ESPM UNIT - V

Project Control and Process Instrumentation: Seven Core Metrics, Management Indicators, Quality Indicators, Life Cycle Expectations Pragmatic Software Metrics, Metrics Automation.

Tailoring the Process: Process Discriminates.

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.

THE SEVEN CORE METRICS

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

a) 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)

b) 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)

Overview of the seven core metrics

METRIC	PURPOSE	PERSPECTIVES
Work and progress	Iteration planning, 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. actuals, 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 SLOC 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.
- They improve fidelity across the life cycle.

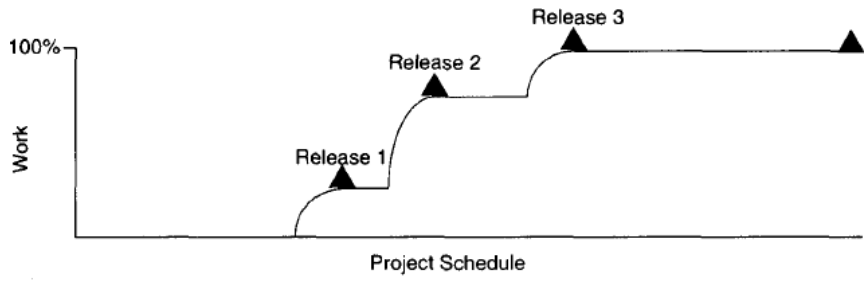
MANAGEMENT INDICATORS

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.

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), 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



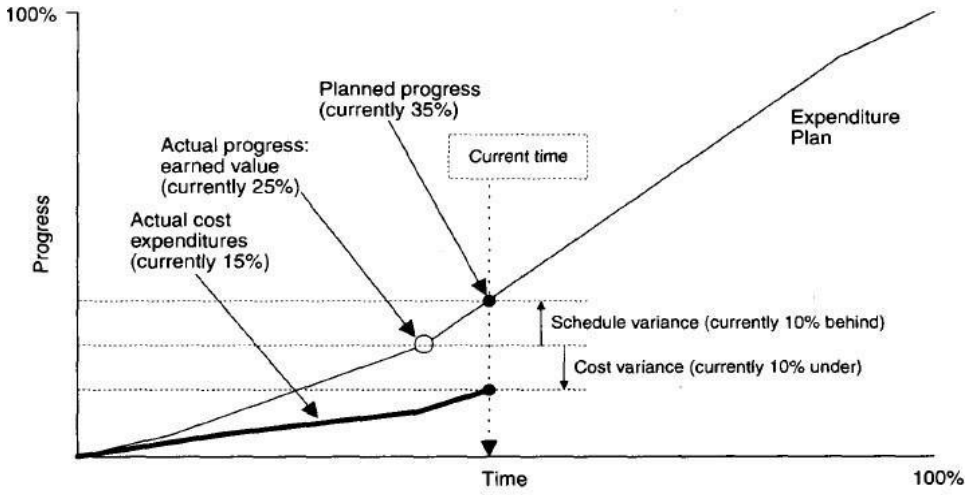
Expected progress for a typical project with three major releases

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.

