3.1-Software Metrics

To control a project effectively, it is imperative to measure the activities and processes that comprise the project. By measuring processes, you can easily evaluate and improve them and produce quality work product, "**You can't control what you can't measure**".

Continuous improvement in processes is an important goal of project management. Continuous improvement is required to evolve best practices that accelerate the speed of development while keeping a strong focus on meeting project schedules and quality targets. To understand processes you need to quantify and measure the entities that comprise a process. Software entities such as **delays**, **effort**, **personnel skills**, and **quality** are abstract entities. However, they still need to be defined and quantified to ensure they will produce a better project management.

In this chapter you will learn about metrics that can be used to measure efficiency of the processes that are followed in software projects. This chapter details the metrics of **design**, **project**, **effort**, **product**, and **maintenance** so that you can use them effectively in the various phases of the SDLC.

Finally, this chapter will also help you to identify the qualities of ideal metrics and create them for an organization.

3.1.1-What is a Metric

A metric enables you to measure something. For example, to measure the quality of a product deliverable, such as a prototype, the metric could be the number of defects reported for the prototype.

You can measure and quantify a factor only if the factor has numeric values. The numeric value assigned to a qualitative factor can be analyzed and compare to similar values for other factors. The analysis of metrics enables you to make important decisions related to project management. For example, while measuring the quality of a prototype, the data for defects recorded for the prototype enables you to identify the causes of the defects. After categorizing the defects into different types, you devise solutions to deliver a quality prototype in the future.

After the defects are identified, you can assign a severity level to each defect. Severity of defects is classified as follows:

-Severe: Severe defects critically affect the functionality of a product. For example, when a user clicks the **Display** button in a report dialog box, the client computer is disconnected from the server.

-Major: Major defects logically affect the functionality of a product. For example, there is no provision for a user to quit an application by using a button on the main menu screen.

-Minor: A minor defect is a slight defect that may act as an irritant for user, but doesn't disturb the functionality of the application.

Therefore, from the preceding example, the explanation of a metric can be elucidated. A metric quantifies a qualitative factor. In the example, the qualitative factor is the quality of the prototype. As a result, a metric that measures the quality of a product is named a quality or defects metric. The quantification of a qualitative factor defines the need for a metric. The need for a metric is to record, measure, and monitor the changes in a qualitative factor. Referring to the same example, the quality of the prototype is measured in terms of the number of defects that are captured and analyzed for it. After you analyze the changes in a metric, you can decide whether that particular factor requires improvement or not. Improvement in that factor would also impact the overall management of the project. After referring to the example, you can decide that the quality metric certainly requires improvement. Improvement can be by reducing the number of all types of defects. In addition, the organization that created the prototype needs to strengthen its defects prevention mechanism and monitor it regularly.

3.1.2-Characteristics of a Metric

A metric must have certain characteristics so that it is effective. These include:

-Goal-oriented approach

-Measurable
-Analyzable
-Programming language-independent
-Timely

→Goal-oriented approach

A metric should be goal-oriented. For example, the goal of a metric can be to reduce the expenses incurred in different phases of a software project or to reduce the time spent on it. The goal of the metric is used to define a baseline value. A baseline value is a specification that is formally reviewed and agreed upon. It serves as the basis for further development. Baseline can be changed only through formal change control procedures. During the development cycle of a project, the actual data of a particular metric is compared with the baseline data to compute deviations. The magnitude of deviation enables the management to track and control costs, effort in terms of the time spent, productivity, or improve the quality of the product.

→Measurable

Apart from being goal-oriented, a metric should be measurable. Measurability denotes that a metric can be used to measure a software entity to a high degree of accuracy, if not completely accurately. Measurability of a metric ensures consistent results for all processes in a project.
For example, it might be a good idea to measure the productivity of employees in a project, but the same may not be applicable when measuring the reusability of a piece of code. Measurement of code reusability is not a good metric because each programmer has an individual style to approach and solve a problem. Therefore, the metric to measure code reusability is not advisable because it does not yield objective results consistently.

