5. PROJECT CONTROL AND PROCESS INSTRUMENTATION

5.0 INTRODUCTION

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

5.1 THE SEVEN CORE METRICS

- Seven core metrics are used in all software projects. Three are management indicators and four are quality indicators.
- **MANAGEMENT INDICATORS**
- Work and progress (work performed over time)
- Budgeted cost and expenditures (cost incurred over time)
- Staffing and team dynamics (personnel changes over time)
- **QUALITY INDICATORS**
- Change traffic and stability (change traffic over time)
- Breakage and modularity (average breakage per change over time)
- Rework and adaptability (average rework per change overtime)
- Mean time between failures (MTBF) and maturity (defect rate over time)

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

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

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

5.2 MANAGEMENT INDICATORS

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

5.2.1 Work & Progress

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

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

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

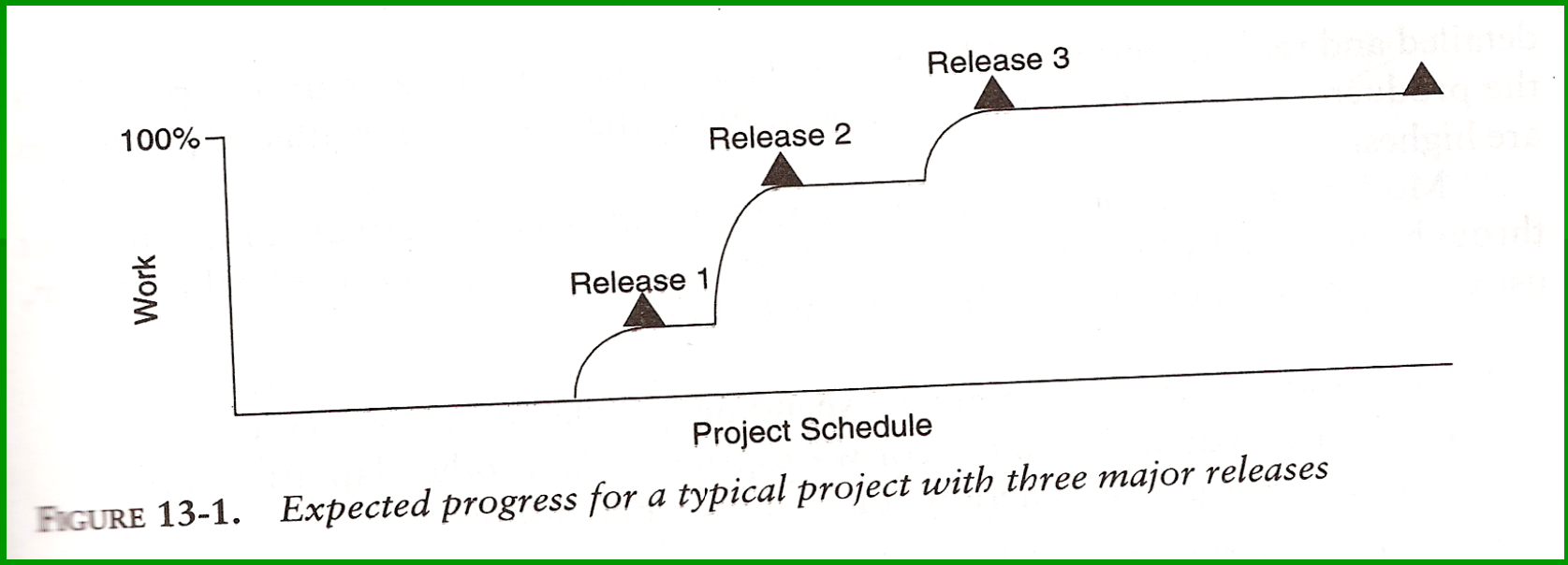


FIGURE 13-1. *Expected progress for a typical project with three major releases*

5.2.2 BUDGETED COST AND EXPENDITURES

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

- **Earned Value:** the value that represents the planned cost of the actual progress.
- **Cost variance:** the difference between the actual cost and the earned value.
- **Positive values** correspond to over - budget situations; negative values correspond to under budget situations.
- **Schedule Variance:** the difference between the planned cost and the earned value. Positive values correspond to behind-schedule situations; negative values correspond to ahead-of-schedule situations.

- Figure 13-2 provides a graphical perspective of these parameters and shows a simple example of a project situation.

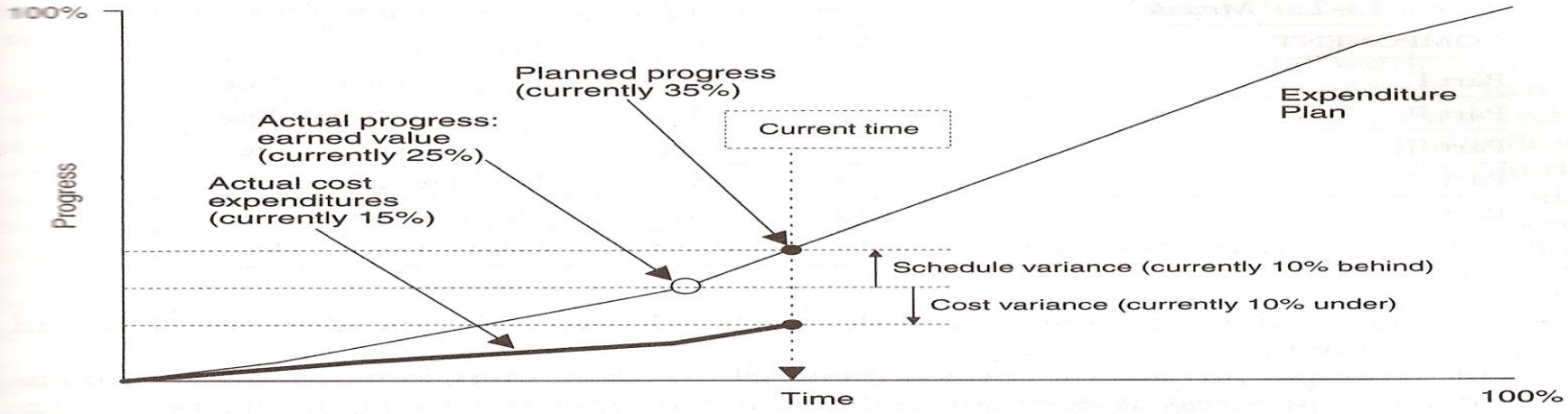


FIGURE 13-2. *The basic parameters of an earned value system*

5.2.3 STAFFING AND TEAM DYNAMICS

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

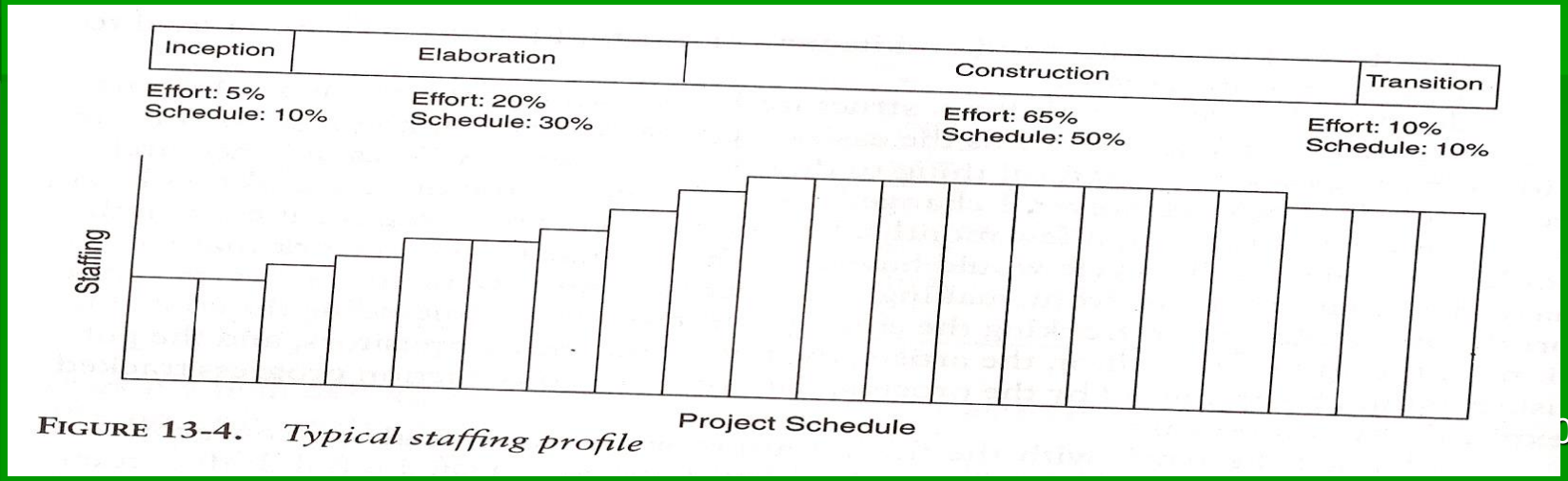


FIGURE 13-4. *Typical staffing profile*

5.3 QUALITY INDICATORS

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

5.3.1 CHANGE TRAFFIC AND STABILITY

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

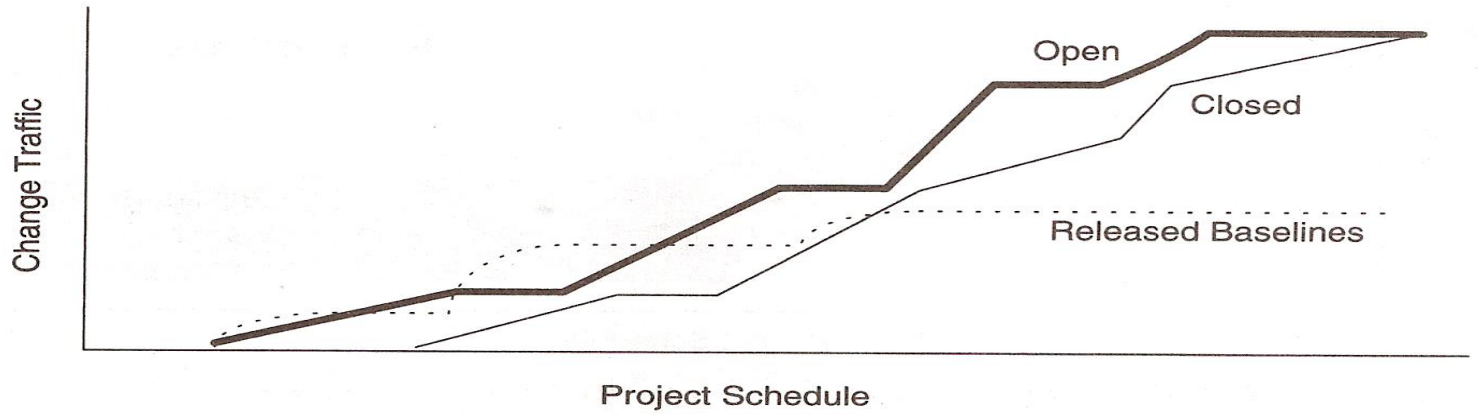


FIGURE 13-5. *Stability expectation over a healthy project's life cycle*

5.3.2 BREAKAGE AND MODULARITY

- Breakage is defined as the average extent of change, which is the amount of software baseline that needs rework (in SLOC, function points, components, subsystems, files, etc). Modularity is the average breakage trend over time. For a healthy project, the trend expectation is decreasing or stable (Figure 13-6).

