Unit-V

PROJECT CONTROL & PROCESS INSTRUMENTATION

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

Need for Software Metrics:

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

INDICATORS:

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

Two types of indicators are:

(i) Management indicators.

(ii) Quality indicators.

Management Indicators

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

Quality Indicators

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

SEVEN CORE METRICS OF SOFTWARE PROJECT

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

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

Seven core metrics related to project control:

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

MANAGEMENT INDICATORS:

Work and progress

This metric measure the work performed over time. Work is the effort to be accomplished to complete a certain set of tasks. The various activities of an iterative development project can be measured by defining a planned estimate of the work in an objective measure, then tracking progress (work completed overtime) against that plan.

The default perspectives of this metric are:

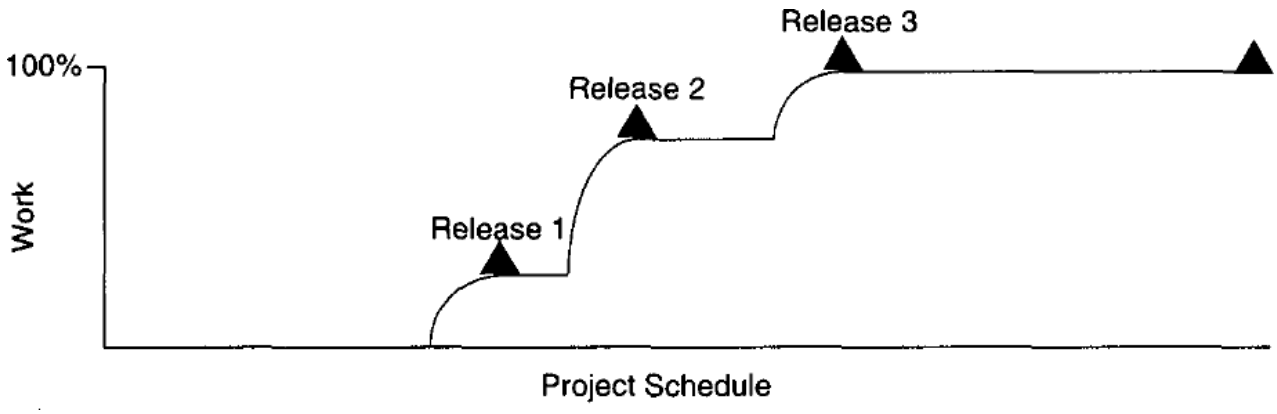
Software architecture team: - Use cases demonstrated.

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

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

Software management team: - milestones completed.

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



Budgeted cost and expenditures

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

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

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

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

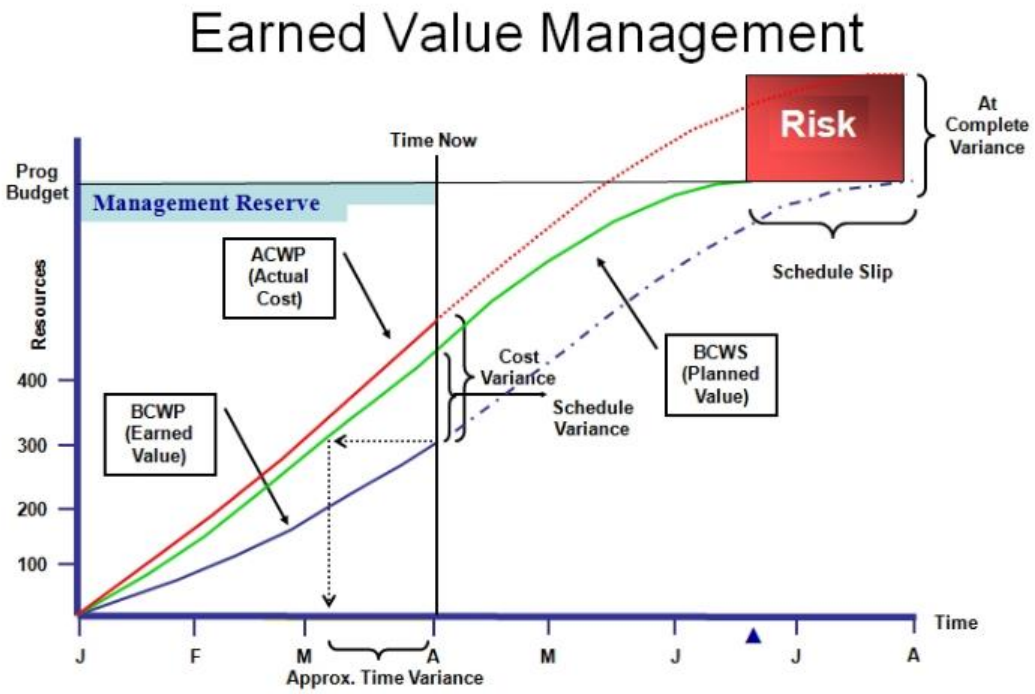
Earned value: It is the value that represents the planned cost of the actual progress.

