## UNIT 4:

## 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 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.

**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

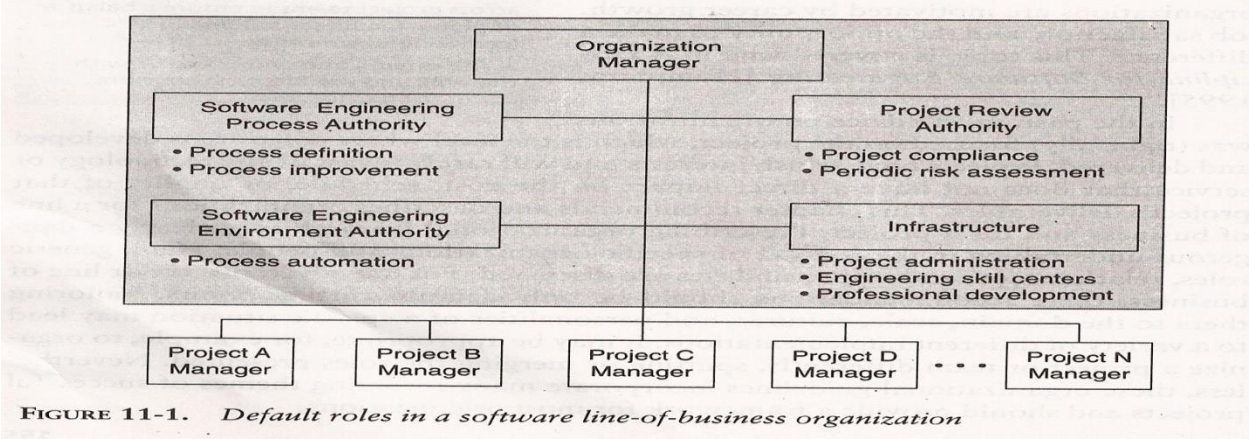



FIGURE 11-1. *Default roles in a software line-of-business organization*

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

**PROJECT ORGANIZATIONS**

- 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

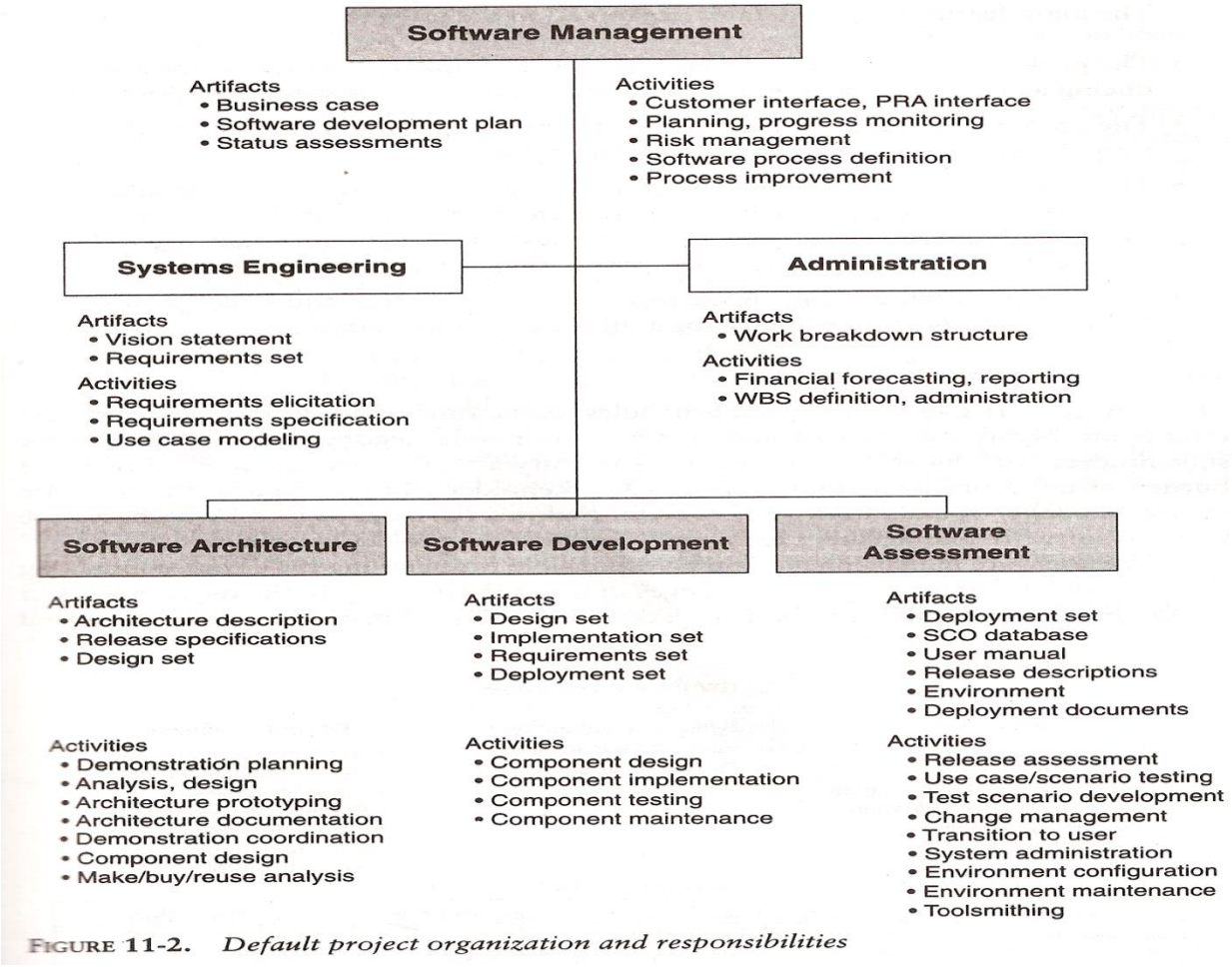



FIGURE 11-2. *Default project organization and responsibilities*

**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 ASSESSMENT TEAM**

- The team is involved in testing and product activities in parallel with the ongoing development. This is an independent team for utilizing the concurrency of activities. The use-case oriented and capability-based testing of a process is done by using two artifacts:
    - Release specification ( the plan and evaluation criteria for a release);
    - Release description (the results of a release)
- Responsibilities: The assessment team is responsible for
    - The exposure of the quality issues that affect the customer's expectations.

- Metric analysis.
- Verifying the requirements.
- Independent testing.
- Configuration control and user development.
- Building project infrastructure

