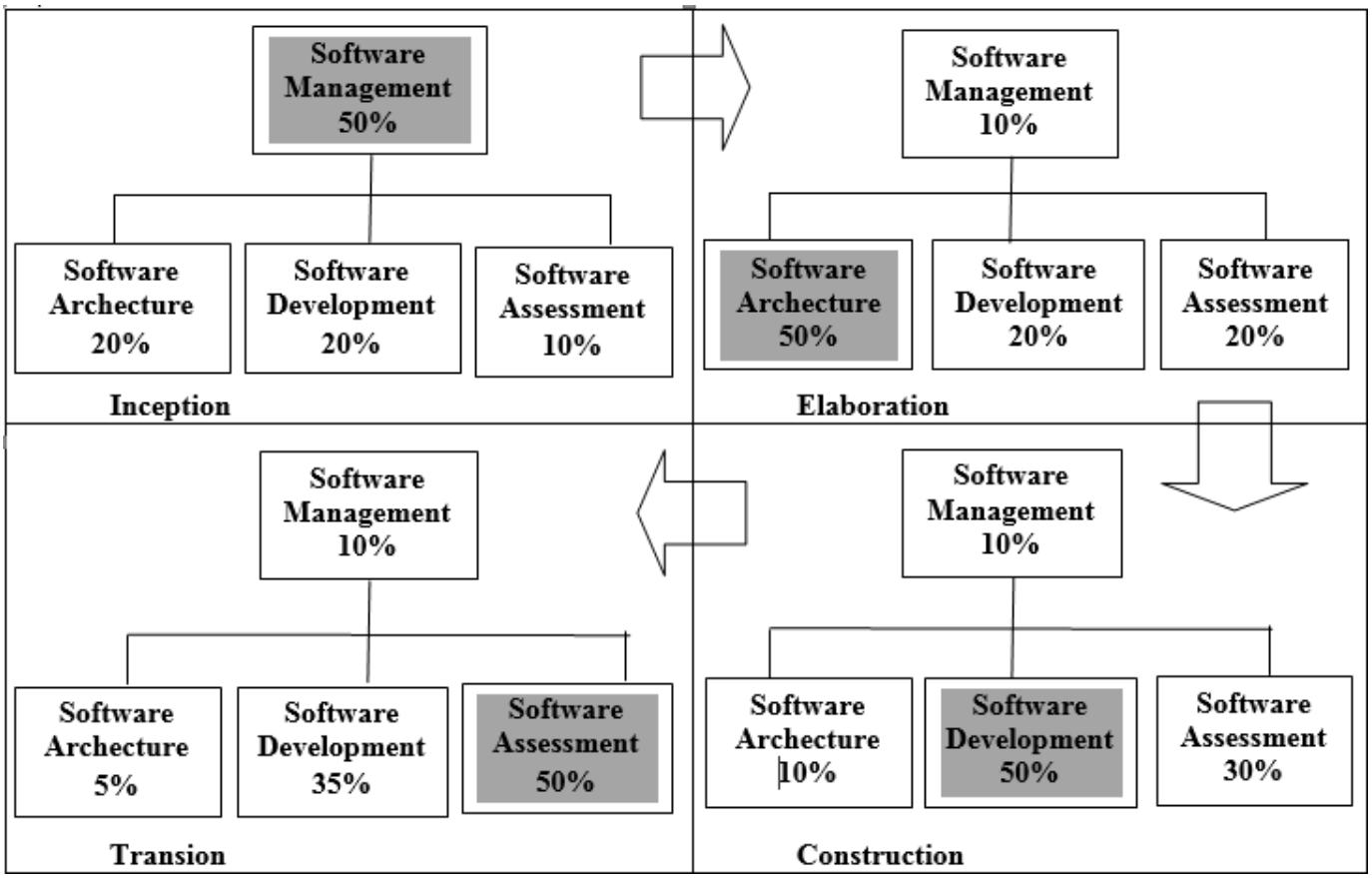
• The development team owns the component construction and maintenance activities. The assessment team is separate from development. This structure fosters an independent quality perspective and focuses a team on testing and product evaluation activities concurrent with on-going development.