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

Schedule variance: It is the difference between the planned cost and the earned value.

Of all parameters in an earned value system, actual progress is the most subjective

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



Among project managers, Earned Value is one of the most demanding requirements for management tools. In the graph shown above, you can visualize what would be the Earned Value, in relation to the estimated budget in the planning and the costs actually allocated during the development of the project, as well as the risks that are taken when taking part in it.

Staffing and team dynamics

This metric measure the personnel changes over time, which involves staffing additions and reductions over time. An iterative development should start with a small team until the risks in the requirements and architecture have been suitably resolved. Depending on the overlap of iterations and other project specific circumstances, staffing can vary. Increase in staff can slow overall project progress as new people consume the productive team of existing people in coming up to speed. Low

attrition of good people is a sign of success. The default perspectives of this metric are people per month added and people per month leaving.

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

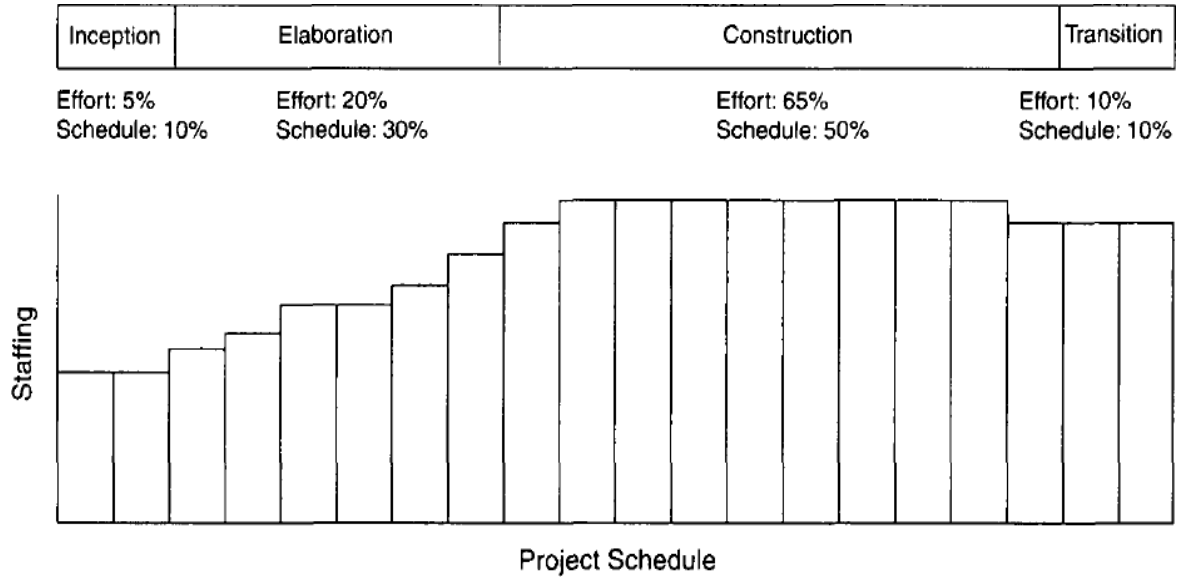


Fig: staffing and Team dynamics

QUALITY INDICATORS:

Change traffic and stability:

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

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

.