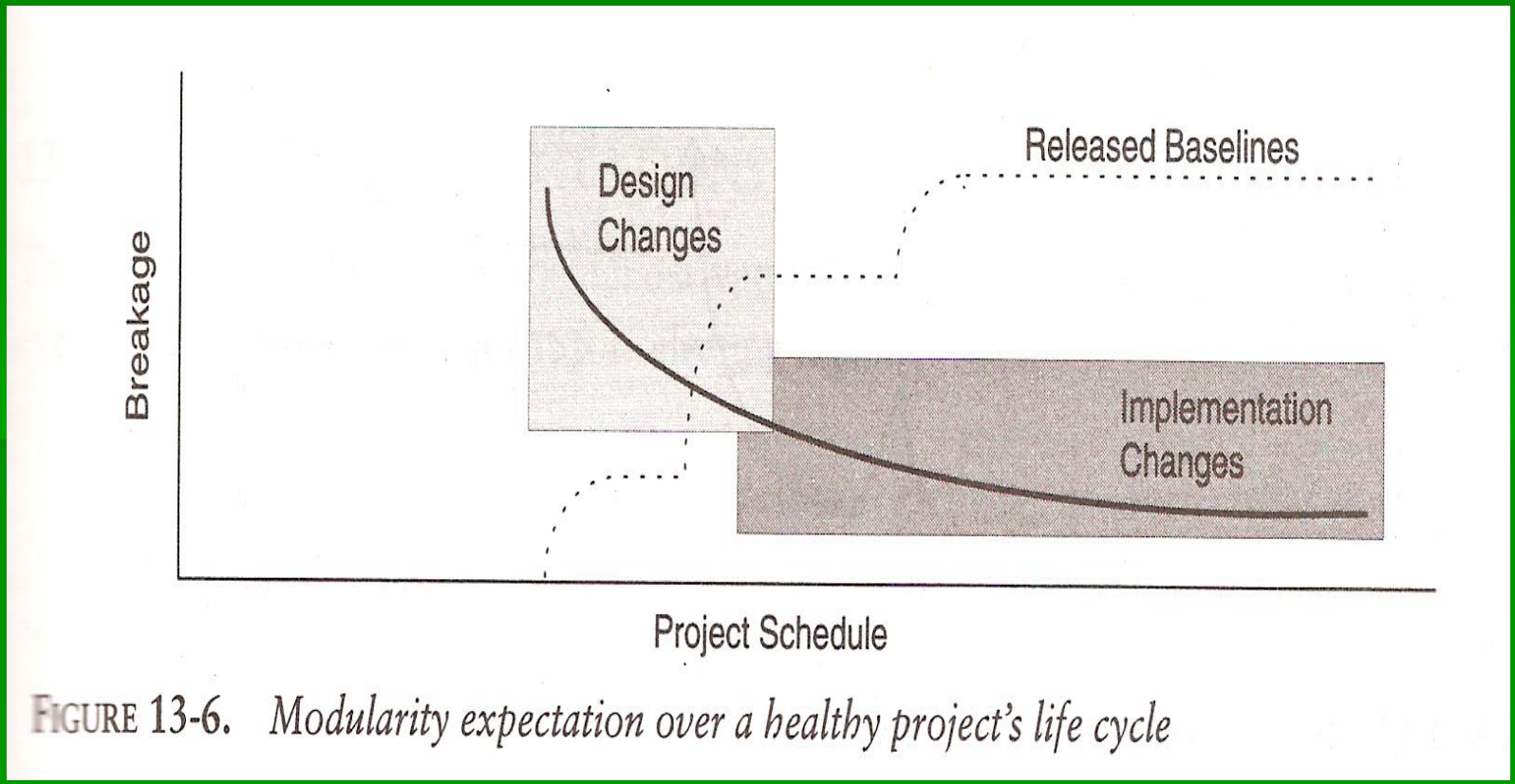


FIGURE 13-6. *Modularity expectation over a healthy project's life cycle*

5.3.3 REWORK AND ADAPTABILITY

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

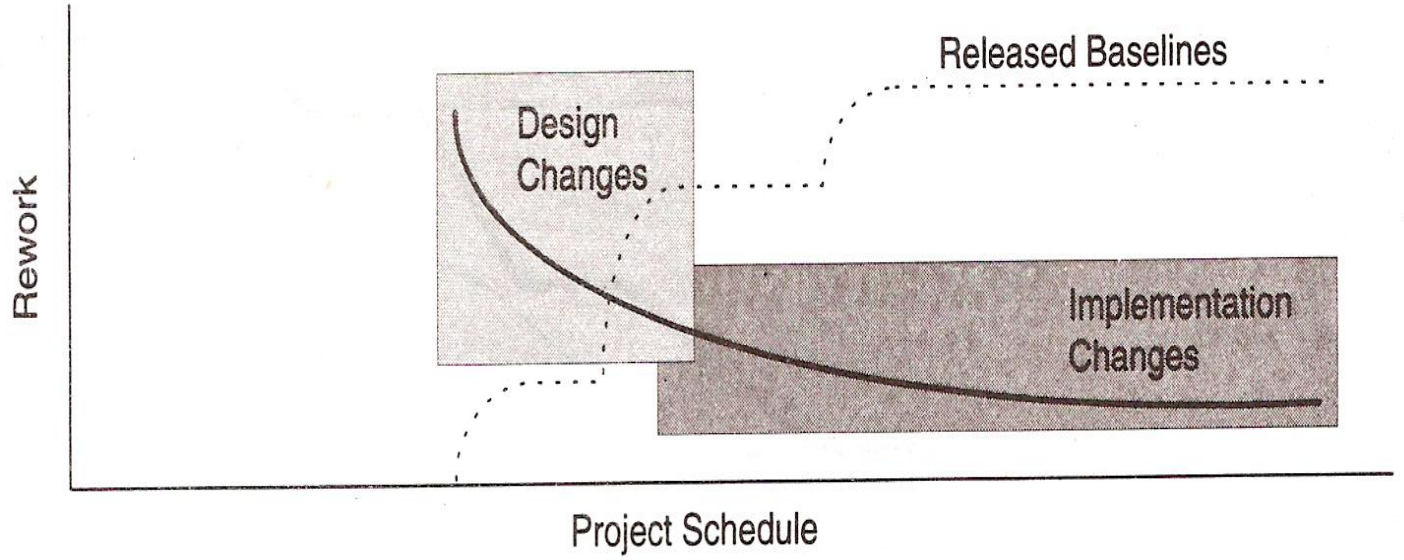
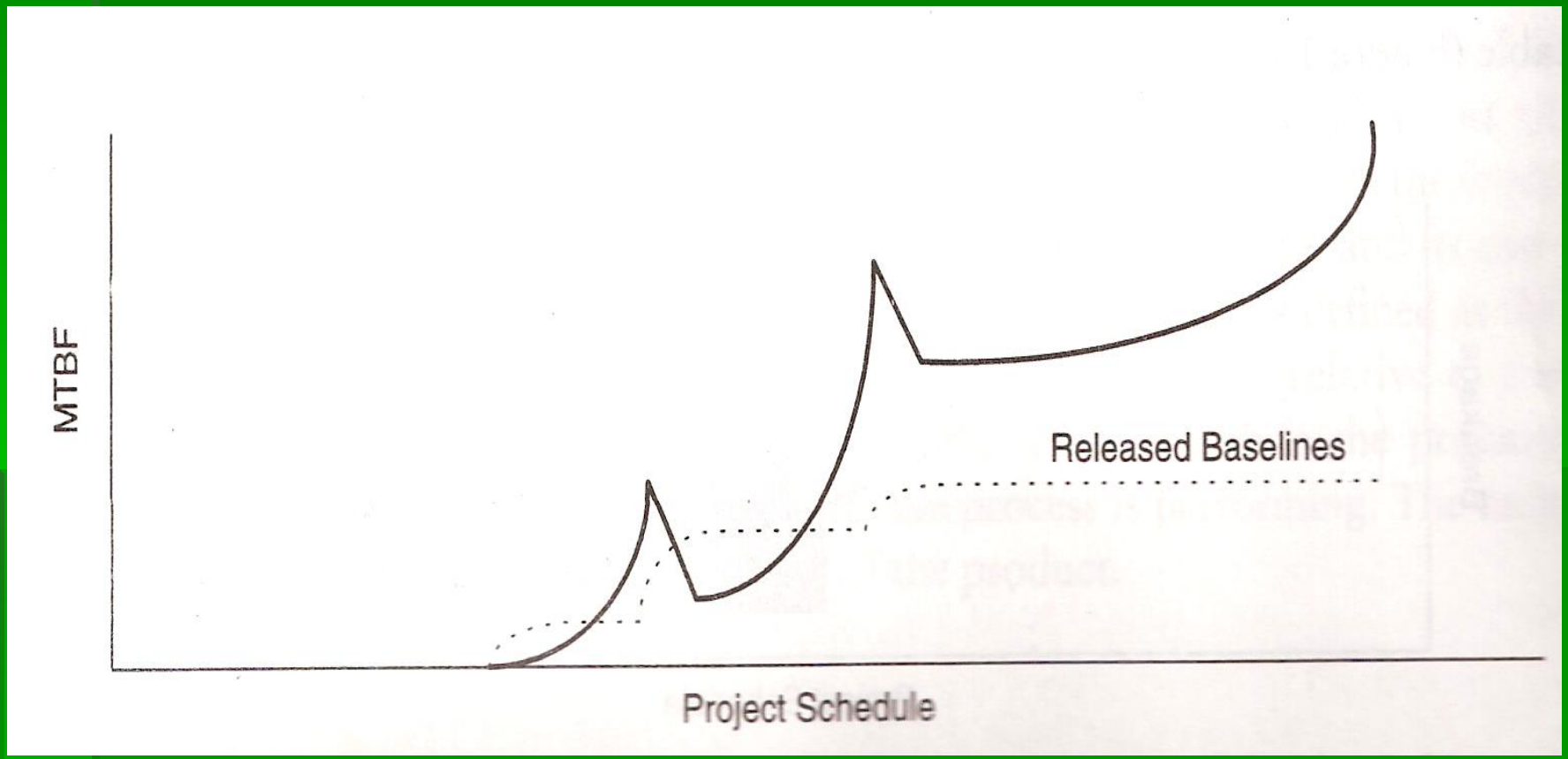