• Quality is everyone's job, integrated into all activities and checkpoints. Each team takes responsibility for a different quality perspective.

Project Organizations and Responsibilities

- **Organizations** engaged in software Line-of-Business need to support projects when the infrastructure necessary to use a common process.
- **Project** organizations need to allocate artifacts & responsibilities across project team to ensure a balance of global (architecture) & local (component) concerns.
- **The organization** must evolve when the WBS & Life cycle concerns.
- **Software lines of business & product teams have different motivation.**
- **Software lines of business** are motivated by return of investment (ROI), new business discriminators, market diversification & profitability.
- **Project teams** are motivated by the cost, Schedule & quality of specific deliverables

Evolution of Organizations



Inception: Software management: **50%** Software Architecture: 20% Software development: 20% Software Assessment (measurement/evaluation):10%	**Elaboration:** Software management: 10% Software Architecture: **50%** Software development: 20% Software Assessment (measurement/evaluation):20%
Construction: Software management: 10% Software Architecture: 10% Software development: **50%** Software Assessment (measurement/evaluation):30%	**Transion:** Software management: 10% Software Architecture: 5% Software development: 35% Software Assessment (measurement/evaluation):**50%**

Figure 4-4: Evaluation of Organization

Figure 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

- It is equally important to elaborate the details of subteams, responsibilities, and work packages, but not until the planning details in the WBS are stable. Defining all the details of lower level team structures prematurely can result in serious downstream inefficiencies.

The Process Automation

Automation needs grow depending on the scale of the effort. Just as the construction process varies depending on whether you are building a dollhouse, a single-family home, or a skyscraper, the software process varies across the spectrum from single-person spreadsheet tasks to large-scale, multiple-organization, catastrophic cost-of-failure applications. The techniques, training, time scales, acceptance criteria, and levels of automation differ significantly at opposite ends of the spectrum.

Most software organizations are confronted with the task of integrating their own environment and infrastructure for software development. This process typically results in the selection of more or less incompatible tools that have different information repositories, are supplied by different vendors, work on different platforms, use different jargon, and are based on different process assumptions. Integrating such an infrastructure has proven to be much more problematic than expected.

Key Points the environment must be a first-class artifact of the process.

▲ Process automation, and change management in particular, are critical to an iterative process. If change is too expensive, the development organization will resist it.

▲ Round-trip engineering and integrated environments promote change freedom and effective evolution of technical artifacts.

▲ Metrics automation is crucial to effective project control.

▲ External stakeholders need access to environment resources to improve interaction with the development team and add value to the process.

Automating the development process and establishing an infrastructure for supporting the various project workflows are important activities of the engineering stage of the life cycle. They include the tool selection, custom tool smiting, and process automation necessary to perform against the development plan with acceptable efficiency. Evolving the development environment into the maintenance environment is also crucial to any long-lived software development project.

To complicate matters further, it is rare to find stakeholders who treat the environment as a first-class artifact necessary for the continued maintenance of the product. The environment that provides the process automation is a tangible artifact that is critical to the life-cycle cost of the system being developed. The top-level WBS recommended in Chapter 10 recognizes the environment as a first-class workflow.

Introduced three levels of process. Each level requires a certain degree of process automation for the corresponding process to be carried out efficiently:

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.

