Software Metrics

A software metric is a quantitative measure of an attribute of a software development project. Two types of software metrics can be captured: process metrics and product metrics. Process metrics measure attributes of the development process and the environment, for example, the experience of the programmers. Product metrics measure attributes of the intermediate or final software product, for example, the number of requirements in the system.

Process Metrics	Product Metrics
Measures Quantifiable Attributes of the Development Process and Environment	Measures Quantifiable Attributes of the Software Products (Intermediate or Final)
• Experience of programmers	• Size (function points, LOC, displays, etc.)
• Cost of development and maintenance	• Number of classes, objects, etc.
• Development time (by phase and activity)	• Number of requirements (specified and derived)
• Development techniques	• Logic structure complexity (flow of control, depth of nesting)
• Process model	• Data structure complexity (number of variables)
• CASE tools	

Dr. Scott Knight and Major Greg Phillips from the Royal Military College of Canada, Department of Electrical and Computer Engineering stated the six principles of good engineering as:

1. A successful software development project meets its cost, schedule, and quality goals.
2. Development plans should set quantitative goals so you can tell if you are meeting them.
3. Plans should be compared with actual performance throughout development to detect potential problems early.
4. Data trends collected over time are often better indicators of potential problems than actual values as they show when deviations from the plans are temporary, fluctuating, growing, or diminishing.
5. There are many explanations, both good and bad, for the same set of data. Metrics do not indicate problems, but where data values should be investigated.
6. Be aware that how metrics are presented can obscure or clarify the message.

These principles strongly suggest creating and incorporating a solid metrics program.

A corporate metrics program is an approach to software metrics collection and assessment institutionalized across the company or corporation. Data from projects is used to establish, validate, and refine the corporate metrics database which provides an empirical foundation for future project estimations, analysis, and prediction.

The Importance of Collecting Metrics

Measurement is the collection of metrics. It is important to measure as:

Measurement provides visibility into your project's status. The only thing worse than being late is not knowing that you are late. Measuring your progress can help you know exactly what your project's status is.

Measurement focuses people's activities. People respond to objectives; when you measure a characteristic of the development process and feed it back to those involved, they naturally work to improve their performance against that characteristic.

Measurement improves morale. Properly implemented, a measurement program can improve morale by bringing attention to chronic problems and their resolution.

Measurement is an essential element of software process improvement. Metric data is used to assess the organization's process improvement initiatives at the department, division, or corporate level. Using data to assess process effectiveness and provide the basis for process improvement is an essential element of total quality management (TQM) and is referred to as statistical process control. One of the requirements for achieving Capability Maturity Model Integration® (CMMI®) Level 2, measurement and analysis activities must be evident and management control is based on collecting and assessing metrics.