→Analyzable

Another important characteristic of a metric is that it should be suitable for analysis. If a metric is not suitable for analysis, it is futile to monitor it for improvement in a project. For example, an organization uses a metric to measure client satisfaction. Over the years, the organization has experienced changes in client preferences and technological advancement in hardware and software. As a result of these changes, it is necessary that the organization test and analyze the metric again. The metric can be modified based on the result of the analysis.

→Programming-language independent

A metric should also be independent of the programming language used for software development. Metrics that change with a change in the programming language cannot provide reliable results. For example, a software project is divided into three main modules. The interface module is coded in Visual Basic (VB) 6.0. The event functions and procedures are written in PowerBuilder, and the programming for the database is done in Oracle. In such a case where many languages are being used, the metrics that can measure the size, complexity, or the effort spent in coding in all three languages differ. Due to the discrepancy, the organization will be unable to obtain accurate results to track a project or plan it further. In addition, the use of such metrics may cause confusion to project managers and developing teams.

→Timely

Finally, a good metric is timely. This means that the data to produce results using the metric should always be available when it is needed.

3.1.3-Steps to Create a Metric

A process group engineer, a system analyst, or a project manager is primarily responsible for designing metrics. Broadly, there are four steps to create a metric:

-Step1: Define the goal of the metric
-Step2: Identify the requirements of the metric
-Step3: Identify the organizational baseline value for the metric
-Step4: Review the metric for its usability

The first step in creating a metric is to define its goal. The definition of a goal is important because it allows the metric to be designed based on the goal. The goal should be as clear, measurable, and as explicit as possible. For example, a metric can have a goal to measure the number of defects reported by clients. The Figure 3.1 depicts the first step to create a metric.

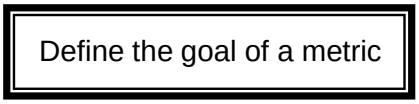

Figure 3.1: First Step to Create a Metric

After defining the goal of the metric, the next step is to identify its requirements. The requirements include human resource, data collection techniques, and methodologies used to process the data. For example, the requirements of a metric that measures the number of defects reported by clients include the availability of quality assurance professionals and past data to specify severity criteria.

Figure 3.2 represents the first two steps in creating a metric.

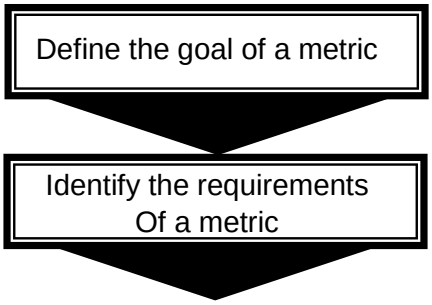

Figure 3.2: First and Second Steps to Create a Metric

The third step in metric creation is the identification of an organizational baseline value for the proposed metric. A baseline value is an average value that an organization may have identified based on prior experience. A metric is designed so as to achieve the baseline value. For example, an organization decides not to have more than 20 severe defects in the acceptance-testing phase. However, a client detects 42 defects during the acceptance-testing phase. In this example, the value 20 is called the baseline value. To measure the worth of a software project, the baseline value is compared with the actual value. If the actual value is greater than the baseline value, the management needs to discuss ways to reduce the number of defects for similar projects in the future. In contrast, if the number of defects is less than 20, the management can explore the reasons for defects fewer than the baseline value. The first three steps are depicted in Figure 3.3.

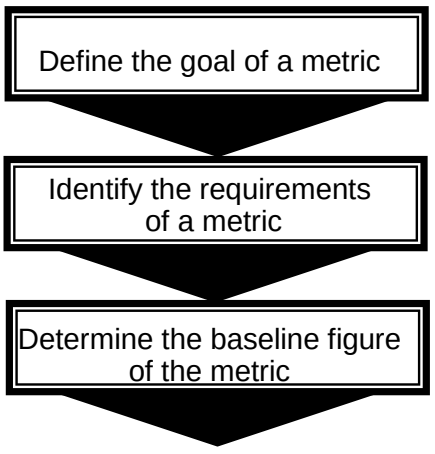

Figure 3.3: First, Second, and Third Steps to Create a Metric

Finally, review the metric for its usability. Process experts can be asked to test and provide feedback on the metric. The feedback can be used to enhance the functionality and the user-friendliness of the metric.