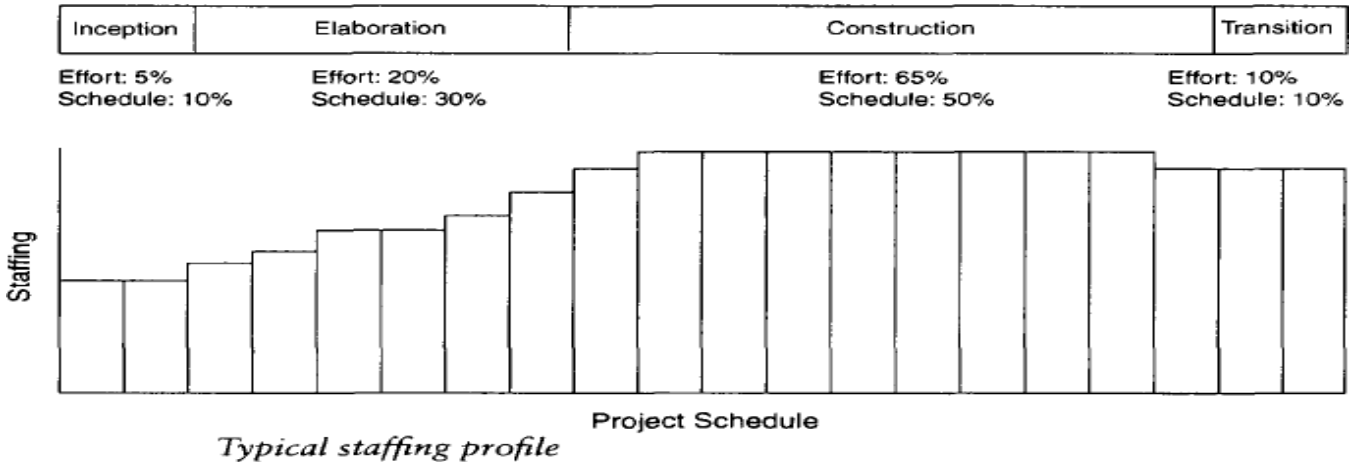


The basic parameters of an earned value system

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 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 development, the sizes of the maintenance and development teams may be the same.



Typical staffing profile

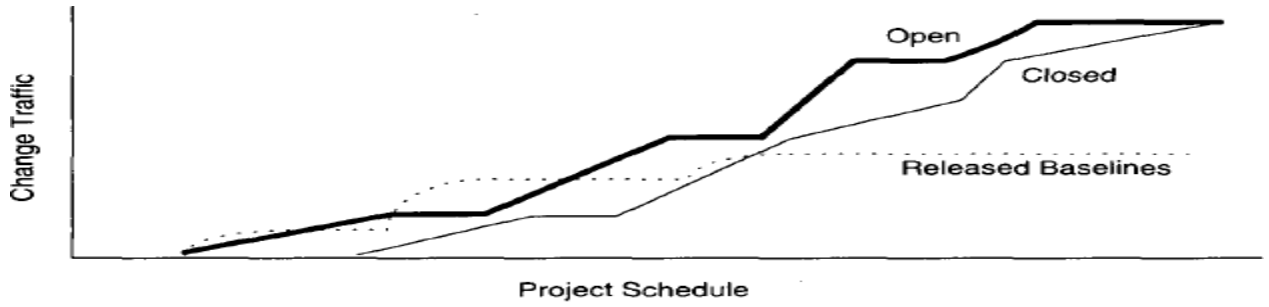
QUALITY INDICATORS

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).

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 This metric can be collected by change type, by release, across all releases, by team, by components, by subsystem, and so forth. ***Stability*** is defined as the relationship between opened versus closed SCOs.

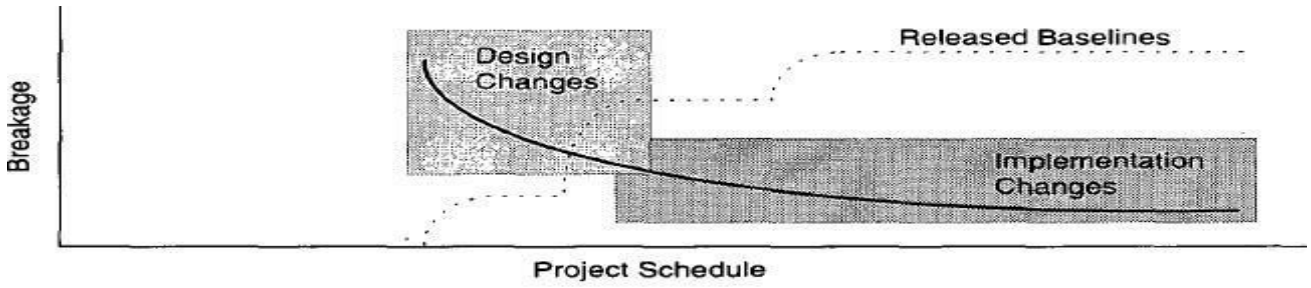


Stability expectation over a healthy project's life cycle

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. For a healthy project, the trend expectation is decreasing or stable