FIGURE 13-7. *Adaptability expectation over a healthy project's life cycle*

5.3.4 MTBF AND MATURITY

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



5.4 LIFE-CYCLE EXPECTATIONS

- There is no mathematical or formal derivation for using the seven core metrics. However, there were specific reasons for selecting them:
  - The quality indicators are derived form the evolving product rather than from the artifacts.
  - They provide insight into the waster generated by the process. Scrap and rework metrics are a standard measurement perspective of most manufacturing processes.
  - They recognize the inherently dynamic nature of an iterative development process. Rather than focus on the value, they explicitly concentrate on the trends or changes with respect to time.
  - The combination of insight from the current value and the current trend provides tangible indicators for management action.
- The actual values of these metrics can vary widely across projects, organizations and domains. The relative trends across the project phases. However, should follow the general pattern shown in Table 13-3.

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

5.5 PRAGMATIC SOFTWARE METRICS

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

1. It is considered meaningful by the customer, manager and performer. Customers come to software engineering providers because the providers are more expert than they are at developing and managing software. Customers will accept metrics that are demonstrated to be meaningful to the developer.
2. It demonstrates quantifiable correlation between process perturbations and business performance. The only real organizational goals and objectives are financial: cost reduction, revenue increase and margin 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.
5. It is a natural by-product of the process: The metric does not introduce new artifacts or overhead activities; it is derived directly from the mainstream engineering and management workflows.
6. It is supported by automation: Experience has demonstrated that the most successful metrics are those that are collected and reported by automated tools, in part because software tools require rigorous definitions of the data they process

5.6 METRICS AUTOMATION

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

- Metrics data management server: data management support for populating the metric displays of the GUI and storing the data extracted by the agents.
- Metrics definitions: actual metrics presentations for requirements progress (extracted from requirements set artifacts), design progress (extracted from design set artifacts), implementation progress (extracted from implementation set artifacts), assessment progress (extracted from deployment set artifacts), and other progress dimensions (extracted from manual sources, financial management systems, management artifacts, etc.)
- Actors: typically, the monitor and the administrator

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

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

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

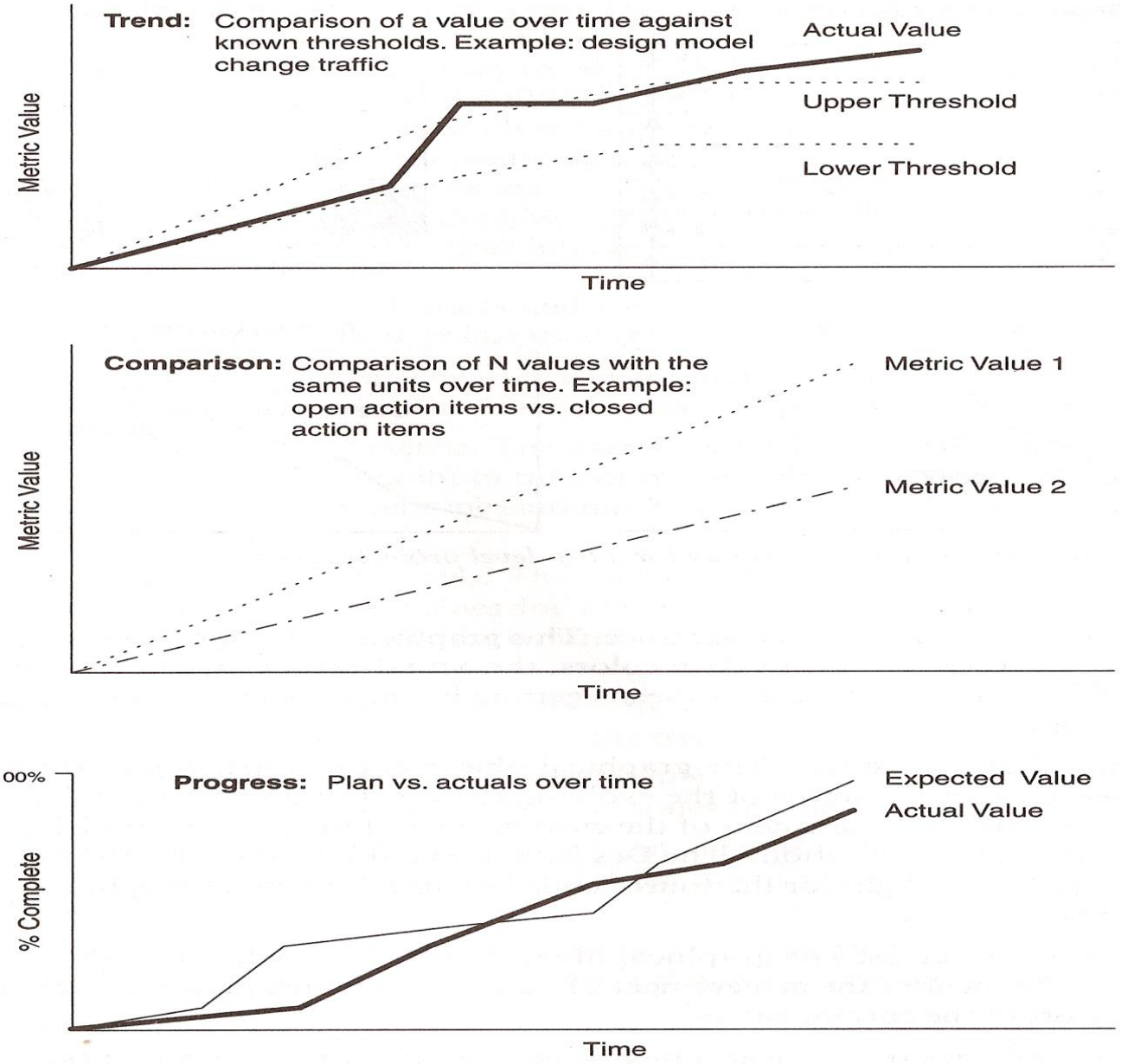


FIGURE 13-9. *Examples of the fundamental metrics classes*

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

1. **Project activity Status:** the graphical object in the upper left provides an overview of the status of the top-level WBS elements. The seven elements could be coded red, yellow and green to reflect the current earned value status. (In Figure 13-10, they are coded with white and shades of gray). For example, green would represent ahead of plan, yellow would indicate within 10% of plan, and red would identify elements that have a greater than 10% cost or schedule variance. This graphical object provides several examples of indicators: tertiary colors, the actual percentage, and the current first derivative (up arrow means getting better, down arrow means getting worse).
2. **Technical artifact status:** the graphical object in the upper right provides an overview of the status of the evolving technical artifacts. The Req light would display an assessment of the current state of the use case models and requirements specifications. The Des light would do the same for the design models, the Imp light for the source code baseline and the Dep light for the test program.

3. **Milestone progress:** the graphical object in the lower left provides a progress assessment of the achievement of milestones against plan and provides indicators of the current values.
4. **Action item progress:** the graphical object in the lower right provides a different perspective of progress, showing the current number of open and close issues.