While the main focus of process automation is the workflow of a project-level environment, the infrastructure context of the project's parent organization and the tool building blocks are important prerequisites**.**

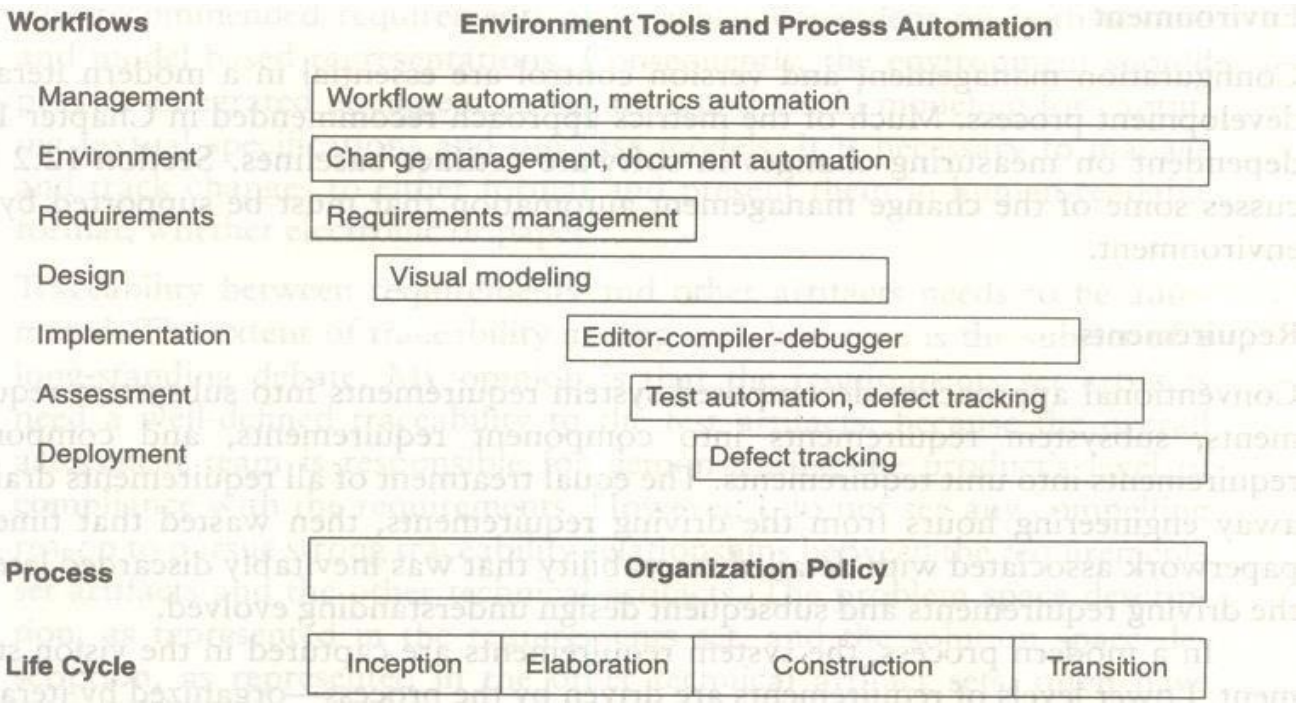


Figure 4-5: Process Automation

Project Control and process instrumentation

Software metrics are used to implement the activities and products of the software development process. Hence, the quality of the software products and the achievements in the development process can be determined using the software metrics.

Need for Software Metrics

- Software metrics are needed for calculating the cost and schedule of a software product with great accuracy.
- Software metrics are required for making an accurate estimation of the progress.
- The metrics are also required for understanding the quality of the software product.

Indicators

An indicator is a metric or a group of metrics that provides an understanding of the software process or software product or a software project. A software engineer assembles measures and produce metrics from which the indicators can be derived.

Two types of indicators are: (i) Management indicators.
(ii) Quality indicators.

Management Indicators

The management indicators i.e., technical progress, financial status and staffing progress are used to determine whether a project is on budget and on schedule. The management indicators that indicate financial status are based on earned value system.

Quality Indicators

The quality indicators are based on the measurement of the changes occurred in software.

Seven Core Metrics of Software Project

Software metrics instrument the activities and products of the software development/integration process. Metrics values provide an important perspective for managing the process. The most useful metrics are extracted directly from the evolving artifacts.