The final step to create a metric is depicted along with the previous steps in Figure 3.4.

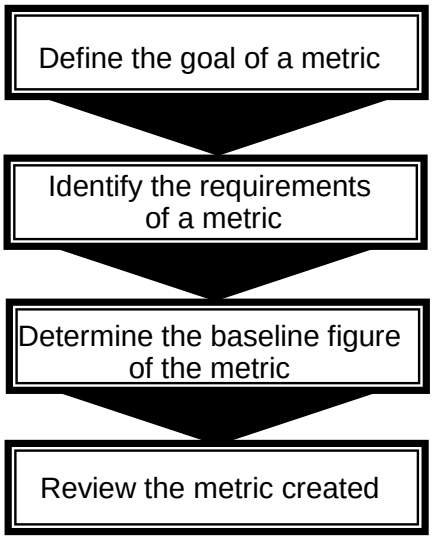

Figure 3.4: Complete Steps to Create a Metric

3.2-Types of Software Metrics

Four types of metrics are usually used in an organization. All these metrics operate with a single objective to improve the quality of processes and products in an organization. The types of metrics commonly used by organizations include:

- ***Design metrics.*** These are used to record design issues, which correspond to the requirements document. Design metrics enable you to decide how much you have deviated from the requirements of the project. Lesser the deviation, fewer the number of defects. Design issues need to be addressed as soon as they are recorded so that the same are not repeated in the later phases of the SDLC.
- ***Project metrics***. Project metrics is a set of Metrics related to all SDLC phases. Project Metrics measure the effectiveness of the processes followed in each phase.
- ***Product metrics***. These metrics are used to measure the quality of deliverables produced in a software project life cycle.
- ***Maintenance metrics***. These measure the progress of a maintenance project.

3.2-1-Design Metrics

During the design phase it is possible to measure the characteristics of a software program. Complexity is a characteristic measured during this phase. Complexity can be measured for the structure, data components, and the interface design of a software program. Complexity can affect the size, testability, and effort spent on developing and testing the modules of a project. The measure of the complexity of a software design is **called a design metric**. Design metrics are seldom used in organizations because organizations find it time-consuming to analyze their results. Moreover, design metrics are difficult to implement.

There can be many varieties of design metrics. One variety is ***architecture design metrics***.

→Architecture Design Metrics

An architecture design metric measures the complexities of a software program by referring to the design of the program. All software projects are developed from a blue print. The blue print or the architecture design of the software is crucial for predicting the features and functionality of the final product.

An architecture design metric addresses three types of complexities of a software program. As a result, there are three types of architecture design metrics. These are listed below:

- Structural complexity
- Data complexity
- System complexity

The number of fan-out modules in a software program defines the structural complexity metric. A fan-out module refers to a module invoked by a module parent to it. For example, a module, Chk-name, invokes a module to check for a user name that exists on a database. Chk-name also invokes another module that checks for a string of not more than 12 characters. In addition, a module checks for invalid numeric and alphanumeric strings. Therefore, the module Chk-name invokes three modules as represented in Figure 3.5. This indicates that the fan-out value is three. Multiple fan-out modules increase the complexity and the speed of execution of a software program. This can affect the testability requirements of the modules as well.