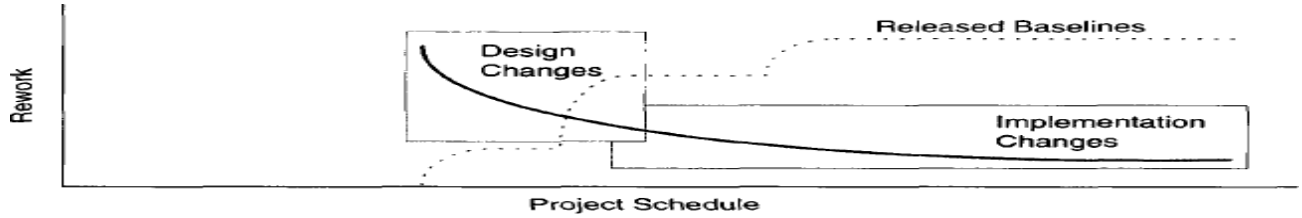


Modularity expectation over a healthy project's life cycle

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.

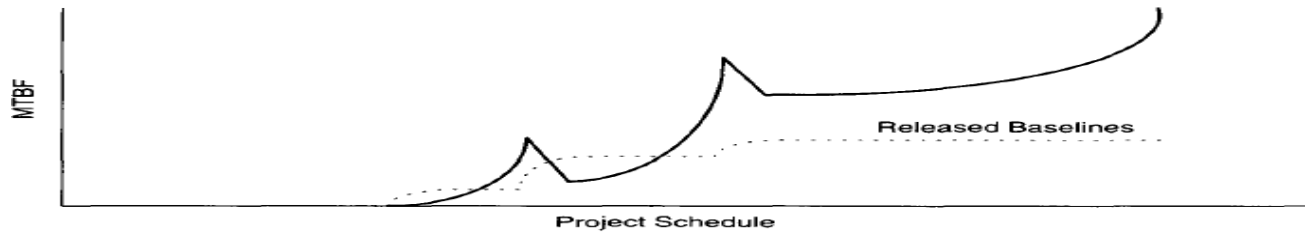


Adaptability expectation over a healthy project's life cycle

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. MTBF stands for Mean- Time- Between –Failures.

Maturity is defined as the MTBF trend over time



Maturity expectation over a healthy project's life cycle

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 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 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. Theonly 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 actiondirectly.
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.

