Unit-V

***Chapter*-1**

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.

1.1 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.

1.1.1 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.

1.1.2 Quality Indicators

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

1.2 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

1.2.1 MANAGEMENT INDICATORS:

1.2.1.1 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

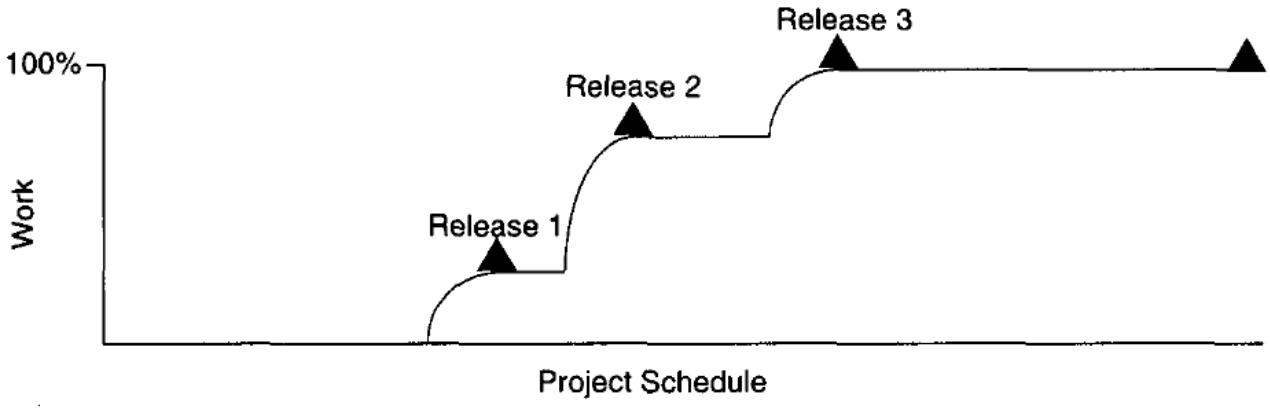


Fig: work and progress

1.2.1.2 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.