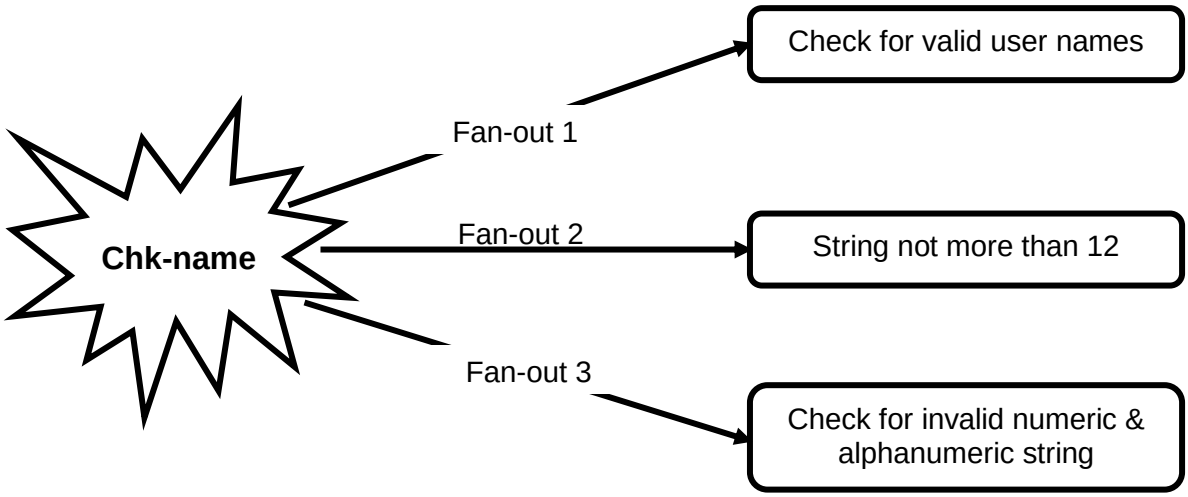

Figure 3.5: Fan-out of a Module

The data complexity metric measures the complexity of the internal interface of a software program. An internal interface is defined by its input and output variables. For example, a module, chk-dt validates a date entered by a user in the mm/dd/yy format. For this, an input variable accepts a date value. An incorrect date value is replaced by a corrected date value in an output variable. Existence of such variables increases the complexity of a software program.

You obtain the system complexity metric by combining the values of structural complexity and data complexity. As you have seen, system complexity provides a complete view of the complexity in a software program. After determining the complexity of a program by manually counting the fan-out and data variables in the module of a software program, you can plan the effort required to develop and test the software. For example, there are three main modules in a project. Module 1 invokes modules 2 and 3. Module 3 contains maximum number of variables. In such a situation, you can plan for more resources and time to test and develop module 1 first. Subsequently, module 3 can be developed and tested. Module 2 that neither invokes any other module nor is relatively complex can be planned for later development and testing.

3.2-2-Project Metrics

Project metrics are specific to the actual execution of a project. They enable you to execute a project with a systematic and technical approach. Project metrics can help you avoid project delays. You can also use them to avoid and mitigate project risks. Project metrics are also used to assess the quality of a finished product on a periodic basis. Examples of project metrics are the metrics that measure the size, complexity, or the time and effort spent on a project.

Six important project metrics are commonly used in an organization. The following metrics are applied in the different phases of the SDLC.

- Effort
- Productivity in FP
- Cost
- Size
- Defects
- Testing

→1-Effort Metric

The effort metric enables you to determine the amount of effort required to complete a project. Effort is generally estimated close to accuracy in all the phases of the SDLC. To define an effort metric, you create a metric like the one shown in Table 3.1. The table depicts effort in man-months. A man-month is the amount of effort required to complete work in a month.

Table 3.1: All Effort Metric

	Metric(s)	Analysis	Design	Construction	Testing	Total
Effort	Planned effort (man-months)	5	12	25	10	52
	Actual effort (man-months)	8	14	35	8	65
	Percentage of increase in effort (%)	60	17	40	-20	25

The planned value for effort is known as the baseline value because it serves as the nodal point for comparing the planned and the achieved values. You can calculate the actual effort in man-months by using the actual effort data from the daily effort-tracking database. For example, in Table 3.1, the planned effort to complete the analysis phase is 5 man-months. However, the actual effort spent on analysis is 8 man-months. On the other hand, the planned effort to complete the testing phase is 10 man-months whereas actually it takes 8 man-months to complete testing. The difference between planned and actual effort is expressed in percentage. This difference is termed as the increase in effort. For example, the difference between planned and actual effort during the analysis phase is 60 percent. This increase in effort is a cause of concern for the management. The management should strive to gradually reduce it. On the contrary, the testing phase displays negative percentage increase in effort. The management needs to investigate what was done differently in the testing phase so that the effort was curtailed.