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

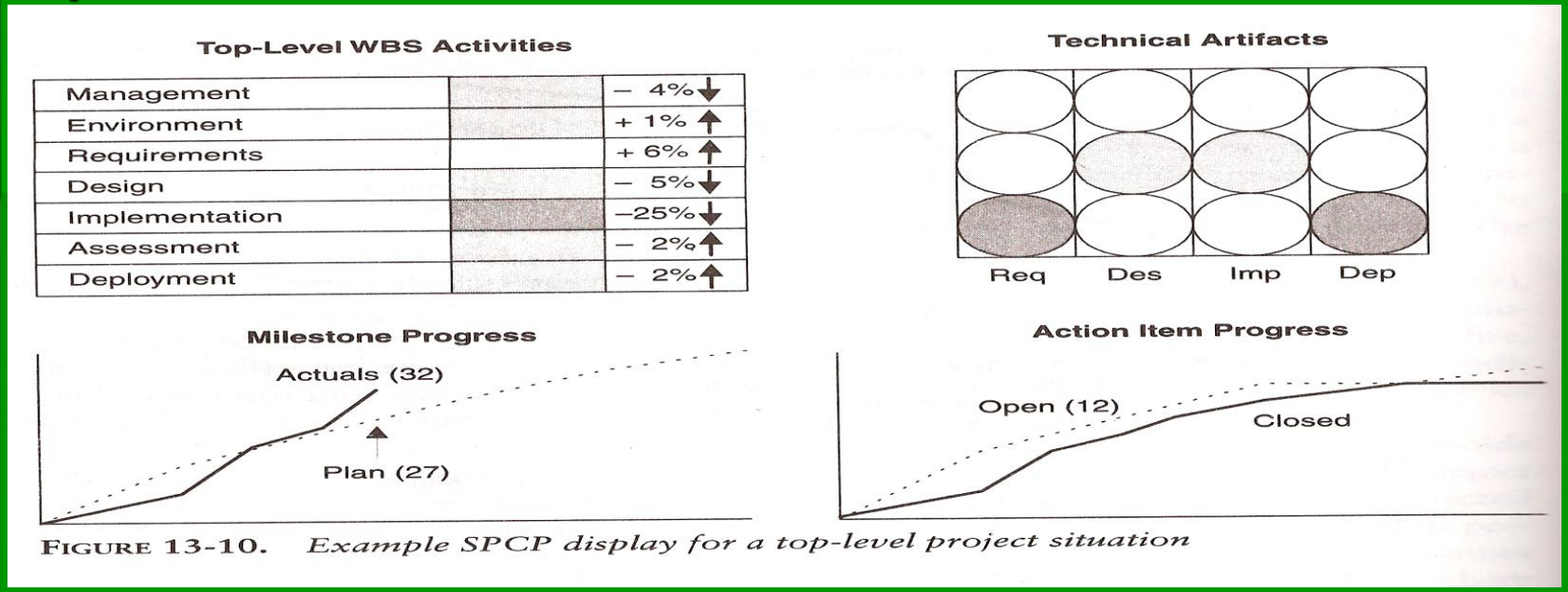


FIGURE 13-10. *Example SPCP display for a top-level project situation*

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

5.7 PROCESS DISCRIMINATES

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

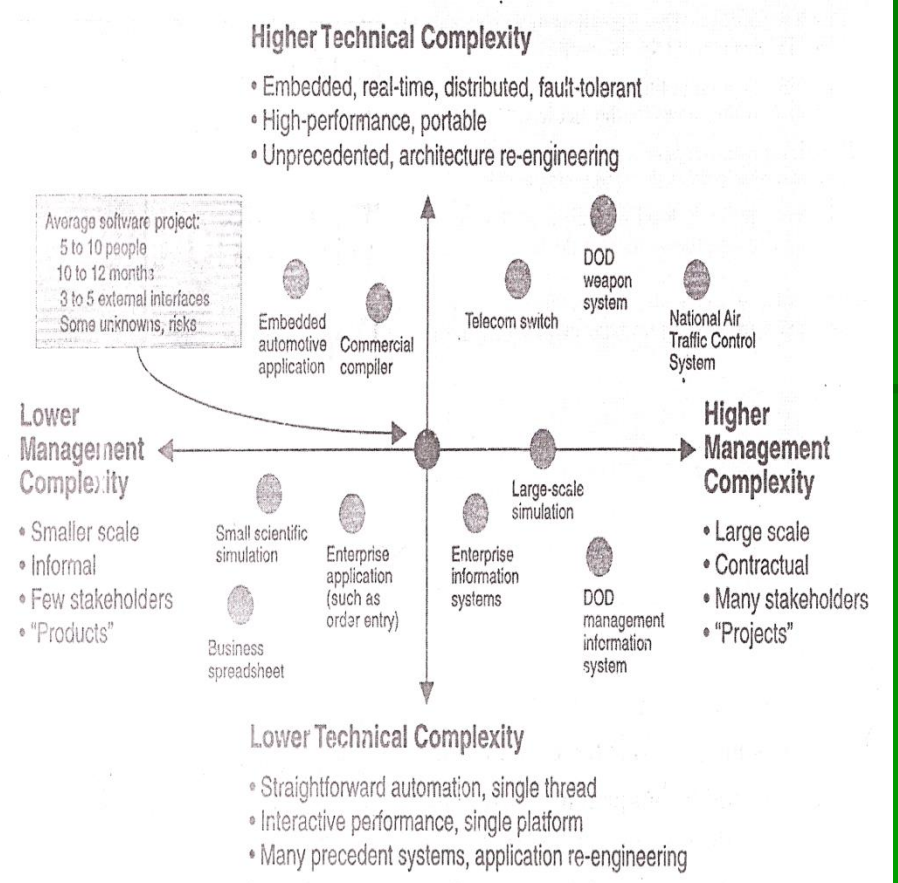


FIGURE 14-1. *The two primary dimensions of process variability*

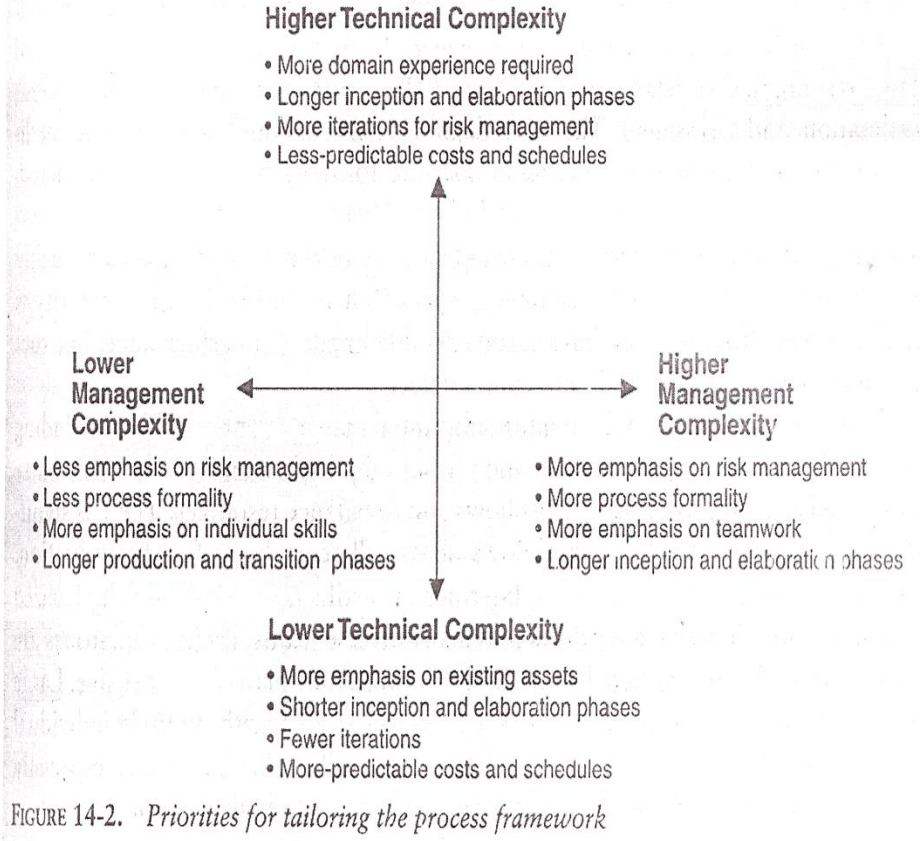


FIGURE 14-2. *Priorities for tailoring the process framework*

5.7.1 SCALE

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

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

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

5.7.2 STAKEHOLDER COHESION OR CONTENTION

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

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

5.7.3 PROCESS FLEXIBILITY OR RIGOR

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

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

5.7.4 PROCESS MATURITY

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

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

5.7.5 ARCHITECTURAL RISK

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

5.7.6 DOMAIN EXPERIENCE

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

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

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

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

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

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

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

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

MODEL QUESTIONS

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

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

5.12 The Command Center Processing and Display System-Replacement (CCPDS-R)

- The Command Center Processing and Display System-Replacement (CCPDS-R) project was performed for the U.S. Air Force by TRW Space and Defense in Redondo Beach, California. The entire project included systems engineering, hardware procurement, and software development, with each of these three major activities consuming about one-third of the total cost. The schedule spanned 1987 through 1994.
- a The metrics histories were all derived directly from the artifacts of the project's process. These data were used to manage the project and were embraced by practitioners, managers, and stakeholders.

There are very few well-documented projects with objective descriptions of what worked, what didn't, and why. This was one of my primary motivations for providing the level of detail contained in this appendix. It is heavy in project-specific details, approaches, and results, for three reasons:

1. Generating the case study wasn't much work. CCPDS-R is unique in its detailed and automated metrics approach. All the data were derived directly from the historical artifacts of the project's process.
2. This sort of objective case study is a true indicator of a mature organization and a mature project process. The absolute values of this historical perspective are only marginally useful. However, the trends, lessons learned, and relative priorities are distinguishing characteristics of successful software development.
3. Throughout previous chapters, many management and technical approaches are discussed generically. This appendix provides in a real-world example at least one relevant benchmark of performance.

- The CCPDS-R project produced a large-scale, highly reliable command and control system that provides missile warning information used by the National Command Authority. The procurement agency was Air Force Systems Command Headquarters, Electronic Systems Division, at Hanscom Air Force Base, Massachusetts. The primary user was US Space Command, and the full-scale development contract was awarded to TRWs Systems Integration Group in 1987. The CCPDS-R contract called for the development of three subsystems:

1. The Common Subsystem was the primary missile warning system within the Cheyenne Mountain Upgrade program. It required about 355,000 source lines of code, had a 48-month software development schedule, and laid the foundations for the subsystems that followed (reusable components, tools, environment, process, procedures). The Common Subsystem included a primary installation in Cheyenne Mountain, with a backup system deployed at Offutt Air Force Base, Nebraska.
2. The Processing and Display Subsystem (PDS) was a scaled-down missile warning display system for all nuclear-capable commanders-in-chief. The PDS software (about 250,000 SLOC) was fielded on remote, read-only workstations that were distributed worldwide.
3. The STRATCOM Subsystem (about 450,000 SLOC) provided both missile warning and force management capability for the backup missile warning center at the command center of the Strategic Command.

5.12.1 CCPDS-R life-cycle overview

- The CD phase was very similar in intent to the inception phase. The primary products were a system specification (a vision document), an FSD phase proposal (a business case, including the technical approach and a fixed-price-incentive and award-fee cost proposal), and a software development plan. The CD phase also included a system design review, technical interchange meetings with the government stakeholders (customer and user), and several contract-deliverable documents. These events and products enabled the FSD source selection to be based on demonstrated performance of the contractor-proposed team as well as the FSD proposal.
- From a software perspective, there was one additional source selection criterion included in the FSD proposal activities: a software engineering exercise. This was a unique but very effective approach for assessing the abilities of the two competing contractors to perform software development. The Air Force was extremely concerned with the overall software risk of this project: Recent projects had demonstrated dismal software development performance. The Air Force acquisition authorities had also been frustrated with previous situations in which a contractor's crack proposal team was not the team committed to perform after contract award, and contractor proposals exaggerated their approaches or capabilities beyond what they could deliver.
- CCPDS-R was also a very large software development activity and was one of the first projects to use the Ada programming language. There was serious concern that the Ada development environments, contractor processes, and contractor training programs might not be mature enough to use on a full-scale development effort. The purpose of the software engineering exercise was to demonstrate that the contractor's proposed software process, Ada environment, and software team were in place, were mature, and were demonstrable.