1.2.1.3 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: work and progress

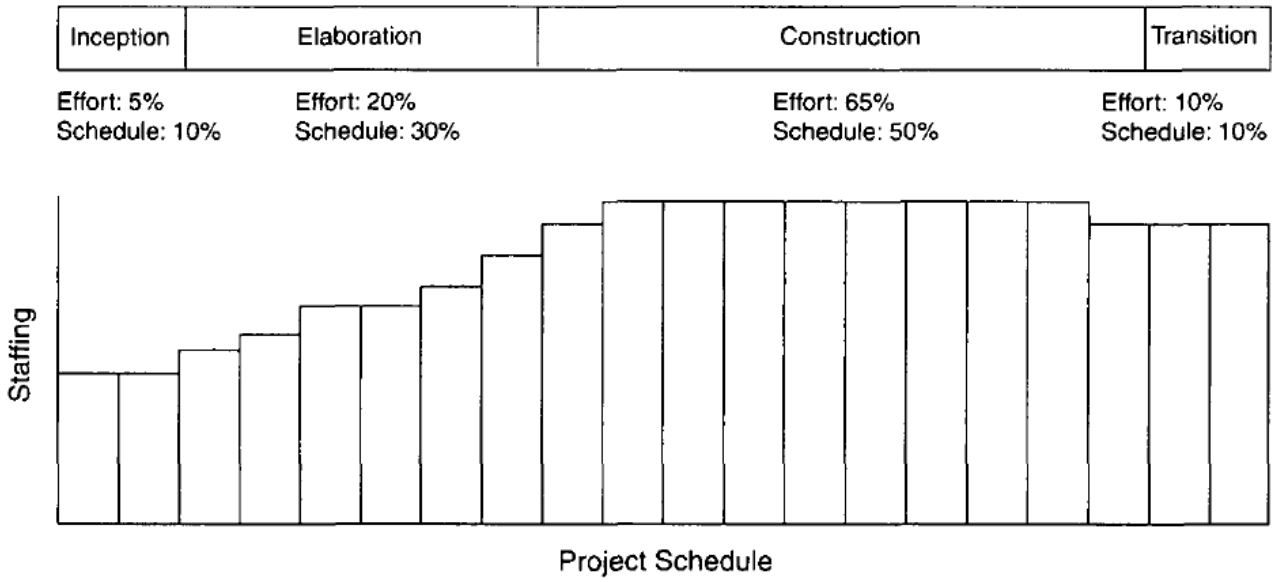


Fig: staffing and Team dynamics

1.2.2 QUALITY INDICATORS:

1.2.2.1 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

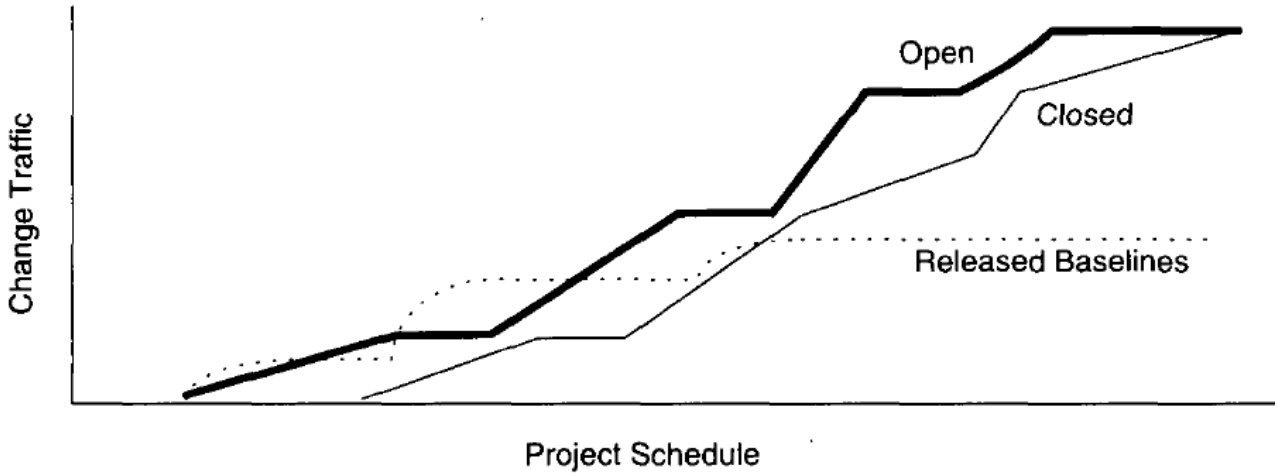


Fig: Change traffic and stability

1.2.2.2 Breakage and modularity

This metric measures the average breakage per change over time. Breakage is defined as the

average extent of change, which is the amount of software baseline that needs rework and measured in source lines of code, function points, components, subsystems, files or other units.

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

1.2.2.3 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.

1.2.2.4 *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.

1.3 LIFE -CYCLE EXPECTATIONS:

The default pattern of life-cycle metrics evolution is as follows:

table

1.4 METRICS AUTOMATION:

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

SPCP:

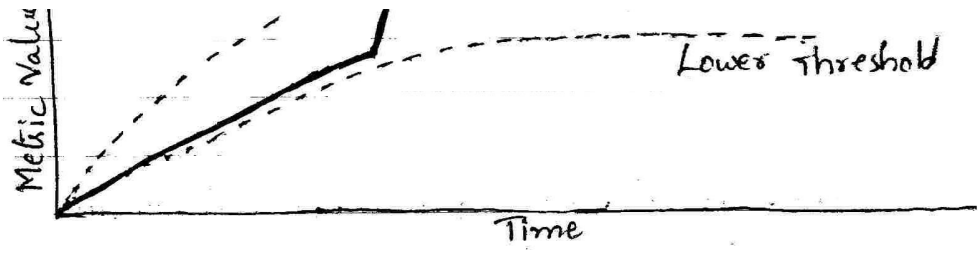
To implement a complete SPCP, the following are necessary.