There are seven core metrics that are used in managing a modern process.

Seven core metrics related to project control:

Management Indicators	**Quality Indicators**
Work and Progress	Change traffic and stability
Budgeted cost and expenditures	Breakage and modularity
Staffing and team dynamics	Rework and adaptability
	Mean time between failures (MTBF) and maturity

Management Indicators

Work and progress

This metric measure the work performed over time. Work is the effort to be accomplished to complete a certain set of tasks. 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 overtime) against that plan.

The default perspectives of this metric are:

Software architecture team: - Use cases demonstrated.

Software development team: - SLOC under baseline change management, SCOs closed

Software assessment team: - SCOs opened, test hours executed and evaluation criteria meet.

Software management team: - milestones completed.

The below figure shows expected progress for a typical project with three major releases

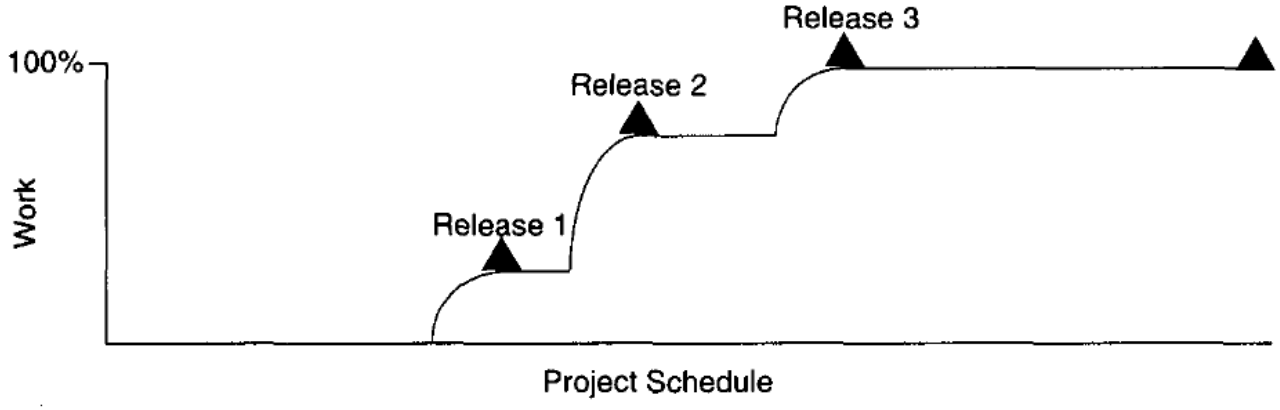


Figure 4-6: Work and Progress

Budgeted cost and expenditures

This metric measures cost incurred over time. Budgeted cost is the planned expenditure profile over the life cycle of the project. To maintain management control, measuring cost expenditures over the project life cycle is always necessary. Tracking financial progress takes on an organization - specific format. Financial performance can be measured by the use of an earned value system, which provides highly detailed cost and schedule insight. The basic parameters of an earned value system, expressed in units of dollars, are as follows:

Expenditure Plan - It is the planned spending profile for a project over its planned schedule.

Actual progress - It is the technical accomplishment relative to the planned progress underlying the spending profile.

Actual cost: It is the actual spending profile for a project over its actual schedule. **Earned value**: It is the value that represents the planned cost of the actual progress. **Cost variance**: It is the difference between the actual cost and the earned value. **Schedule variance**: It is the difference between the planned cost and the earned value. Of all parameters in an earned value system, actual progress is the most subjective

Assessment: Because most managers know exactly how much cost they have incurred and how much schedule they have used, the variability in making accurate assessments is centered in the actual progress assessment. The default perspectives of this metric are cost per month, full-time staff per month and percentage of budget expended.