- The software engineering exercise occurred immediately after the FSD proposals were submitted. The customer provided both bidders with a simple two-page specification of a "missile warning simulator." This simulator had some of the same fundamental requirements as the CCPDS-R full-scale system, including a distributed architecture, a flexible user interface, and the basic processing scenarios of a simple CCPDS-R missile warning thread. The exercise requirements included the following:
- Use the proposed software team.
- Use the proposed software development techniques and tools.
- Use the FSD-proposed software development plan.
- Conduct a mock design review with the customer 23 days after receipt of the specification.
- Four primary use cases were elaborated and demonstrated.
- A software architecture skeleton was designed, prototyped, and documented, including two executable, distributed processes; five concurrent tasks (separate threads of control); eight components; and 72 component-to-component interfaces.
- A total of 4,163 source lines of prototype components were developed and executed. Several thousand lines of reusable components were also integrated into the demonstration.
- Three milestones were conducted and more than 30 action items resolved.
- Production of 11 documents (corresponding to the proposed artifacts) demonstrated the automation inherent in the documentation tools.
- The Digital Equipment Corporation VAX/VMS tools, Rational R1000 environment, LaTeX documentation templates, and several custom-developed tools were used.
- Several needed improvements to the process and the tools were identified. The concept of evolving the plan, requirements, process, design, and environment at each major milestone was considered potentially risky but was implemented with rigorous change management.

- In preparing for the CCPDS-R project, TRW placed a strong emphasis on evolving the right team. The CD phase team represented the essence of the architecture team which is responsible for an efficient engineering stage. This team had the following primary responsibilities:
- Analyze and specify the project requirements
- Define and develop the top-level architecture
- Plan the FSD phase software development activities
- Configure the process and development environment
- Establish trust and win-win relationships among the stakeholders

1.Network Architecture Services (NAS). This foundation middleware provided reusable components for network management, interprocess communications, initialization, reconfiguration, anomaly management, and instrumentation of software health, performance, and state. This CSCI was designed to be reused across all three CCPDS-R subsystems.

2. System Services (SSV). This CSCI comprised the software architecture skeleton, real-time data distribution, global data types, and the computer system operator interface.

3. Display Coordination (DCO). This CSCI comprised user interface control, display formats, and display population.

4. Test and Simulation (TAS). This CSCI comprised test scenario generation, test message injection, data recording, and scenario playback.

5. Common Mission Processing (CMP). This CSCI comprised the missile warning algorithms for radar, nuclear detonation, and satellite early warning messages.

6. Common Communications (CCO). This CSCI comprised external interfaces with other systems and message input, output, and protocol management.

5.9 MODERN PROJECT PROFILES

5.9.1 Continuous Integration

In the iterative development process, firstly, the overall architecture of the project is created and then all the integration steps are evaluated to identify and eliminate the design errors. This approach eliminates problems such as down stream integration, late patches and shoe-horned software fixes by implementing sequential or continuous integration rather than implementing large-scale integration during the project completion.

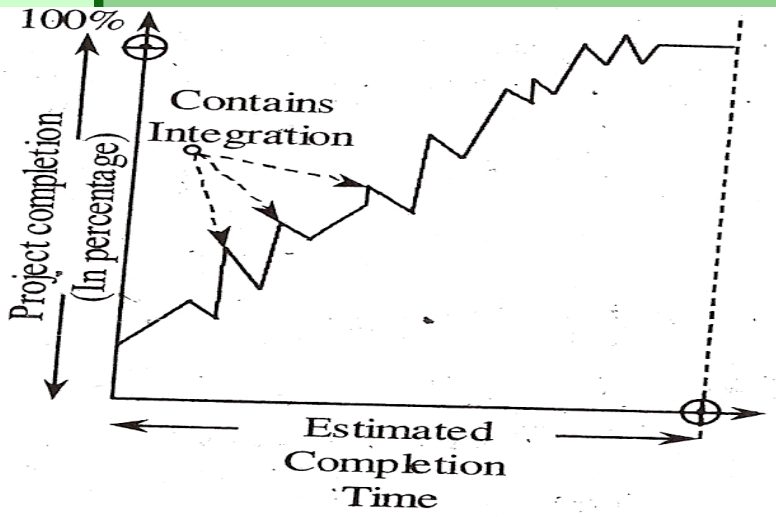


Figure (a): Modern Project Profile

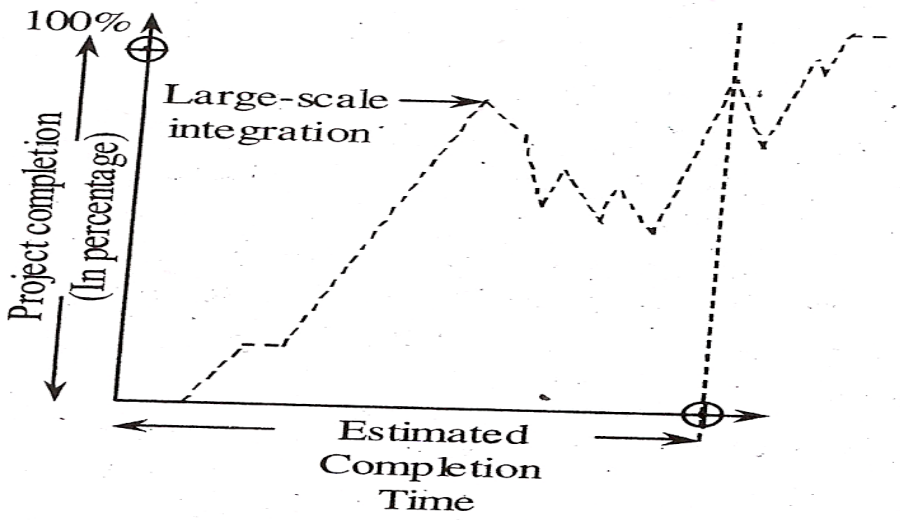


Figure (b): Traditional Project Profile

- Moreover, it produces feasible and a manageable design by delaying the ‘design breakage’ to the engineering phase, where they can be efficiently resolved. This can be one by making use of project demonstrations which forces integration into the design phase.
- With the help of this continuous integration incorporated in the iterative development process, the quality tradeoffs are better understood and the system features such as system performance, fault tolerance and maintainability are clearly visible even before the completion of the project.
- In the modern project profile, the distribution of cost among various workflows or project is completely different from that of traditional project profile as shown below:

Software Engineering Workflows	Conventional Process Expenditures	Modern process Expenditures
Management	5%	10%
Environment	5%	10%
Requirements	5%	10%
Design	10%	15%
Implementation	30%	25%
Assessment	40%	25%
Deployment	5%	5%
Total	100%	100%

As shown in the table, the modern projects spend only 25% of their budget for integration and Assessment activities whereas; traditional projects spend almost 40% of their total budget for these activities. This is because, the traditional project involve inefficient large-scale integration and late identification of design issues.

5.9.2 Early Risk Resolution

- In the project development lifecycle, the engineering phase concentrates on identification and elimination of the risks associated with the resource commitments just before the production stage. The traditional projects involve, the solving of the simpler steps first and then goes to the complicated steps, as a result the progress will be visibly good, whereas, the modern projects focuses on 20% of the significant requirements, use cases, components and risk and hence they occasionally have simpler steps.
- To obtain a useful perspective of risk management, the project life cycle has to be applied on the principles of software management. The following are the 80:20 principles.
- The 80% of Engineering is utilized by 20% of the requirements

- Before selecting any of the resources, try to completely understand all the requirement because irrelevant resource selection (i.e., resources selected based on prediction) may yield severe problems.
- 80% of the software cost is utilized by 20% of the components
- Firstly, the cost-critical components must be elaborated which forces the project to focus more on controlling the cost.
- 80% of the bugs occur because of 20% of the components
- Firstly, the reliability-critical components must be elaborated which give sufficient time for assessment activities like integration and testing, in order to achieve the desired level of maturity.
- 80% of the software scrap and rework is due to 20% if the changes.

- The change-critical components r elaborated first so that the changes that have more impact occur when the project is matured.
- 80% of the resource consumption is due to 20% of the components.
- Performance critical components are elaborated first so that, the trade-offs with reliability; changeability and cost-consumption can be solved as early as possible.
- 80% of the project progress is carried-out by 20% of the people
- It is important that planning and designing team should consist of best processionals because the entire success of the project depends upon a good plan and architecture.
- The following figure shows the risk management profile of a modern project.