Many international organizations also calculate the effort for onsite and offshore development activities to calculate the total effort on development.

→2-Productivity Metric

Another project metric is the productivity metric. In a software project, you measure human effort according to the number of hours spent on an FP of work. The effort in performing an FP of work in an hour is called ***productivity***. To calculate productivity, you first determine the total amount of FP for the software project. Then, distribute the total FP among the phases in the SDLC.

To calculate accurate productivity for a particular phase, you need two types of data:

- Actual effort expressed in person hours
- Actual size of the project in FP

Just like you did for the effort metric, you can retrieve the actual effort data from the daily effort-tracking database to calculate productivity. On the other hand, the actual size of a project in FP is retrieved from the project plan. The project plan contains the estimated size of the project, which is finalized during the analysis phase.

To understand how productivity metric is calculated, consider Table 3.2. The table displays the planned effort versus actual effort for a software project. The effort data is expressed in terms of hours per FP. Assume the actual software size is 100 FP. To complete 100 FP of work in the project, the estimated

productivity required is 8 hours per FP. Note, that this value is a total of the productivity required for all the phases. However, the actual productivity reflected in the table is 7.9 hours per FP. This indicates that the productivity is lower than planned. The productivity metric in Table 3.2 makes it clear that productivity was lower in the initial phases of development as compared to the later ones. In this situation, the management needs to figure out the cause of low productivity in the initial phases. They may even want to find out what was done differently in the later phases to improve productivity.

Table 3.2: A Productivity Metric

Metric(s)	***Productivity* in hours/FP Construction**				
	Analysis	**Design**	**Construction**	**Testing**	**Total**
Planned	1.2	1.6	2.8	2.4	8.0
Actual	1.0	1.4	3.0	2.5	7.9
Difference (+/-)	0.2	0.2	-0.2	-0.1	0.1

→3-Cost Metric

Cost is yet another project metric that enables you to manage a project efficiently. A cost metric measures the planned versus actual expense incurred on a project. An important component or the cost metric is the cost of resources. This includes the human resource and material resources required for a project. Incidentally, cost is the most difficult metric to control. This metric usually runs out of control during a project. Cost overrun can be attributed to a rise in the price of the required resources, change, or shift in project objectives, or increase in the requirement or resources.

Table 3.3 displays a sample cost metric. In the table, the cost metric includes the cost of travel and communication for project-related purposes. You can calculate the expenses incurred during every phase and record them as shown in Table 3.3. This enables you to control and reduce costs in different phases.

Table 3.3: A Cost Metric

Metric		***Cost in US dollars (s)***				
		Analysis	***Design***	***Construction***	***Testing***	***Total***
Resources	***Planned***	40,000	100,000	200,000	100,000	440,000
	Actual	55,000	130,000	300,000	250,000	735,000
Communication	***Planned***	10,000	7000	1000	1000	19,000
	Actual	50,000	4,500	700	500	55,000
Total planned		50,000	107,000	201,000	101,000	459,000
Total actual		105,000	134,500	300,700	250,500	790,0OO
Total Deviation in percentage: (actual- planned)/ planned*100		110	26	50	148	72

You can also use the cost metric to measure the deviation in the total planned expenditure when compared with the actual expenses incurred during a project. For example, the data in the table reveals that there is an increase in communication costs over the plan in the analysis phase. During the subsequent phases, the cost deviations related to communication with the client reduce.

The deviation in costs enables you to calculate the amount of additional investment that needs to be provided for the project. Determining the estimated cost at project initiation helps map the budget with the resource allocation required for a project.

→4-Size Metric

Another important project metric is the size metric, which is calculated for an entire project and not for any particular phase of a project. The size of a project may vary with respect to the changes required by the client. You may record data related to the size of a project in a table format as displayed in Table 3.4.

The size metric directly affects the effort, testing, and productivity metrics of a project.