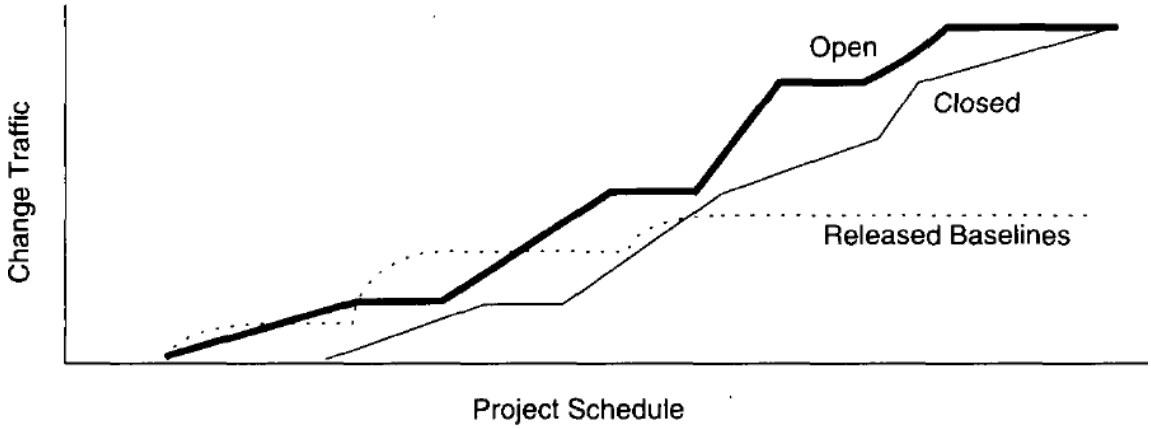


Fig: Change traffic and stability

Breakage and modularity

This metric measures the average breakage per change over time. Breakage is defined as the average extent of change, which is the amount of software baseline that needs rework and measuredin source lines of code, function points, components, subsystems, files or other units.

Modularity is the average breakage trend over time. This metric can be collected by rework SLOC per change, by change type, by release, by components and by subsystems.

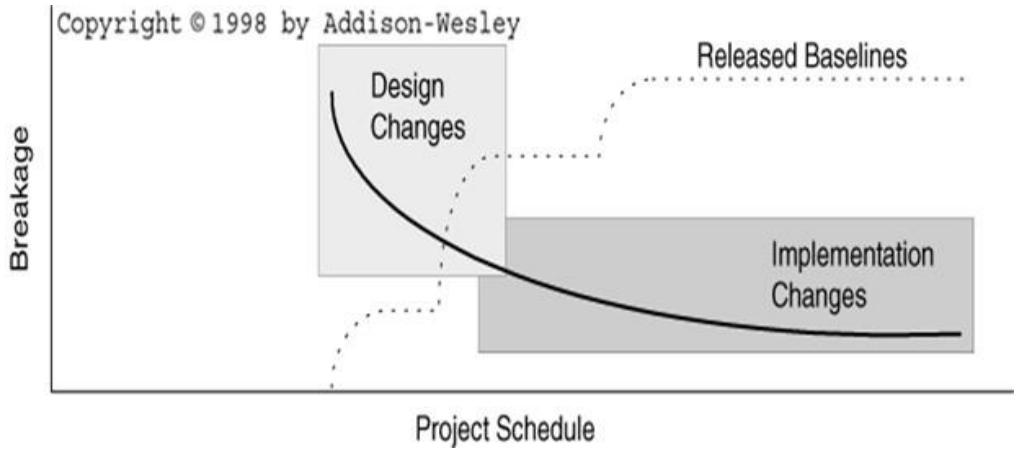


Fig: Modularity expectation over a healthy project's life cycle.

Rework and Adaptability:

This metric measures the average rework per change over time. Rework is defined as the average cost of change which is the effort to analyze, resolve and retest all changes to software baselines. Adaptability is defined as the rework trend over time. This metric provides insight into rework measurement. All changes are not created equal. Some changes can be made in a staff- hour, while others take staff-weeks. This metric can be collected by average hours per change, by change type, by release, by components and by subsystems.