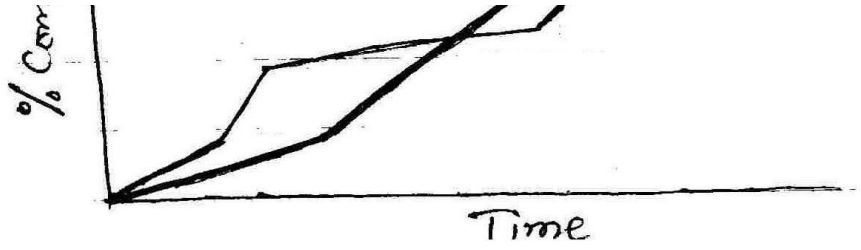
- Metrics primitives - trends, comparisons and progressions
- A graphical user interface.
- Metrics collection agents
- Metrics data management server
- Metrics definitions - actual metrics presentations for requirements progress, implementation progress, assessment progress, design progress and other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

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

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

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





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

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

1 Mark Questions

1. What are the seven core metrics?
2. What is meant by management indicators?
3. What is meant by progress?
4. What is meant by staffing?
5. What is meant by team dynamic?
6. In quality indicators metrics are developed in?
7. What is meant by breakage?
8. What is meant by modularity?
9. What is meant by rework?
10. What is meant by adaptability?
11. Expand MTFB?
12. What is meant by maturity?
13. Write any two life cycle expectation?
14. How can we reduce the complexity?
15. What are the dimensions that metrics contain?
16. On which programs the 7 core metrics are based on?
17. What are the 3 fundamentals sets of management metrics?
18. What are the use cases that are present in staffing?
19. What are the use case that are present in team dynamics?
20. What are specific monitors?

10 Mark Questions

1. Define metric. Discuss seven core metrics for project control and process instrumentation with suitable examples?

2. List out the three management indicators that can be used as core metrics on software projects. Briefly specify meaning of each?

3. Explain the various characteristics of good software metric. Discuss the metrics Automation using appropriate example?

4. Explain about the quality indicators that can be used as core metrics on software projects.

5. Explain Management Indicators with suitable example?

6. Define MTBF and Maturity. How these are related to each other?

7. Briefly explain about Quality Indicators?

8. Write short notes on Pragmatic software metrics?

Next Generation cost modes:

Next-generation software cost models should estimate large-scale architectures with economy of scale. This implies that the process exponent during the production stage will be less than 1.0. My reasoning is that the larger the system, the more opportunity there is to exploit automation and to reuse common processes, components, and architectures.