METRICS AUTOMATION

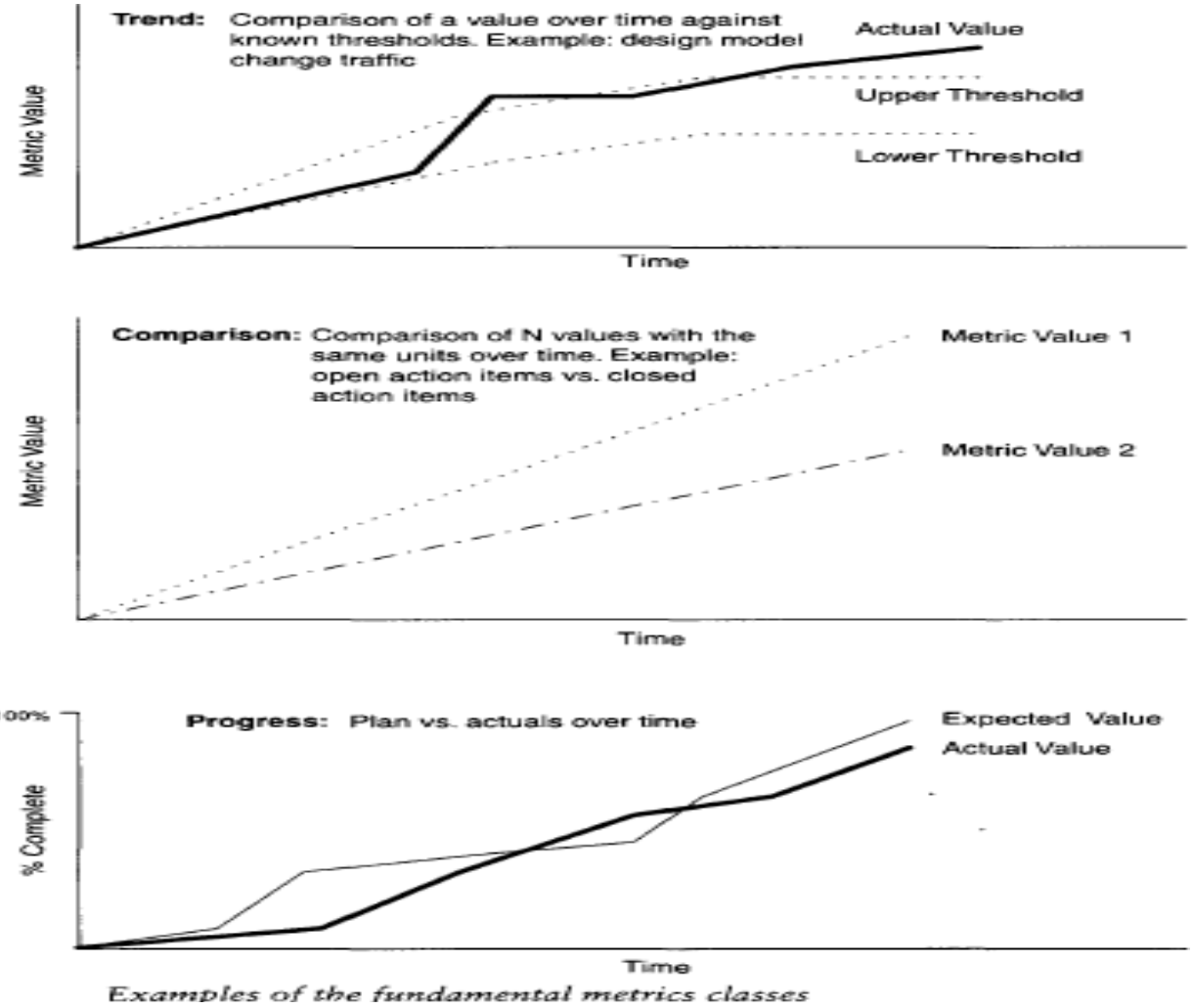
There are many opportunities to automate the project control activities of a software project. For managing against a plan, a software project control panel (SPCP) that maintains an on-line version of the status of evolving artifacts provides a key advantage.

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 theengineering 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.