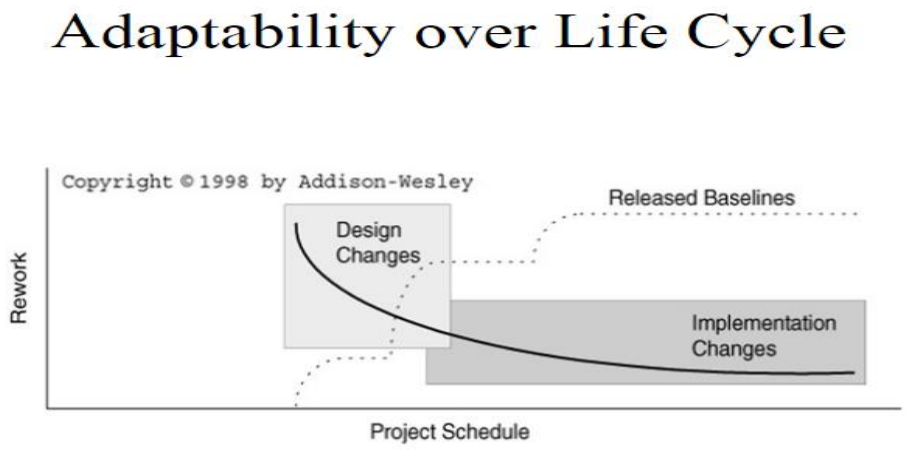


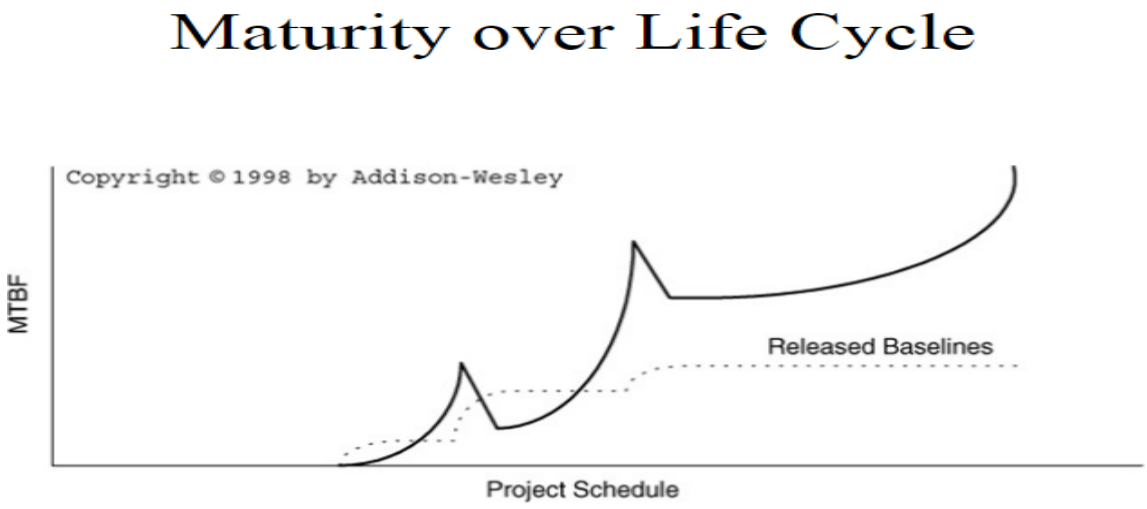
Fig: Adaptability expectation over a healthy project's life cycle

MTBF and Maturity:

This metric measures defect rater over time. MTBF (Mean Time Between Failures) is the average usage time between software faults. It is computed by dividing the test hours by the number of type 0 and type 1 SCOs. Maturity is defined as the MTBF trend over time.

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

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



Lifecycle 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 from the evolving product rather than from the artifacts.

• They provide insight into the waste generated by the process. Scrap and rework metrics are a standard measurement perspective of most manufacturing processes.

• They recognize the inherently dynamic nature of an iterative development process. Rather than focus on the value, they explicitly concentrate on the trends or changes with respect to time.

• The combination of insight from the current value and the current trend provides tangible indicators for management action.

The actual values of these metrics can vary widely across projects, organizations, and domains. The relative trends across the project phases, however, should follow the general pattern shown in Table 13-3. A mature development organization should be able to describe metrics targets that are much more definitive and precise for its line of business and specific processes.

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