Table 3.4: A Size Metric

Metric		***Total***
Size in FP	Size Planned	100
	Size Actual	120
	Change in size	20
	Growth of size in percentage	16%

There are two main components of the size metric:

- Change in the required effort due to change in project size
- Percentage growth in project size

A change in the effort simply measures the effort in terms of FP for every change requested by the client and accepted by the development team. The same data can also be presented as percentage. The growth of size in percentage metric reflects the change in the size of a project compared to its original or planned size. This is an important metric because frequent changes in the size of a project may involve additional effort and resources, which have to be deployed by the management. Frequent changes in the size of a project also mean that the design specifications are not clearly defined at the start of the project. By using the size metric, you can avoid the design pitfalls observed in one project in future projects.

→5-Defects Metric

Defects are errors that occur in a work product. The number of defects defines the quality of a project. If the number of defects is large, the quality of the product is inferior. In contrast, if the number of defects is small, the quality of the product is superior.

Defects can occur during any phase of the SDLC. There can be defects related to the analysis of project requirements, design, or coding of software. Defects are not detected in a particular phase continue to be present during the subsequent phases in the SDLC. To facilitate calculation and categorization of defects, many organizations use automated tools and techniques. These tools and techniques also help you in defining the severity of defects.

You can use the displayed Table 3.5 to record and track the number of defects reported per KLOC in every phase.

Table 3.5: A Defects Metric

Metric		***Defects per KLOC***				
		Analysis	***Design***	***Construction***	***Testin***	***Total***
Severe	Planned (less than equal to)	0	0	0	0	0
	Actual	5	3	15	8	31
Major	Planned (less than equal to)	5	2	5	10	22
	Actual	3	6	12	18	39
Minor	Planned (less than equal to)	20	15	15	20	70
	Actual	12	10	12	25	59

Table 3.5 depicts the defects in various categories in every phase of the SDLC. It is evident from the table that the management had expected no defects in the severe category in all phases. However, there is a considerable increase in severe defects in all phases.

Similarly, less than equal to 10 major defects per KLOC were estimated during the testing phase but 18 major defects were reported in this phase. In the same phase, 25 minor defects per KLOC were also reported as compared to the expected value of 20 minor defects. The management needs to study the causes for an increase in the number of defects in all phases.

→6-Testing Metric

The testing metric is used to measure the number of test cases required to test software. Test case is a specification that needs to be executed to test a particular module in a software program. There are separate test cases for integration testing and unit testing. *Integration testing* refers to the overall black box testing of the entire software project. In contrast, *unit testing* refers to testing individual modules of a software project.

You can use the format displayed in Table 3.6 to record the number of test cases required for every phase.

Table 3.6: A Testing Metric

Metric(s)		***Construction (Test cases per FP)***
Testing	Planned	*5*
	Actual	4

The size of a phase determines the number of test cases. Test cases enable rigorous testing of every module of the entire software. From Table 3.6, it can be observed that during the construction phase, the number of test cases per FP were less than planned. This is a cause of concern for the management. The management and the QA department should find out the reasons for such discrepancy. If the number of test cases is less than planned, it implies that software is not tested rigorously. It may also imply that software is not tested according to the software specifications developed in the analysis phase. If a software product is expected to be complex and extensive, it is advisable to develop more test cases per FP.

3.2.3-Product Metrics

Analysis done using the product metrics can provide you an insight into the quality of work done by the development teams. Product metrics trace and measure the quality of a deliverable in different phases. For example, a source code is the output deliverable of the construction phase. A set of metrics measures the quality and timeliness of the deliverable. To measure the metric of source code, you can use the format displayed in Table 3.7.