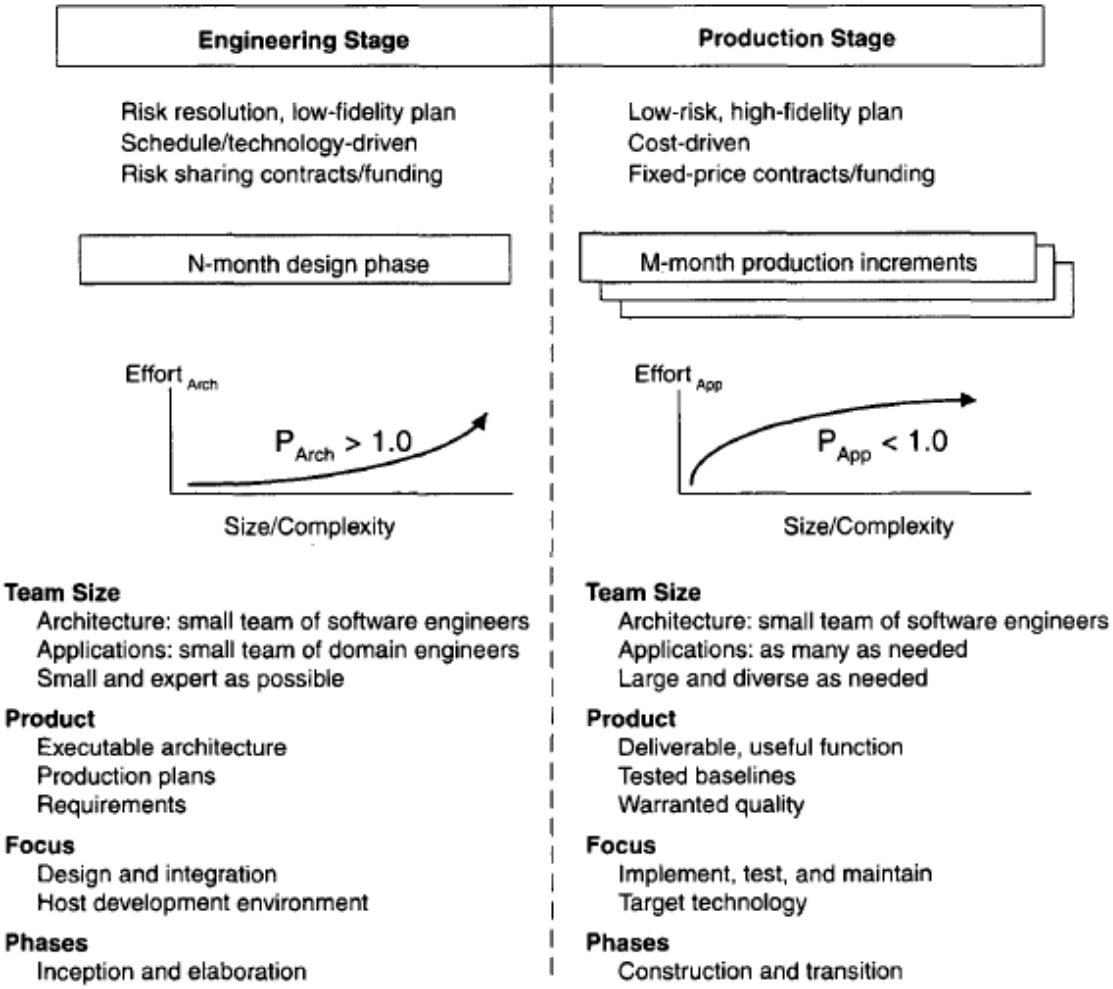
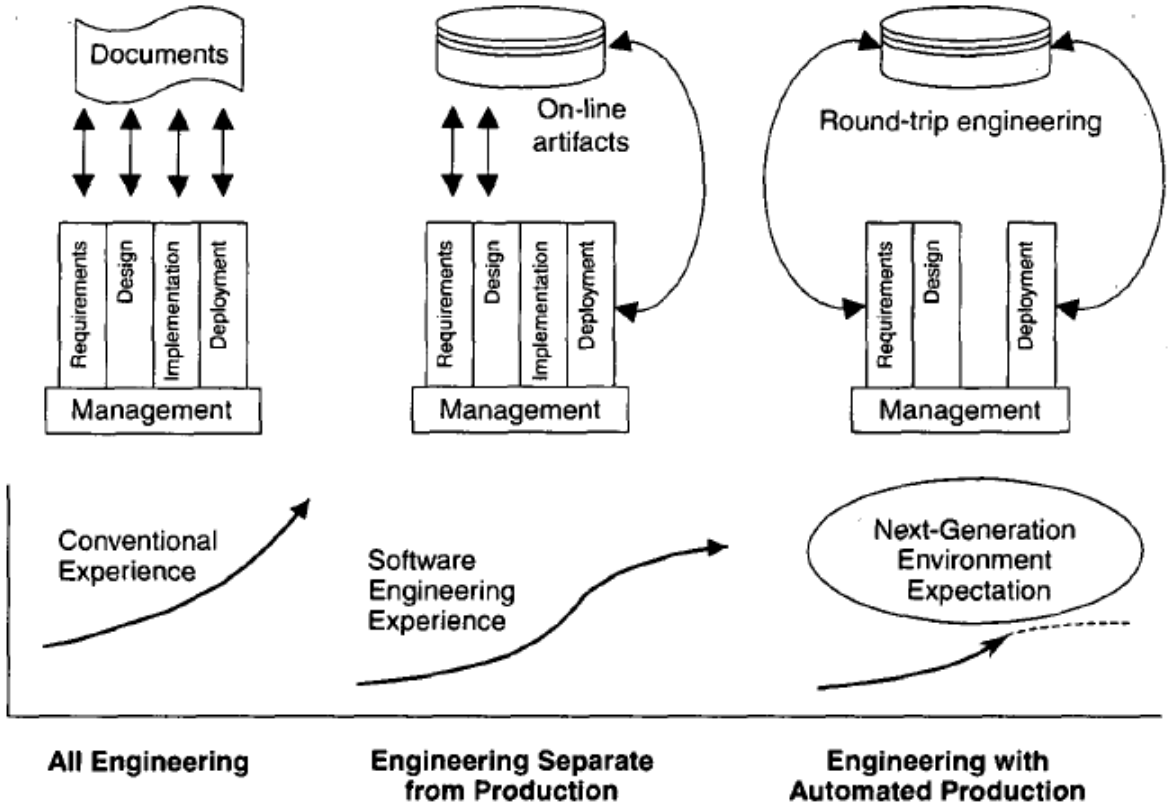