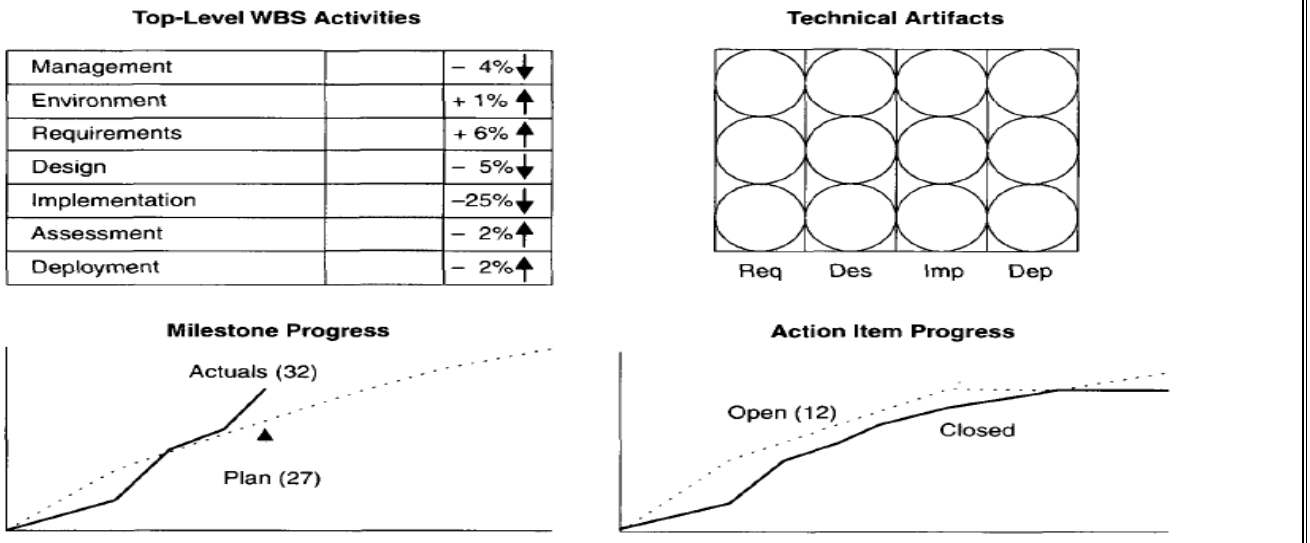
- ***Monitor:*** defines panel layouts from existing mechanisms, graphical objects, and linkages to project data; queries data to be displayed at different levels of abstraction
- ***Administrator:*** installs thesystem; defines new mechanisms, graphical objects, and linkages; archiving functions; defines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



Examples of the fundamental metrics classes

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 valuestatus. (In Figure 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 achievementof 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.



Example SPCP display for a top-level project situation

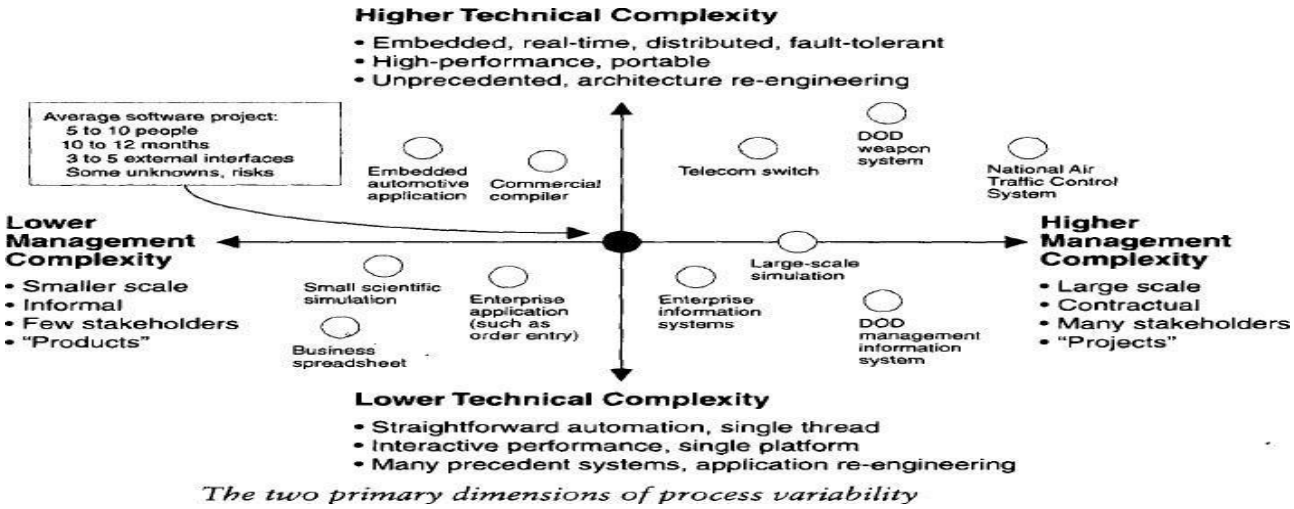
The following top-level use case, which describes the basic operational concept of an SPCP, corresponds to a monitor interacting with the control panel:

- 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.