Table 3.7: Format of a Product Metric

Source code	***Deliverable name***	
	Effectiveness (%)	***Conformance to requirements (%)***
Planned	100	100
Actual	72	90

In Table 3.7, two factors that are measured:

- Effectiveness
- Conformance to requirements

Effectiveness of a deliverable is measured in terms of the quantity of the deliverable completed and the number of quality goals achieved. You can use the following formula to calculate ***effectiveness*** of a deliverable:

Effectiveness = (Quantity in %) * (Quality in %) / 100

For example, **80 percent** of source code is completed and **90 percent** of the quality goals are achieved in developing that source code. Therefore, after substituting the values in the effectiveness formula, you can calculate effectiveness as follows:

Effectiveness = 80 * 90/100

➔***Effectiveness = 72***

It is also clear from the table that the source code conformed to most of the requirements mentioned in the requirements document. To calculate the conformance to requirements, you count the number of requirement in the requirement document sent by the client. Next, you count the number of requirements that are actually completed. Finally, the difference between the two values is calculated and expressed in terms of percentage. The value indicates the extent of deviation between the planned and actual conformance to requirements.

3.2.4-Maintenance Metrics

Maintenance metrics are used for maintenance projects. Maintenance projects require enhancements based on client feedback or changes in the market, technology, and user preferences. The client fills up a change request form in which the changes and the types of changes required are mentioned. Table 3.8 displays a sample change request form.

Table 3.8: A Sample Change Request Form

Project Name: Client Name: Date:	
Functional description of the change desired	
Type of Change (Tick the appropriate one(s))	**Change Trigger Component (Tick the appropriate one(s))**
-Correction -Enhancement -Adaptation -Re-engineering	-Requirement -Design -Code -Other
Total effort required to perform change:	
Responsibility of performing change:	

There are two types of measurements performed for maintenance projects. These are:

- Extent of change required
- Type of maintenance requested by the client

The first type of measurement is to gauge the extent of change required. To do this, you count the total number of modules in the software application launched. You also keep track of all the modifications done in any module of the application. In addition, the modules added and deleted from the previous version of the application are also tracked. You can use the format displayed in Table 3.9 to track these details.

Table 3.9: A Module Track Metric

No. of modules remaining unmodified	6
No. of modules added to a released software program	3
No. of modules modified in a released software program	10
No. of modules deleted from a released software program	2
Total	21

The second type of measurement performed for maintenance projects is in terms of the type of maintenance requested by the client. There are two metrics used to measure maintenance activities in an organization.

- Corrective
- Upgrades

Corrective maintenance addresses the software-related defects reported by the client. The defects can be either related to the functionality of software or violation of requirements specifications. An organization should strive to keep this maintenance under control because it clearly indicates basic design defects in the design structure of software.

The other type of metric used for maintenance in an organization is upgrades. These include enhancements on software to incorporate additional functionality. In addition, upgrades include adapting software to the changing technology and making preventive changes to the software to eliminate anticipated problems.

Table 3.10: A Maintenance Metric

Type of Maintenance	Planned in %	Actual in %
Corrective	17	70
Upgrades	83	30

Table 3.10 displays the corrective and upgrades maintenance metrics in an organization. Note that the percentage of corrective defects that the organization initially planned is much lower than what it actually addressed. In contrast, upgrades that were actually addressed formed a much lower percentage of what was actually planned. This indicates that the organization is spending effort in addressing basic design defects as compared to enhancements. This may also imply that unit and functional testing of software was ineffective.

You can use the data collected for maintenance metric to calculate the software maturity index (SMI). This indicates the stability and reliability of a software product. If the value of SMI is less than 1, the software product is not very stable and needs transformations for improvement. When the SMI value reaches 1, the software product is said to be stable. A stable software product is unlikely to change frequently in response to user and market preferences. However, the software can always be open to technological breakthroughs and enhancements.

Summary

Metrics assign values to quantitative factors. Metrics facilitate project planning, scheduling, and improving the SDLC process and product quality. The four main types of metrics used widely during the SDLC of a project are design, project, product, and maintenance.

Design metrics help measure the design and architecture of a software project. Project metrics comprise effort, productivity, cost, defects, and size. Project metrics measure the effectiveness of the important factors for a project. They are implemented in every phase of the SDLC.

Product metrics measure the quality and conformance to the requirements of the deliverables of a phase. They also measure the timeliness in the delivery of project deliverables. Maintenance metrics are a set of metrics similar to the metrics for a development project. They are used to measure the cost, effort, productivity, and defects associated with a maintenance project. You use maintenance metrics to track the number of required changes implemented in a maintenance project.