**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

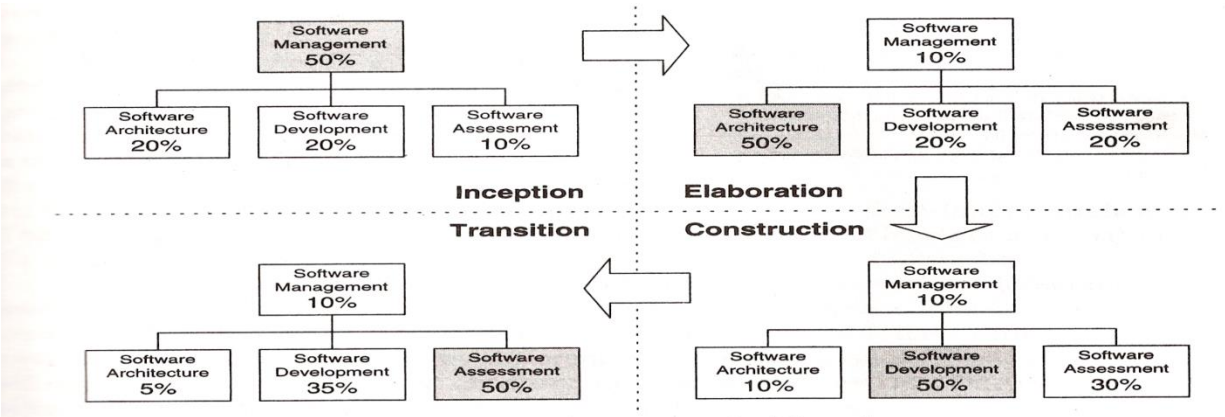



FIGURE 11-7. *Software project team evolution over the life cycle*

**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

**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.

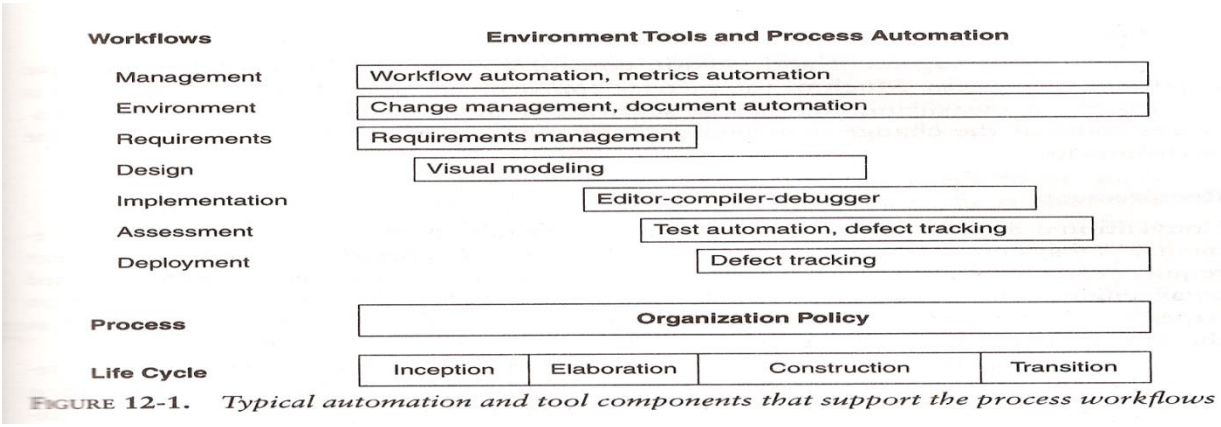



FIGURE 12-1. *Typical automation and tool components that support the process workflows*