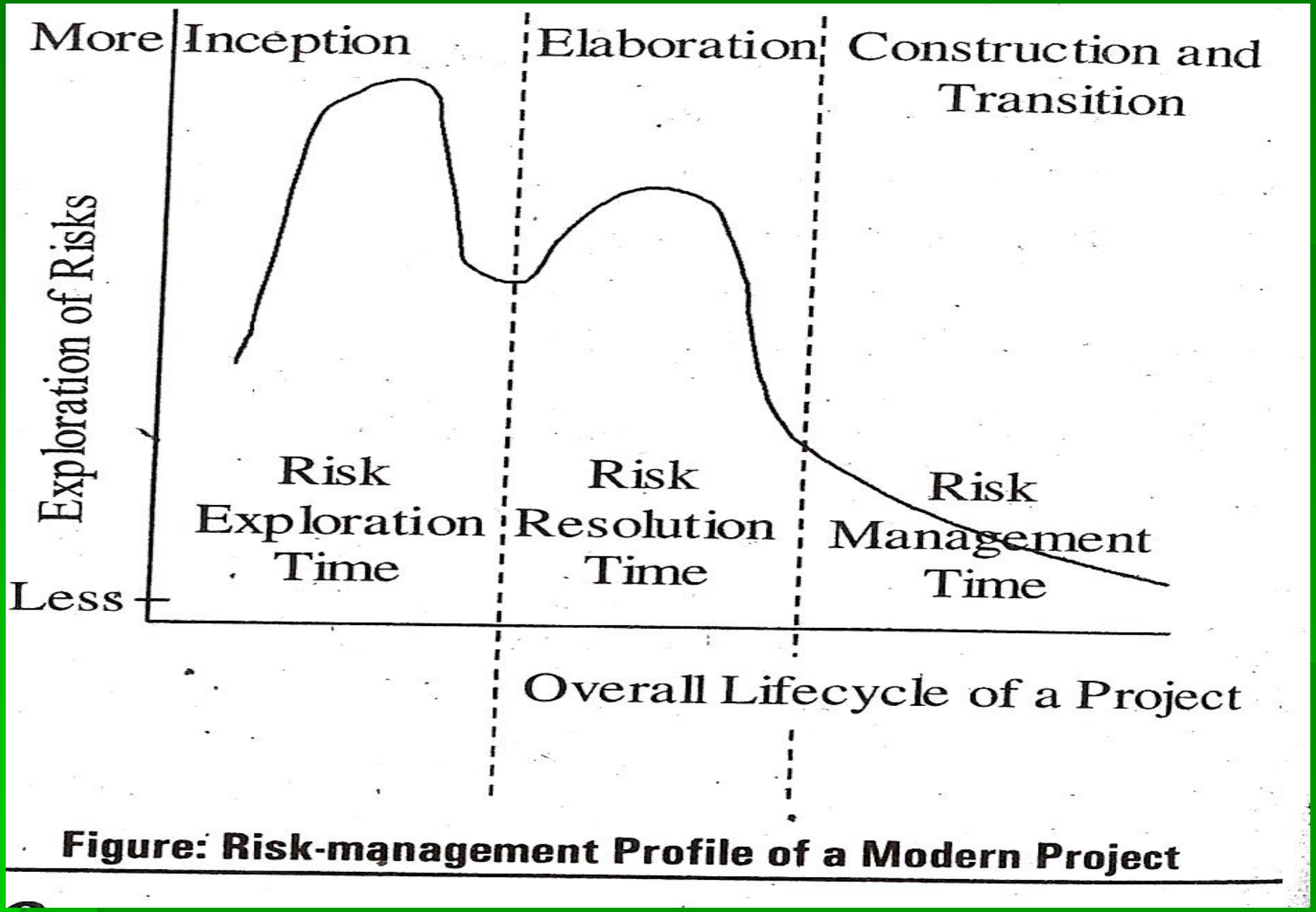


Figure: Risk-management Profile of a Modern Project

5.9.3 Evolutionary requirements

- The traditional methods divide the system requirements into subsystem requirements which in turn gets divided into component requirements. These component requirements are further divided into unit requirements. The reason for this systematic division is to simplify the traceability of the requirements.
- In the project life cycle the requirements and design are given the first and the second preference respectively. The third preference is given to the traceability between the requirement and the design components these preferences are given in order to make the design structure evolve into an organization so it parallels the structure of the requirements organization.
- Modern architecture finds it difficult to trace the requirements because of the following reasons.
  - Usage of Commercial components
  - Usage of legacy components
  - Usage of distributed resources
  - Usage of object oriented methods.
- Moreover, the complex relationships such as one-one, many-one, one-many, conditional, time-based and state based exists the requirements statement and the design elements.

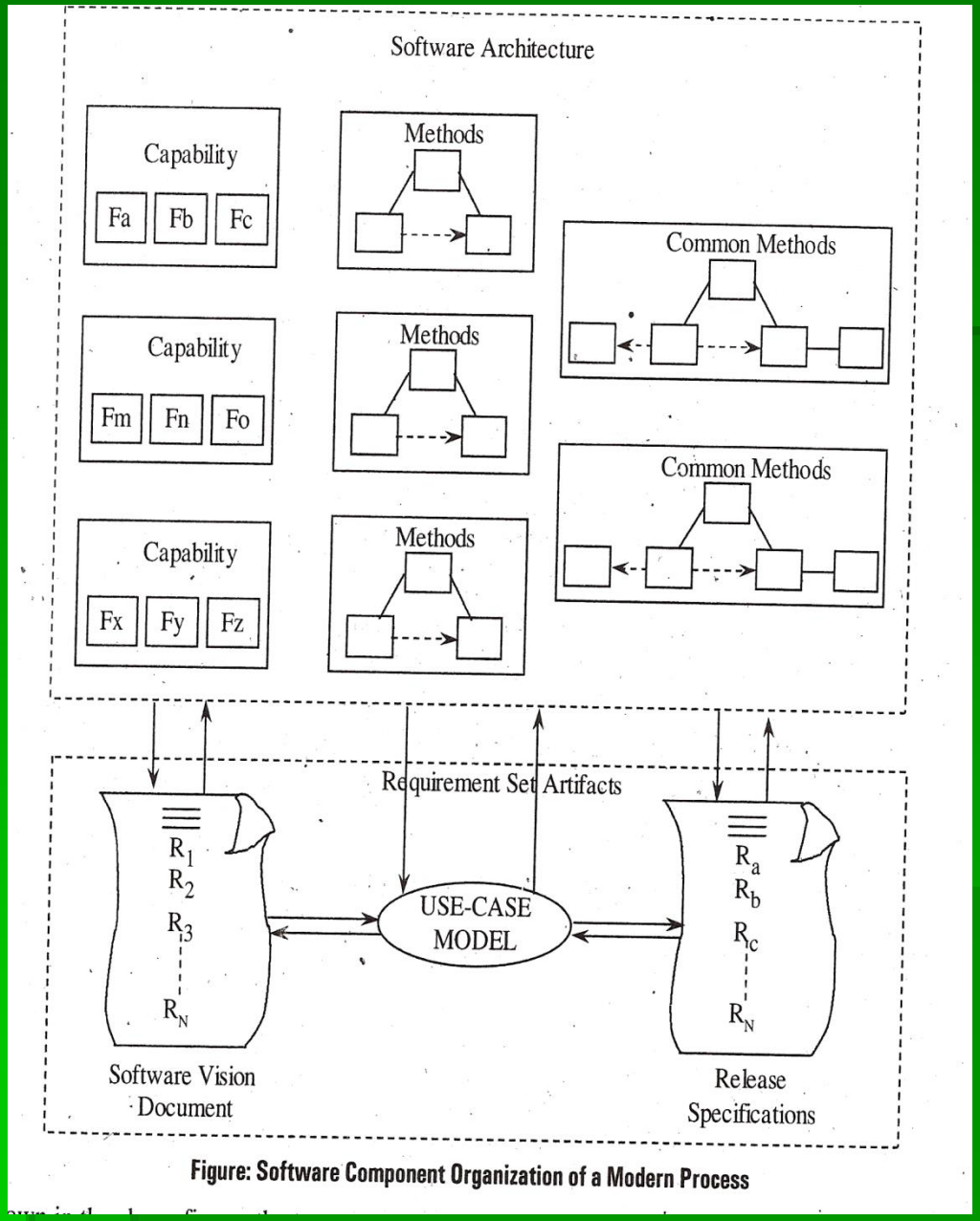


Figure: Software Component Organization of a Modern Process

As shown in the above figure, the top category system requirements are kept as the vision whereas, those with the lower category are evaluated. The motive behind theses artifacts is to gain fidelity with respect to the progress in the project lifecycle. This serves as a significant different from the traditional approach because, in traditional approach the fidelity is predicted early in the project life cycle.

5.9.4 Teamwork among stakeholders

- Most of the characteristics of the classic development process worsen the stakeholder relationship s which in turn makes the balancing of requirement product attributes and plans difficult. An iterative process which ahs a good relationship between the stakeholders mainly focuses on objective understanding by each and every individual stakeholder. This process needs highly skilled customers, users and monitors which have experience in both the application as well as software. Moreover, this process requires an organization whose focus is on producing a quality product and achieves customer satisfaction.
- The table below shows the tangible results of major milestones in a modern process.

Obvious result	Actual result
Demonstration at early stage reveals the design issued and uncertainty in a tangible form.	Demonstration firstly reveals the significant assets and risks associated with complicated software systems such that they can be worked out at the time of setting the life-cycle goals.
Non-Complaint design	Various perspective like requirements use cases etc are observed in order to completely understand the compliance.
Issues of influential requirements are reveals but without traceability	Both the requirement changes and the design trade-offs are considerably balanced.
The design is considered to be “guilty until its innocency is proved.	The engineering issues can be integrated into the succeeding iteration’s plans.

- From the above table, it can be observed that the progress of the project is not possible unless all the demonstration objectives are satisfied. This statement does not present the renegotiation of objectives, even when the demonstration results allow the further processing of trade offs present in the requirement, design, plans and technology.

- Modern iterative process that rely on the results of the demonstration need al its stakeholders to be well-educated and with a g good analytical ability so as to distinguish between the obviously negative results and the real progress visible. For example, an early determined design error can be treated as a positive progress instead to a major issue.

5.9.5 Principles of Software Management

- **Software management basically relies on the following principles, they are,**

1. ***Process must be based on architecture-first approach***

   If the architecture is focused at the initial stage, then there will be a good foundation for almost 20% of the significant stuff that are responsible for the overall success of the project. This stuff include the requirements, components use cases, risks and errors. In other words, if the components that are being involved in the architecture are well known then the expenditure causes by scrap and rework will be comparatively less.

2. ***Develop an iterative life-cycle process that identifies the risks at an early stage***

   An iterative process supports a dynamic planning framework that facilitates the risk management predictable performance moreover, if the risks are resolved earlier, the predictability will be more and the scrap and rework expenses will be reduced.

3. *After the design methods in-order to highlight components-based development.*

The quantity of the human generated source code and the customized development can be reduced by concentrating on individual components rather than individual lines-of-code. The complexity of software is directly proportional to the number of artifacts it contains that is, if the solution is smaller then the complexity associated with its management is less.

4. *Create a change management Environment*

Highly-controlled baselines are needed to compensate the changes caused by various teams that concurrently work on the shared artifacts.

5. *Improve change freedom with the help of automated tools that support round-trip engineering.*

The roundtrip-engineering is an environment that enables the automation and synchronization of engineering information into various formats. The engineering information usually consists requirement specification, source code, design models test cases and executable code. The automation of this information allows the teams to focus more on engineering rather than dealing with over head involved.

6. *Design artifacts must be captured in model based notation.*

The design artifacts that are modeled using a model based notation like UML, are rich in graphics and texture. These modeled artifacts facilitate the following tasks.

- **Complexity control**
- **Objective fulfillment**
- **Performing automated analysis**

7. *Process must be implemented or obtaining objective quality control and estimation of progress.*

The progress in the lifecycle as well as the quality of intermediately products must be estimated and incorporated into the process. This can be done with the help of well defined estimation mechanism that are directly derived from the emerging artifacts. These mechanisms provide detailed information about trends and correlation with requirements.

8. *Implement a Demonstration-based Approach for Estimation of intermediately Artifacts*

This approach involves giving demonstration on different scenarios. It facilitates earl integration and better understanding of design trade-offs. Moreover, it eliminates architectural defects earlier in the lifecycle. The intermediately results of this approach are definitive.