Staffing and team dynamics

This metric measure the personnel changes over time, which involves staffing additions and reductions over time. 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 circumstances, staffing can vary. Increase in staff can slow overall project progress as new people consume the productive team of existing people in coming up to speed. Low attrition of good people is a sign of success. The default perspectives of this metric are people per month added and people per month leaving.

These three management indicators are responsible for technical progress, financial status and staffing progress.

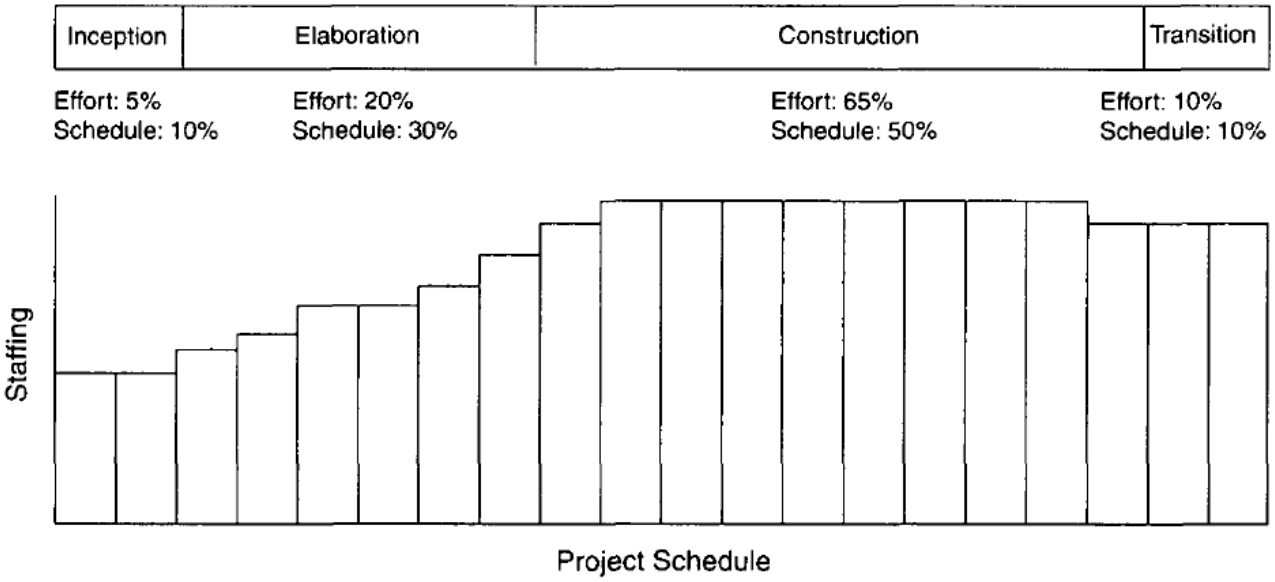


Figure 4-7: staffing and Team dynamics

Quality Indicators

Change traffic and stability

This metric measures the change traffic over time. The number of software change orders opened and closed over the life cycle is called change traffic. Stability specifies the relationship between opened versus closed software change orders. This metric can be collected by change type, by release, across all releases, by term, by components, by subsystems, etc.