- **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

**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

**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

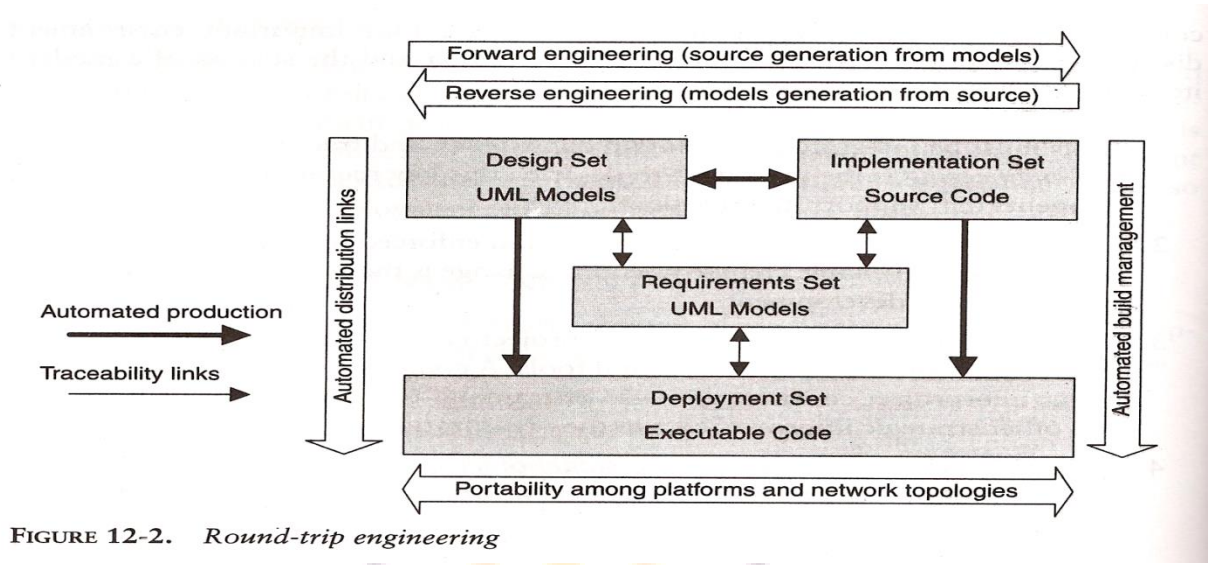



FIGURE 12-2. *Round-trip engineering*

## 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 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.

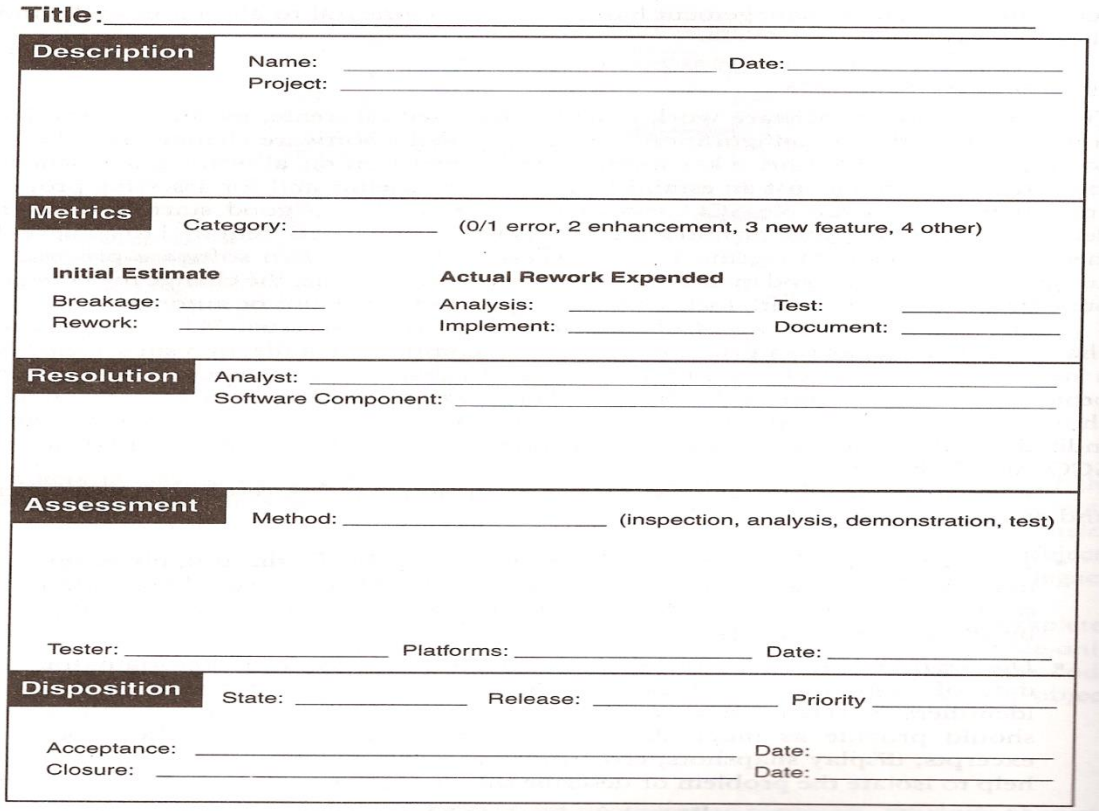



FIGURE 12-3. *The primitive components of a software change order*

**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

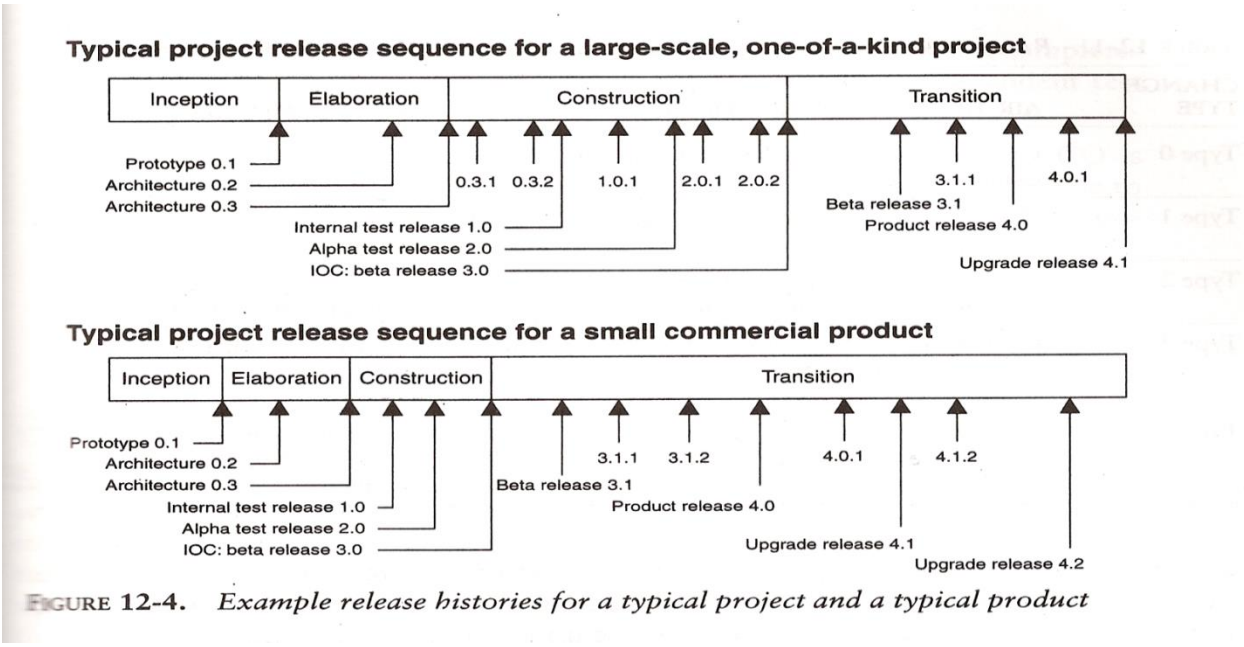



FIGURE 12-4. *Example release histories for a typical project and a typical product*

| 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 actual and submitting new SCOs.
- [In assessment]: The independent test assesses whether the SCO is completely resolved. When the independent test team deems the change to be satisfactorily resolved, the SCO is submitted to the CCB for final disposition and closure.
- [Closed]: when the development organization, independent test organization and CCB concur that the SCO is resolved, it is transitioned to a closed status.

**INFRASTRUCTURES**

- 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.
    - **Process-Primitive definitions**
        - Life-cycle phases (inception, elaboration, construction, transition)
        - Checkpoints (major milestones, minor milestones, status assessments)
        - Artifacts (requirements, design, implementation, deployment, management sets)
        - Roles and responsibilities (PRA, SEPA, SEEA, project teams).
    - **Organization software policies**
        - Work breakdown structure
        - Software development plan
        - Baseline change management
        - Software metrics
        - Development environment
        - Evaluation criteria and acceptance criteria
        - Risk management
        - Testing and assessment.
    - **Walver policy**

- **Appendixes**
  - Current process assessment
  - Software process improvement plan.
- 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)

**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.

**PROJECT CONTROL & PROCESS INSTRUMENTATION**

INTRODUCTION: 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.

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:

Management indicators.

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**

1. Work and Progress
2. Budgeted cost and expenditures
3. Staffing and team dynamics

**Quality Indicators**

4. Change traffic and stability
5. Breakage and modularity
6. Rework and adaptability

7. Mean time between failures (MTBF) and maturity

**MANAGEMENTINDICATORS:**

1. Work and progress

This metric measures 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.

Budgeted cost and expenditures

This metric measure 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 thespending profile.