FIGURE 16-1. *Next-generation cost models*

- Quantifying the scale of the software architecture in the engineering stage is an area ripe for research. Over the next decade, two breakthroughs in the software process seem possible, both of them realized through technology advances in the supporting environment.
- The first breakthrough would be the availability of integrated tools that automate the transition of information between requirements, design, implementation, and deployment elements. These tools would allow more comprehensive



E 16-3. *Automation of the construction process in next-generation environments*

- Automation of the construction process in next-generation environments round-trip engineering among the engineering artifacts.
- The second breakthrough would focus on collapsing today's four sets of fundamental technical artifacts into three sets by automating the activities associated with human-generated source code, thereby eliminating the need for a separate implementation set.
- This technology advance, illustrated in Figure 16-3, would allow executable programs to be produced directly from UML representations without human intervention. Visual modeling tools can already produce code subsets from UML models, but producing complete subsets is still in the future.

- Some of today's popular software cost models are not well matched to an iterative software process focused an architecture-first approach
- Many cost estimators are still using a conventional process experience base to estimate a modern project profile

- A next-generation software cost model should explicitly separate architectural engineering from application production, just as an architecture-first process does.

Two major improvements in next-generation software cost estimation models:

- Separation of the engineering stage from the production stage will force estimators to differentiate between architectural scale and implementation size.
- Rigorous design notations such as UML will offer an opportunity to define units of measure for scale that are more standardized and therefore can be automated and tracked.

Modern Software Economics:

Changes that provide a good description of what an organizational manager should strive for in making the transition to a modern process:

1. Finding and fixing a software problem after delivery costs 100 times more than fixing the problem in early design phases
2. You can compress software development schedules 25% of nominal, but no more.
3. For every $1 you spend on development, you will spend $2 on maintenance.
4. Software development and maintenance costs are primarily a function of the number of source lines of code
5. Variations among people account for the biggest differences in software productivity.
6. The overall ratio of software to hardware costs is still growing – in 1955 it was 15:85; in 1985 85:15
7. Only about 15% of software development effort is devoted to programming
8. Software systems and products typically cost 3 times as much per SLOC as individual software programs.
9. Walkthroughs catch 60% of the errors.
10. 80% of the contribution comes from 20% of the contributors.