09. *The Points Increments and generations must be made based on the evolving levels of detail*

Here, the 'levels of detail' refers to the level of understanding requirements and architecture. The requirements, iteration content, implementations and acceptance testing can be organized using cohesive usage scenarios.

10. *Develop a configuration process that should be economically scalable*

The process framework applied must be suitable for variety of applications. The process must make use of processing spirit, automation, architectural patterns and components such that it is economical and yield investment benefits.

5.9.6 Best Practices Associated with software Management

- According to airline software council, there are about nine best practices associated with software management. Theses practices are implemented in order to reduce the complexity of the larger projects and to improve software management discipline.
- The following are the best practices of software management:

1. *Formal Risk Management:* Earlier risk management can be done by making use of iterative life cycle process that identifies the risks at early stage.
2. *Interface Settlement:* The interface settlement is one of the important aspects of architecture first approach because; obtaining architecture involves the selection of various internal and external interfaces that are incorporated into the architecture.
3. *Formal Inspections:* There are various defect removal strategies available. Formal inspection is one of those strategies. However this is the least important strategy because the cost associated with human recourses is more and is defect detection rate for the critical architecture defects is less.

4. *Management and scheduling based on metrics:* This principle is related to the model based approach and objective quality control principles. It states to use common notations fro the artifacts so that quality and progress can be easily measured.
5. *Binary quality Gates at the inch-pebble level:* The concept behind this practice is quite confusing. Most of the organizations have misunderstood the concept and have developed an expensive and a detailed plan during the initial phase of the lifecycle, but later found the necessity to change most of their detailed plan due to the small changes in requirements or architectural. This principle states that first start planning with an understanding of requirements and the architecture. Milestones must be established during engineering stage and inch-pebble must be followed in the production stage.
6. *Plan versus visibility of progress throughout the progress:* This practice involves a direct communication between different team members of a project so that, they can discuss the significant issues related to the project as well as notice the progress of the project in-comparison to their estimated progress.

7. *Identifying defects associated with the desired quality:* This practice is similar to the architecture-first approach and objective quality control principles of software management. It involves elimination of architectural defects early in the life-cycle, thereby maintaining the architectural quality so as to successfully complete the project.
8. *Configuration management:* According to Airline software council, configuration management serves as a crucial element for controlling the complexity of the artifacts and for tracing the changes that occur in the artifacts. This practice is similar to the change management principle of software management and prefers automation of components so as to reduce the probability of errors that occur in the large-scale projects.

9.Disclose management accountability: The entire managerial process is disclosed to al the people dealing with the project.

5.10 NEXT GENERATION SOFTWARE ECONOMICS

5.10.1 Next generation software cost models

- In comparison to the current generation software cost modes, the next generation software cost models should perform the architecture engineering and application production separately. The cost associated with designing, building, testing and maintaining the architecture is defined in terms of scale, quality, process, technology and the team employed.
- After obtaining the stable architecture, the cost of the production is an exponential function of size, quality and complexity involved.
- The architecture stage cost model should reflect certain diseconomy of scale (exponent less than 1.0) because it is based on research and development-oriented concerns. Whereas the production stage cost model should reflect economy of scale (exponent less than 1.0) for production of commodities.

- **The next generation software cost models should be designed in a way that, they can assess larger architectures with economy of scale. Thus, the process exponent will be less than 1.0 at the time of production because large systems have more automated proves components and architectures which are easily reusable.**
- **The next generation cost model developed on the basis of architecture-first approach is shown below.**
- **At architectural engineering Stage**
  - A Plan with less fidelity and risk resolution
  - It is technology or schedule-based
  - It has contracts with risk sharing
  - Team size is small but with experienced professionals.
  - The architecture team, consists of small number of software engineers
  - The application team consists of small number of domain engineers.
  - The output will be an executable architecture, production and requirements
  - The focus of the architectural engineering will be on design and integration of entities as well as host development environment.
  - It contains two phases they are inspection and elaboration.

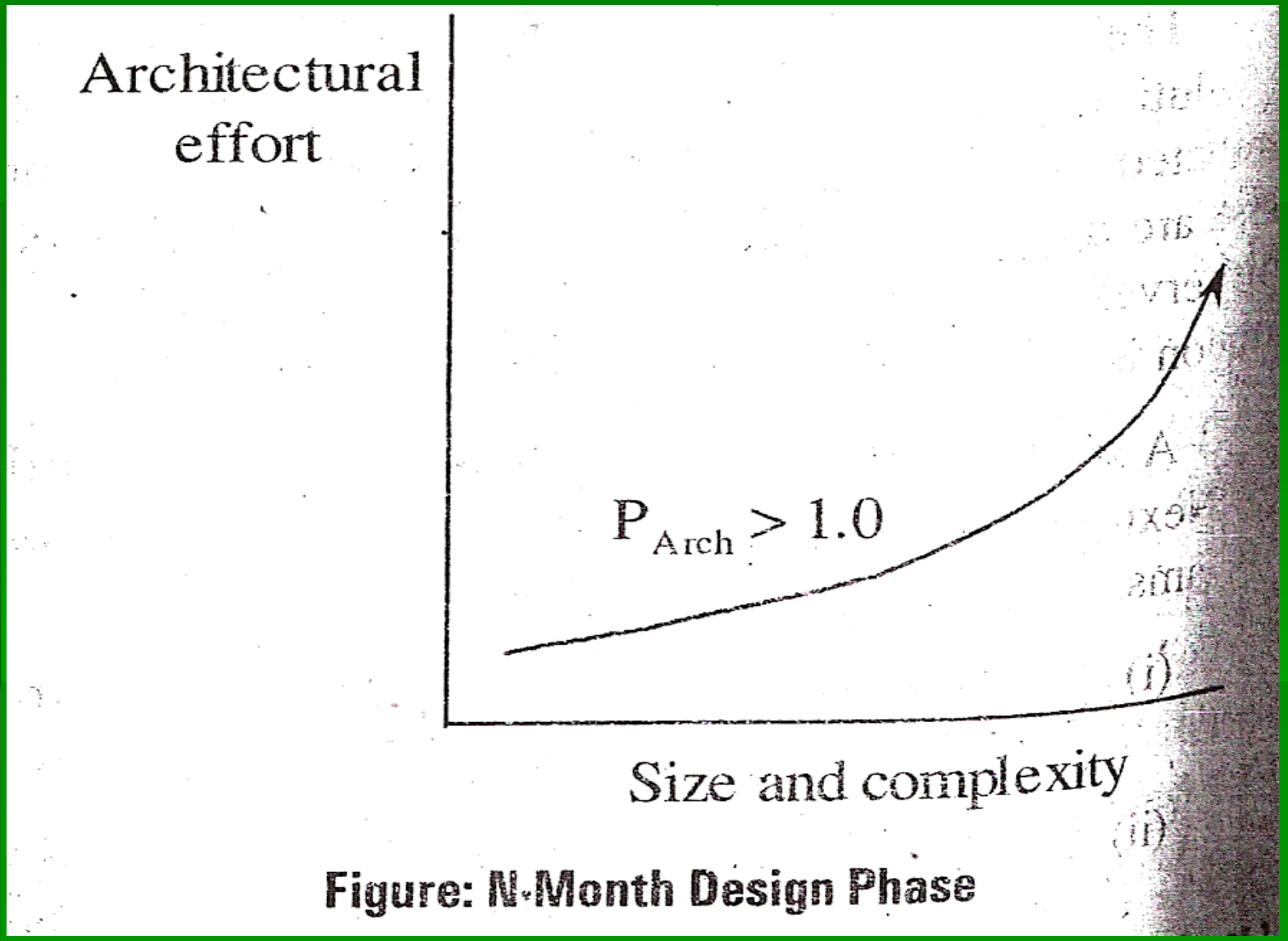


Figure: N-Month Design Phase

- At Application production stage
  - A plan with high fidelity and lower risk
  - It is cost-based
  - It has fixed-priced contracts
  - Team size is large and diverse as needed.
  - Architecture team consists of a small number of software engineers.
  - The Application team may have nay number of domain engineers.
  - The output will be a function which is deliverable and useful, tested baseline and warranted quality.
  - The focus of the application production will be on implementing testing and maintaining target technology.
  - It contains two phases they are construction and transition

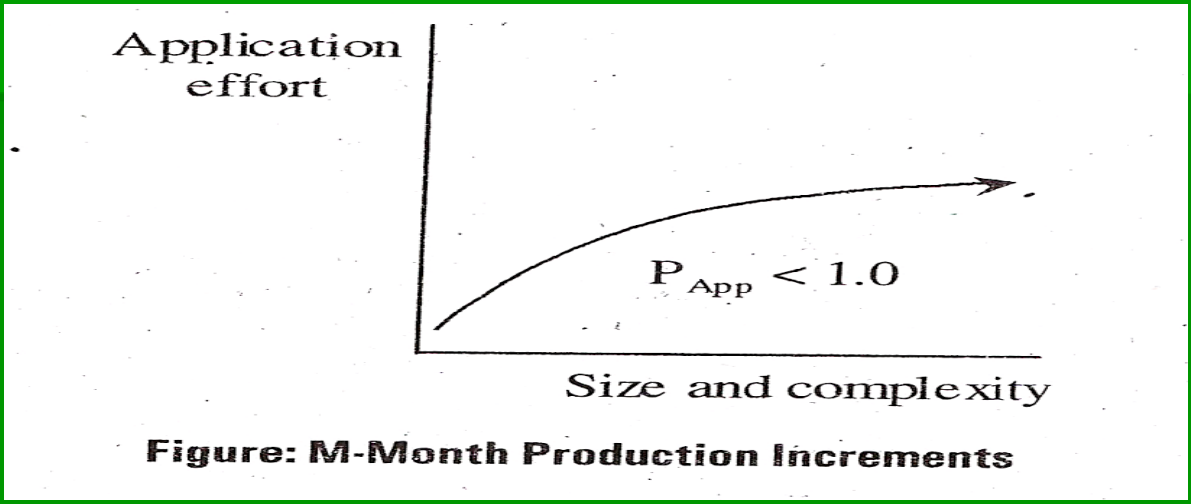


Figure: M-Month Production Increments

Total Effort = Func(TechnologyArch, ScaleArch, Quality Arch, Process Arch) + Func(TechnologyApp, ScaleApp, Quality App, Process App)