Actual cost: It is the actual spending profile for a project over its actual schedule.

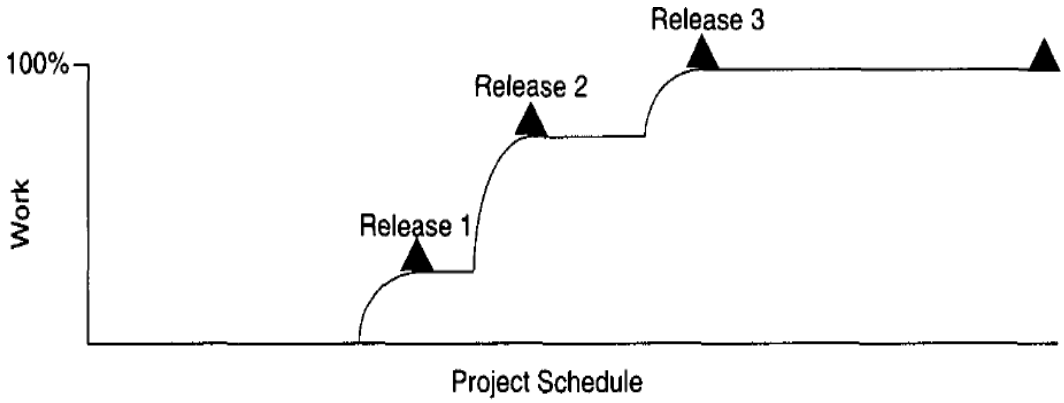
Earned value: It is the value that represents the planned cost of the actual progress.

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

staff per month and percentage of budget expended.

Staffing and team dynamics

This metric measures 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 ofthis 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.

Budgeted cost and expenditures

This metric measure 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.

Staffing and team dynamics

This metric measures 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.