The below figure shows stability expectation over a healthy project's life cycle

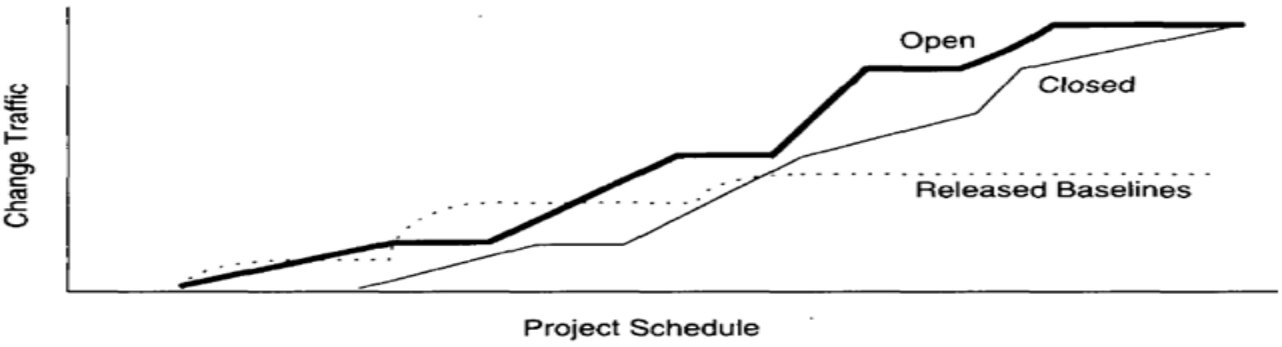


Figure 4-8: Change traffic and stability

Breakage and modularity

This metric measures the average breakage per change over time. Breakage is defined as the average extent of change, which is the amount of software baseline that needs rework and measured in source lines of code, function points, components, subsystems, files or other units.
Modularity is the average breakage trend over time. This metric can be collected by revoke

SLOC per change, by change type, by release, by components and by subsystems.

Rework and adaptability

This metric measures the average rework per change over time. 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. This metric provides insight into rework measurement. All changes are not created equal. Some changes can be made in a staff- hour, while others take staff-weeks. This metric can be collected by average hours per change, by change type, by release, by components and by subsystems.

MTBF and Maturity

This metric measures defect rater over time. MTBF (Mean Time between Failures) is the average usage time between software faults. It 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.

Software errors can be categorized into two types deterministic and nondeterministic. Deterministic errors are also known as Bohr-bugs and nondeterministic errors are also called as Heisen-bugs. Bohr-bugs are a class of errors caused when the software is stimulated in a certain way such as coding errors. Heisen-bugs are software faults that are coincidental with a certain probabilistic occurrence of a given situation, such as design errors. This metric can be collected by failure counts, test hours until failure, by release, by components and by subsystems.

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

Life - Cycle Expectations

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

• They provide insight into the waste 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 project phases, however, should follow the general pattern shown. A mature development organization should be able to describe metrics targets that are much more definitive and precise for its line of business and specific processes.

Table 4-1. The default pattern of life-cycle metrics evolution

metric	inception	elaboration	construction	transition
Progress	5%	25%	90%	100%
Architecture	30%	90%	100%	100%
Applications	<5%	20%	85%	100%

Expenditures	Low	Moderate	High	High
Effort	5%	25%	90%	100%
Schedule	10%	40%	90%	100%
Staffing	Small team	Ramp up	Steady	Varying
Stability	Volatile	Moderate	Moderate	Stable
Architecture	Volatile	Moderate	Stable	Stable
Applications	Volatile	Volatile	Moderate	Stable
Modularity	50%-100%	25%-50%	<25%	5%-10%
Architecture	>50%	>50%	<15%	<5%
Applications	>80%	>80%	<25%	<10%
Adaptability	Varying	Varying	Benign.	Benign
Architecture	Varying	Moderate	Benign	Benign
Applications	Varying	Varying	Moderate	Benign
Maturity	Prototype	Fragile	Usable	Robust
Architecture	Prototype	Usable	Robust	Robust
Applications	Prototype	Fragile	Usable	Robust

Pragmatic Software Metrics

Measuring is useful, but it doesn't do any thinking for the decision makers. It only provides data to help them to ask right questions, understand the context, and make objective decisions. Because of the highly dynamic nature of software projects, these measures must be available at any time, tailorable to various subsets of the evolving product (release, version, component, class), and maintained so that trends can be assessed (first and second derivatives with respect to time). This situation has been achieved in practice

only in projects where the metrics were maintained on-line as an automated by-product of the development/integration environment. The basic characteristics of a good metric are as follows:

1. It is considered meaningful by the customer, manager, and performer. If any one of these stakeholders does not see the metric as meaningful, it will not be used. "The customer is always right" is a sales motto, not an engineering tenet. 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 increase.