Total Time = Func(ProcessArch, EffortArch) + Func(ProcessApp, EffortApp,)

- **The next generation infrastructure and environment automated various management activities with low effort. It relieves many of the sources of diseconomy of scale by reusing the common processes that are repetitive in a particular project. It also reuses the common outcomes of the project. The prior experience and matured processes utilized in these types of models eliminate the scrap rework sources. Here, the economics of scale will be affected.**
- **The architecture and applications of next generation cost models have difference scales and sized which represents the solution space. The size can be computed inters of SLOC or megabytes of executable code while the scale can be computed in 0-terms of components, classes, processes or nodes. The requirement or use cases of solution space are different from that of a problem space. Moreover, there can be more than one solution to a problem. Where cost serves as a key discriminator. The cost estimates must be determined to find an optimal solution. If an optional solution is not found then different solution s need to be selected or to change the problem statement.**

- A strict notation must be applied for design artifacts so, that the prediction of a design scale can be improved. The Next-generation software cost model should automate the process of measuring design scale directly from UML diagrams. There should be two major improvements. There are,
  - Separate architectural engineering stage from application production stage. This will yield greater accuracy and more precision of lifecycle estimate.
  - The use of rigorous design notations. This will enable the automation and standardization of scale measure so that they can be easily traced which helps to determine the total cost associated with production.
- The next generation software process has two potential breakthroughs, they are,
  - Certain integrated tools would be available that automates the information transition between the requirements, design, implementation and deployment elements. These tools facilitate roundtrip engineering between various artifacts of engineering.
  - It will reduce the four sets of fundamental technical artifacts into three sets. This is achieved by automating the activities related to human-generated source code so as to eliminate the need fro a separate implementation set.

5.10.2 'An organizational manager should strive for making the transition to a modern process'.

- The transition to a modern process should be made based on the following quotations laid by Boehm.

1. **Identifying and solving a software problem in the design phase is almost 100 times cost effective than solving the same problem after delivery.**

   This quotation or metric serves as a base for most software processes. Modern processes, component-based development techniques and architectural frameworks mainly focuses on enhancing this relationship. The architectural errors are solved by implementing an architecture-first approach. Modern process plays a crucial role in identification of risks

2. Software Development schedules can be compressed to a Maximum of 25 percent

If we want a reduction in the scheduled time, then we must increase the personnel resources which inturn increases the management overhead. The management overhead, concurrent activities scheduling, sequential activities conservation along some resource constraints will have the flexibility limit of about 25 percent.

This metric must be acceptable by the engineering phase which consists of detailed system content if we have successfully completed the engineering then compression in the production stage will be automatically flexible. The concurrent development must be possible irrespective of whether a business organization implements the engineering phase over multiple projects or whether a project implements the engineering phase over multiple incremental stages.

3. The maintenance cost will be almost double the development cost

Most o the experts in the software industry find it difficult to maintain the software than development. The ratio between development and maintenance can be measured by computing productivity cost. One of the interesting fact of iterative development is that the dividing line between the development and maintenance is vanishing. Moreover, a good iterative process and an architecture will cause the reduction in the scrap and rework levels so this ratio (i.e.,) 2:1 can be reduced to 1:1.

4. **Both the software development cot and the maintenance cost are dependent on the number of lines in the source code.**

This metric was applicable to the conventional cost models which were lacking in-terms of commercial components, reusing techniques, automated code generators etc. The implementation of commercial components, reusing techniques and automated code generators will make this metric inappropriate. However, the development cost is still dependent on the commercial components, reuse technique and automatic code generators and their integration.

The next-generation cost models should focus more on the number of components and their integration efforts rather than on the number of lines of code.

5. **Software productivity mainly relies on the type of people employed**

The personal skills, team work ability and the motivation of employees are the crucial factors responsible for the success and the failure of any project. The next-generation cost models failure should concentrate more on employing a highly skilled team of professionals at engineering stage.

6. **The ratio of software to hardware cost is increasing.**

   As the computers are becoming more and more popular, the need for software an hardware applications is also increasing. The hardware components are becoming cheaper whereas, the software applications are becoming more complicated as a result, highly skilled professionals needed for development and controlling the software applications, the in turn increases the cost. In 1955 the software to hardware cost ratio was 15:85 and in 1985 this ratio was 85:15. This ratio continuously increases with respect to the need for variety of software applications. Certain software applications have already been developed which provides automated configuration control and analysis of quality assurance. The next-generation cost models must focus on automation of production and testing.

7. **Only 15% of the overall software development is dedicated process to programming.**

- The automation and reusability of codes have lead to the reduction in programming effort. Earlier in 1960s, the programming staff was producing about 200 machine instructions per month and in 1970s and 1980s, the machine instruction count has raised to about 1000 machine instructions. Now as days, programmers are able to produce several thousand instructions without even writing few hundreds of them

8. **Software system and products cost three times the cost associated with individual software programs per SLOC software-system products cost 9 times more than the cost of individual software program.**

   In the software development, the cost of each instruction depends upon the complexity of the software. Modern processes and technologies must reduce this diseconomy of scale. The economy of the scale must be achievable under the customer specific software systems with a common architecture, common environment and common process.

9. **60% of Errors are caught by walkthrough**

   The walkthrough and other forms of human inspection catch only the surface and style issues. However, the critical issues are not caught by the walkthroughs so, this metric doesn't prove to the reliable.

10. **Only 20% of the contributors are responsible for the 80% of the contributions.**

    This metric is applicable to most of the engineering concepts such as 80:20 principles of software project management. The next generation software process must facilitate the software organizations in achieving economic scale.

5.11 MODERN PROCESS TRANSITIONS

5.11.1 Indications of a successful project transition to a modern culture

- Several indicators are available that can be observed in order to distinguish projects that have made a genuine cultural transition from projects that only pretends.
- The following are some rough indicators available.

1. **The lower-level managers and the middle level managers should participate in the project development**

   Any organization which ha an employee count less than or equal to 25 does not need to have pure managers. The responsibility of the managers in this type of organization will be similar to that of a project manager. Pure managers are needed when personal resources exceed 25. Firstly, these managers understand the status of the project them, develop the plans and estimate the results. The manager should participate in developing the plans. This transition affects the software project managers.

2. **Tangible design and requirements**

   The traditional processes utilize tons of paper in order to generate the documents relevant to the desired project. Even the significant milestones of a project are expressed via documents. Thus, the traditional process spends most of their crucial time on document preparation instead of performing software development activities.

   An iterative process involves the construction of systems that describe the architecture, negotiates the significant requirements, identifies and resolves the risks etc. These milestones will be focused by all the stakeholders because they show progressive deliveries of important functionalities instead of documental descriptions about the project. Engineering teams will accept this transition of environment from to less document-driven while conventional monitors will refuse this transition.

3. **Assertive Demonstrations are prioritized**

   The design errors are exposed by carrying-out demonstrations in the early stages of the life cycle. The stake holders should not over-react to these design errors because overemphasis of design errors will discourage the development organizations in producing the ambitious future iterating. This does not mean that stakeholders should bare all these errors. Infact, the stakeholders must follow all the significant steps needed for resolving these issues because these errors will sometimes lead to serious down-fall in the project.

   This transition will unmark all the engineering or process issues so, it is mostly refused by management team, and widely accepted by users, customers and the engineering team.

4. **The performance of the project can be determined earlier in the life cycle.**

The success and failure of any project depends on the planning and architectural phases of life cycle so, these phases must employ high-skilled professionals. However, the remaining phases may work well an average team.

5. **Earlier increments will be adolescent**

The development organizations must ensure that customers and users should not expect to have good or reliable deliveries at the initial stages. This can be done by demonstration of flexible benefits in successive increments. The demonstration is similar to that of documentation but involves measuring of changes, fixes and upgrades based on the objectives so as to highlight the process quality and future environments.

6. **Artifacts tend to be insignificant at the early stages but proves to be the most significant in the later stages**

   The details of the artifacts should not be considered unless a stable and a useful baseline is obtained. This transition is accepted by the development team while the conventional contract monitors refuse this transition.

7. **Identifying and Resolving of real issues is done in a systematic order**

   The requirements and designs of any successful project arguments along with the continuous negotiations and trade-offs. The difference between real and apparent issued of a successful project can easily be determined. This transition may affect any team of stakeholders.

8. Everyone should focus on quality assurance

The software project manager should ensure that quality assurance is integrated in every aspect of project that is it should be integrated into every individuals role, every artifact, and every activity performed etc. There are some organizations which maintains a separate group of individuals know as quality assurance team, this team would perform inspections, meeting and checklist inorder to measure quality assurance. However, this transition involves replacing of separate quality assurance team into an organizational teamwork with mature process, common objectives and common incentives. So, this transition is supported by engineering teams and avoided by quality assurance team and conventional managers.

9. **Performance issues crop up earlier in the projects life cycle**

   Earlier performance issues are a mature design process but resembles as an immature design. This transition is accepted by development engineers because it enables the evaluation of performance tradeoffs in subsequent releases.

10. **Automation must be done with appropriate investments**

    Automation is the key concept of iterative development projects and must be done with sufficient funds. Moreover, the stakeholders must select an environment that supports iterative development. This transition is mainly opposed by organizational managers.

11. **Good software organizations should have good profit margins.**

    Most of the contractors for any software contracting firm focus only on obtaining their profit margins beyond the acceptable range of 5% and 15%. They don't look for the quality of finished product as a result, the customers will be affected. For the success of any software industry, the good quality and at a reasonable rate them, customer will not worry about the profit the contractor has made. The bad contractors especially in a government contracting firm will be against this transition.

5.11.2 Characteristics of conventional and iterative software development Process

- The characteristics of the conventional software process are listed below:
  1. It evolves in the sequential order (requirement design-code-test).
  2. It gives the same preference to all the artifacts, components, requirements etc.
  3. It completes all the artifacts of a stage before moving to the other stage in the project life cycle.
  4. It achieves traceability with high-fidelity for al the artifacts present at each life cycle stage.
- The characteristics of the modern iterative development process framework are listed below:
  1. It continuously performs round-trip engineering of requirements, design, coding and testing at evolving levels of abstraction.
  2. It evolves the artifacts depending on the priorities of the risk management.
  3. It postpones the consistency analysis and completeness of the artifacts to the later stages in the life cycle.
  4. It achieves the significant drives (i.e. 20 percent) with high-fidelity during the initial stages of the life cycle.