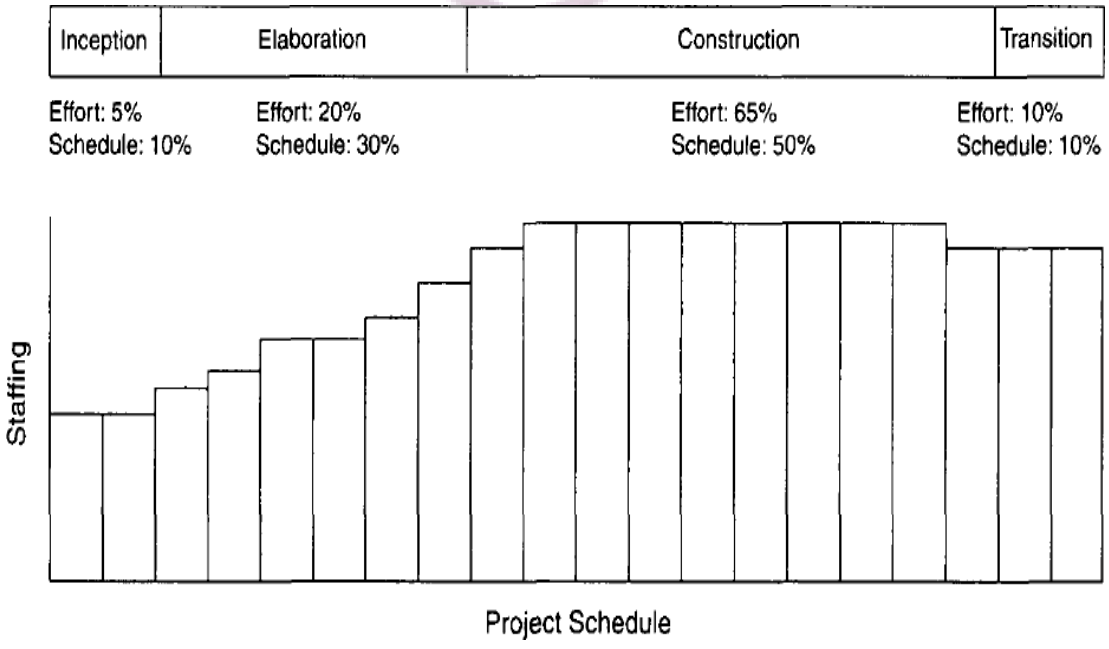



**Fig: staffing and Team dynamics**

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

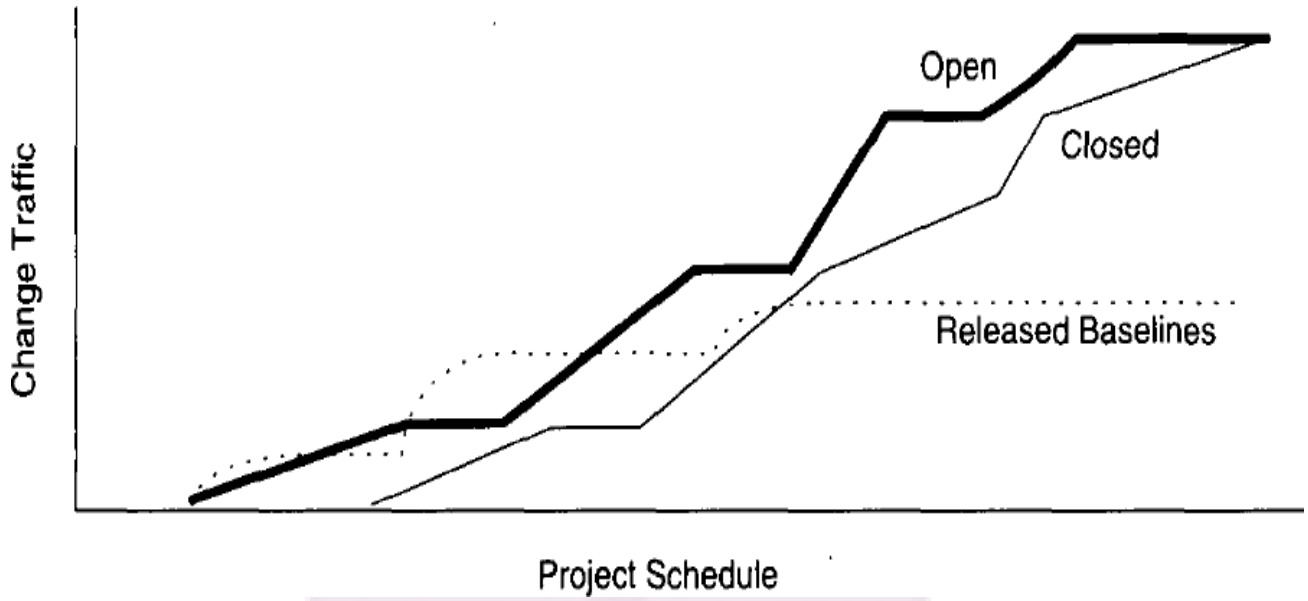



Fig: Change traffic and stability

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 measure defect rather over time. MTBF (Mean Time Between Failures) is theaverage 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:**

There is no mathematical or formal derivation for using seven core metrics properly. However, there were specific reasons for selecting them:

The quality indicators are derived from the evolving product rather than the artifacts. They provide inside 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 and the current trend provides tangible indicators for management action.

Table: The default pattern of life cycle 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% |

**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

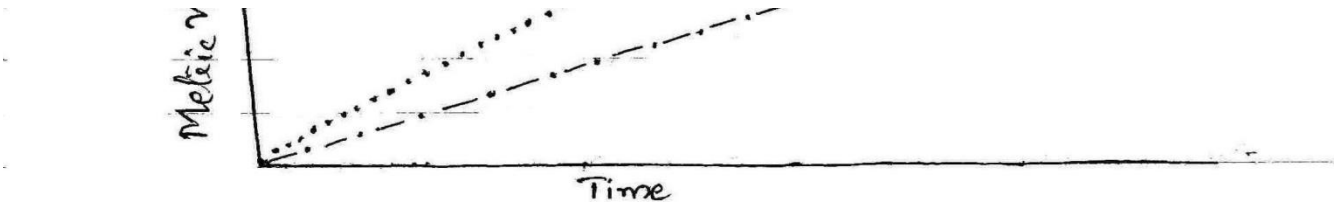
the project. The panel can support standard features and provide extensive capability for detailed situation analysis. SPCP is one example of metrics automation approach that collects, organizes and reports values and trends extracted directly from the evolving engineering artifacts.