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. More typically, a metric presents a perspective. It is up to the decision authority (manager, team, or other information processing entity) to interpret the metric and decide what action is necessary.

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.

When metrics expose a problem, it is important to get underneath all the symptoms and diagnose it. Metrics usually display effects; the causes require synthesis of multiple perspectives and reasoning. For example, reasoning is still required to interpret the following situations correctly:

• A low number of change requests to a software baseline may mean that the software is mature and error-free, or it may mean that the test team is on vacation.

• A software change order that has been open for a long time may mean that the problem was simple to diagnose and the solution required substantial rework, or it may mean that a problem was very time-consuming to diagnose and the solution required a simple change to a single line of code.

• A large increase in personnel in a given month may cause progress to increase proportionally if they are trained people who are productive from the outset. It may cause progress to decelerate if they are untrained new hires who demand extensive support from productive people to get up to speed.

Value judgments cannot be made by metrics; they must be left to smarter entities such as software project managers.

Metrics Automation

Many opportunities are available to automate the project control activities of a software project. A Software Project Control Panel (SPCP) is essential for managing against a plan. This panel integrates data from multiple sources to show the current status of some aspect of the project. The panel can support standard features and provide extensive capability for detailed situation analysis. SPCP is one example of metrics automation approach that collects, organizes and reports values and trends extracted directly from the evolving engineering artifacts.

SPCP

To implement a complete SPCP, the following are necessary.

Metrics primitives - trends, comparisons and progressions

- A graphical user interface.
- Metrics collection agents
- Metrics data management server
- Metrics definitions - actual metrics presentations for requirements progress, implementation progress, assessment progress, design progress and other progress dimensions.
- Actors - monitor and administrator.

Monitor defines panel layouts, graphical objects and linkages to project data. Specific monitors called roles include software project managers, software development team leads, software architects and customers. Administrator installs the system, defines new mechanisms, graphical objects and linkages. The whole display is called a panel. Within a panel are graphical objects, which are types of layouts such as dials and bar charts for information. Each graphical object displays a metric. Panel contains a number of graphical objects positioned in a particular geometric layout. A metric shown in a graphical object is labeled with the metric type, summary level and insurance name (line of code, subsystem, server1). Metrics can be displayed in two modes value, referring to a given point in time and graph referring to multiple and consecutive points in time.

Metrics can be displayed with or without control values. A control value is an existing expectation either absolute or relative that is used for comparison with a dynamically changing metric. Thresholds are examples of control values.

The basic fundamental metrics classes are trend, comparison and progress.

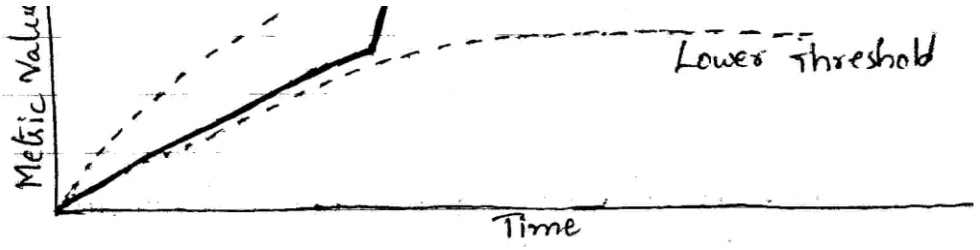


Figure 4-9: Comparison Progress

The format and content of any project panel are configurable to the software project manager's preference for tracking metrics of top-level interest. The basic operation of an SPCP can be described by the following top-level use case.

i. Start the SPCP
ii. Select a panel preference
iii. Select a value or graph metric
iv. Select to superimpose controls
v. Drill down to trend
vi. Drill down to point in time.
vii. Drill down to lower levels of information
viii. Drill down to lower level of indicators.