PROCESS DISCRIMINATES

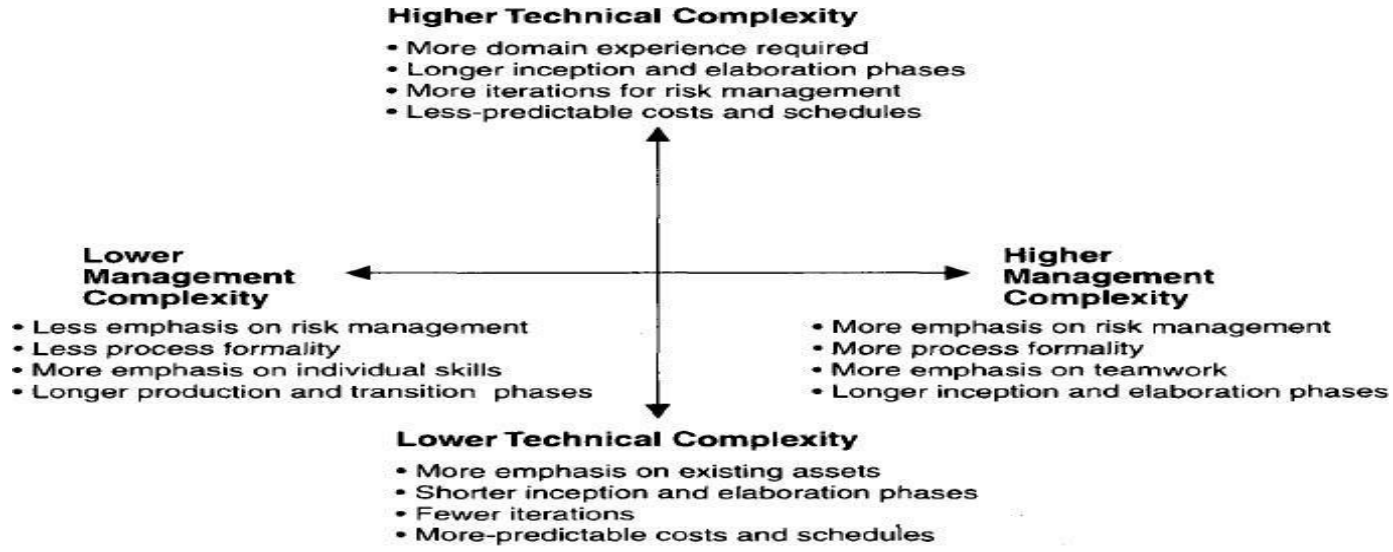
In tailoring the management process to a specific domain or project, there are tow dimensions of discriminating factors: technical complexity and management complexity.

The Figure 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.



The two primary dimensions of process variability

Figure summarizes the different priorities along the two dimensions.



Priorities for tailoring the process framework

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 teammember.
- 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:

a) There are numerous mundane jobs in any large project, especially in the overhead workflows.
b) 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 synchronize project-level and subproject-level workflows and to balance resources.

Process discriminators that result from differences in project size

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

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.

Process discriminators that result from differences in 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 can 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

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.

Process discriminators that result from differences in process flexibility

PROCESS PRIMITIVE	FLEXIBLE PROCESS	INFLEXIBLE 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)

Process Maturity

The process maturity level of the development organization, as defined by thesoftware 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.

Process discriminators that result from differences in process maturity

PROCESS PRIMITIVE	MATURE, LEVEL 3 OR 4 ORGANIZATION	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

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.

Process discriminators that result from differences in 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

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.

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)

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 components in the success or failure of the project. This reflects the importance of staffing or the level of associated risk management.

Differences in workflow priorities between small and large projects

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

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.

Differences in artifacts between small and large projects

ARTIFACT	SMALL COMMERCIAL PROJECT	LARGE, COMPLEX PROJECT
Work breakdown structure	1-page spreadsheet with 2 levels of WBS elements	Financial management system with 5 or 6 levels of WBS elements
Business case	Spreadsheet and short memo	3-volume 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 use 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
User manual	On-line help and 100-page user manual	200-page user manual
Status assessment	Quarterly project reviews	Monthly project management reviews