SPCP:

To implement a complete SPCP, the following are necessary.

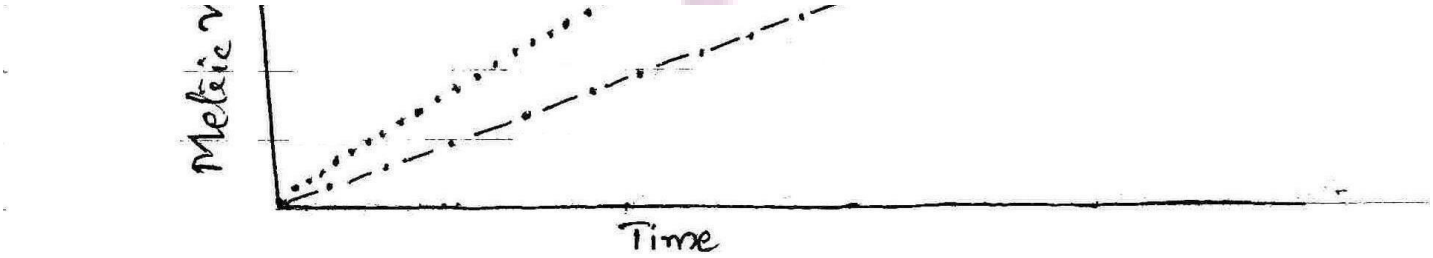
- Metrics primitives - trends, comparisons and progressions
- A graphical user interface.
- Metrics collection agents
- Metrics data management server
- Metrics definitions - actual metrics presentations for requirementsprogress, implementation progress, assessment progress, design progress and other 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. A panel contains a number of graphical objects positioned in a particular geometric layout. A metric shown in a graphical object is labelled 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 dynamicallychanging metric. Thresholds are examples of control values.




The basic fundamental metrics classes are trend, comparison and 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.

,

(3) The real monetary value of documentation is to support later modifications by a separate test team, a separate maintenance team, and operations personnel who are not software literate.

Do it twice. If a computer program is being developed for the first time, arrange matters so that the version finally delivered to the customer for operational deployment is actually the second version insofar as critical design/operations are concerned. Note that this is simply the entire process done in miniature, to a time scale that is relatively small with respect to the overall effort. In the first version, the team must have a special broad competence where they can quickly sense trouble spots in the design, model them, model alternatives, forget the straightforward aspects of the design that aren't worth studying at this early point, and, finally, arrive at an error-free program.

Plan, control, and monitor testing. Without question, the biggest user of project resources-manpower, computer time, and/or management judgment-is the test phase.

This is the phase of greatest risk in terms of cost and schedule.

It occurs at the latest point in the schedule, when backup alternatives are least available, if at all. The previous three recommendations were all aimed at uncovering and solving problems before entering the test phase. However, even after doing these things, there is still a test phase and there are still important things to be done, including:

(1) employ a team of test specialists who were not responsible for the original design;

(2) employ visual inspections to spot the obvious errors like dropped minus signs, missing factors of two, jumps to wrong addresses (do not use the computer to detect this kind of thing, it is too expensive);

(3) test every logic path;

(4) employ the final checkout on the target computer.

**1.** Involve the customer. It is important to involve the customer in a formal way so that hehas committed himself at earlier points before final delivery. There are three points

following requirements definition where the insight, judgment, and commitment of the customer can bolster the development effort. These include a "preliminary software review" following the preliminary program design step, a sequence of "critical software design reviews" during program design, and a "final software acceptance review".