How Many Metrics to Collect

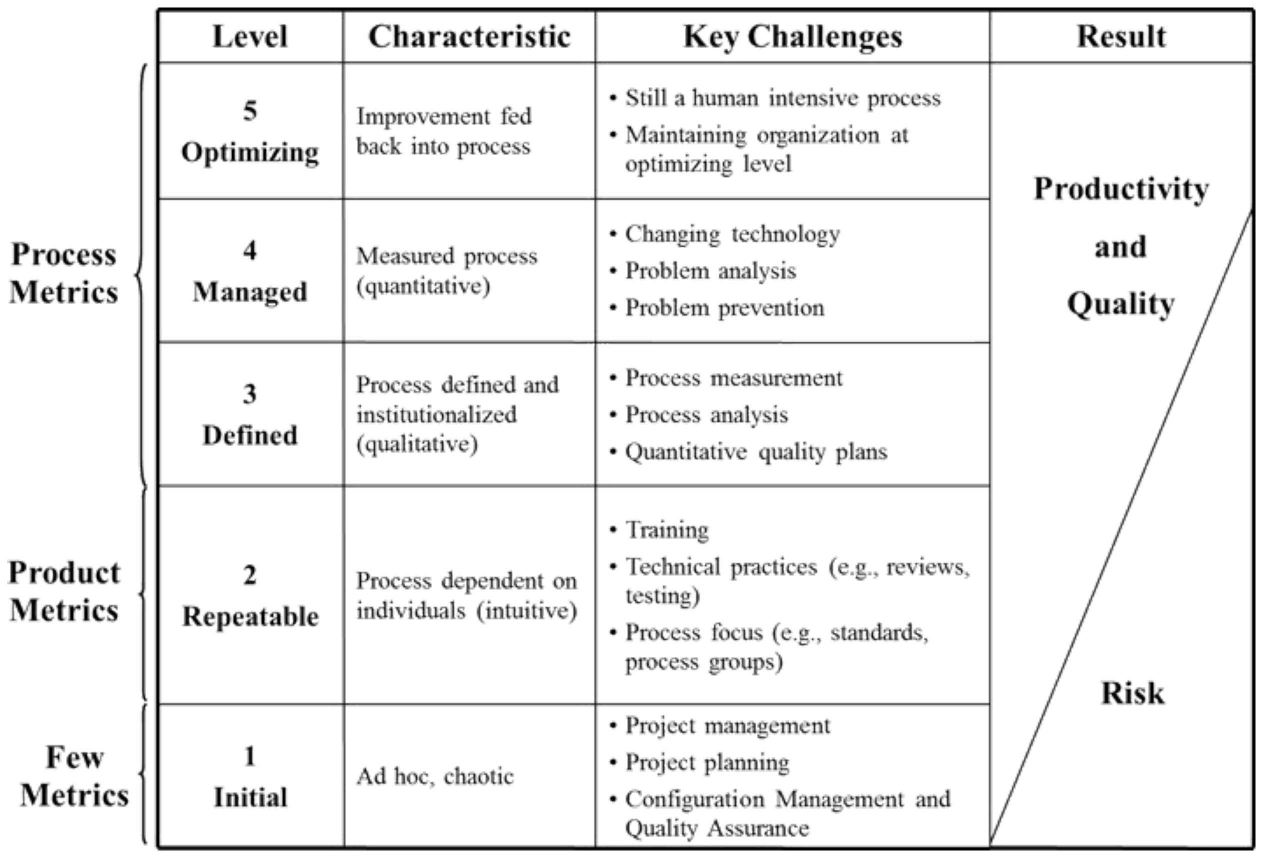
While collecting metrics is important, some organizations waste time and money collecting more data than they need. Be sure you are collecting data for a reason. The University of Maryland's Professor Victor Basili and David Weiss PhD in conjunction with NASA Goddard Space Flight Center developed an approach to software metrics that follows a Goal, Question, Metrics (GQM) process:

1. Define the goals at a conceptual level. The goals can be at the project, department, division, or corporate level that can be measured for productivity and quality.
2. Ask questions at the operational level. The questions should define the goals quantifiably.
3. Establish a set of quantitative metrics. Determine the measures that need to be collected and tracked.
4. Determine how to collect the data.
5. Gather, confirm, and evaluate the data and **provide feedback to the source**. This step is most important. If feedback is not rendered, the source will continue to perform the activities as they have in the past and suggested improvements will be lost.

Independently analyze the data to judge conformance to the goals and to make recommendations to improve the process.

Set goals: Determine how you want to improve your process, for example, reduce the number of defects	**Ask questions**: Generate questions that are aimed at the goals, for example, • What kinds of defects do we have? • Which are most common? • How are they discovered? • Which cost the most to fix?	**Establish metrics**: Set up a measurement to collect a set of metrics that will answer your questions, for example, collect: • Defect types • Quantity of each defect • Creation time • Detection time • Cost to repair

The Software Engineering Institute (SEI) introduced the Capability Maturity Model, which is now replaced by the Capability Maturity Model Integration® (CMMI®), which requires metrics at every process improvement level. The bottom line is as productivity and quality increase, risk decreases.



	Level	Characteristic	Key Challenges	Result
Process Metrics	5 Optimizing	Improvement fed back into process	• Still a human intensive process • Maintaining organization at optimizing level	Productivity and Quality
	4 Managed	Measured process (quantitative)	• Changing technology • Problem analysis • Problem prevention	
	3 Defined	Process defined and institutionalized (qualitative)	• Process measurement • Process analysis • Quantitative quality plans	
Product Metrics	2 Repeatable	Process dependent on individuals (intuitive)	• Training • Technical practices (e.g., reviews, testing) • Process focus (e.g., standards, process groups)	Risk
Few Metrics	1 Initial	Ad hoc, chaotic	• Project management • Project planning • Configuration Management and Quality Assurance	

Who Collects Metrics

Effective m easurement requires a specialized set of skills and knowledge of the following:

- Statistics and multivariate analysis
- Software engineering processes
- Software project management
- Software planning and estimating tools
- Design of data collection forms
- Survey design
- Quality control methods including reviews
- Inspections and standard form of testing
- Accounting principles

What Metrics to Collect

In 1988, The MITRE Corporation conducted a study and developed a set of ten Software Management Metrics based on three years of Government and Industry experience for the Electronics Systems Division of the Air Force Systems Command (AFSC). The study team collected and reported data at monthly intervals, plotted the planned and actual data from the previous 12 months and the planned data for the next 5 months. It established that revised plans may be added to the graphs, but previous plans may not be removed. This study was based on review milestones from the MIL STD 498 process. An overview of these metrics that are still in use today, their descriptions, and how to interpret the metrics are:

MITRE Metric	Description	Interpret the Metric
1. Software Size	The planned size and current estimated size or actual size, measured in lines of code. Used to measure total effort and schedule and productivity.	• Initially an estimate which becomes more accurate until it represents actual data • Increases may indicate additional requirements, better understanding of requirements, optimistic original estimate • Decreases usually result from overestimate at program startup
2. Software Personnel	The number of staff members planned and number of actual staff members currently doing development and documentation. Used to measure productivity and predict schedule delays or cost overruns due to understaffing or overstaffing.	• Total software staff profile should grow through the design phases, peak through the coding and testing phases, and gradually taper off as integration tests are successfully completed • Experienced staff profile should be high during the initial stages, dip slightly during code and unit test, grow again for integration and test • Ratio of total to experienced staff should typically be near 3:1, and should never exceed 6:1 • Understaffing results in schedule slips • Adding inexperienced staff late in the project will seldom improve schedule • Adding experienced staff may improve schedule but will affect cost • High personnel loss may indicate morale problems, will cause cost and schedule slips
3. Software	The number of specified	• Requirements changes are

MITRE Metric	Description	Interpret the Metric
Volatility	functional requirements and unresolved requirements issues. Used as an indication of unstable requirements that may cause rework.	expected early in the program, as details of the system are being defined and understood. At some point the software requirement must be frozen. The later this happens, the greater the impact on cost and schedule. • Requirements changes after Critical Design Review (CDR) have significant schedule impact, even if the change is a requirement deletion
4. Computer Resource Utilization	The estimated and actual percentage of the target systems hardware CPU, storage, and communications capacity being used.	• Most projects experience an upward creep in resource utilization • Little change occurs prior to CDR when estimates are based on modeling and simulation • Target computer performance deteriorates quickly when CPU and I/O channel utilization exceeds 70% for real-time applications • Software development cost and schedule increase rapidly when last minute code and design changes are employed to avoid exceeding resource utilization limits
5. Design Complexity	Uses McCabe's method to measure the designs complexity. Used to determine parts of the design that may be error prone and hard to maintain.	• Modules with high cyclomatic complexity are considered more prone to defects, more difficult to test, and more difficult to maintain • Upper limits between 10 and 14 are proposed in the literature • Exceptions are made when a high rating results from the use of case and exception statements • Complexity is reduced by dividing an overly complex module into multiple modules; this may increase the complexity of the unit or component which contains the module
6. Schedule	Ratio of total schedule to the	• It is not unusual for a project

MITRE Metric	Description	Interpret the Metric
Progress	ratio of the budget spent.	to fall behind early due to difficulty in staffing up, requirements volatility, or insufficient time allocated to design • An upward trend in the estimated schedule to complete should level off as the design is implemented • If the upward trend continues, this may be due to an increase in project size or complexity • An upward trend during system testing may indicate poor code quality or inadequate testing at the unit and component levels
7. Design Progress	The number of requirements that have been scheduled to be documented and actually number documented.	• Increase in software requirements from system design documents allocated to software requirements and design documents indicates poorly defined system requirements or inadequate system design
8. Computer Software Unit (CSU) Development Progress	The planned versus actual units designed, tested, and integrated. Used to measure work planned and done in terms of products that complete some phase of development.	• Plots of units designed and tested should rise with a fairly constant slope • Constant continuous slippage in unit development can be caused by growth in the system size or understaffing • Sporadic unit development can be caused by unexpected personnel losses, by under-experienced personnel, or late requirements changes
9. Testing Progress	The planned and actual configuration items and system tests completed, number of new problem reports, and open or unresolved problem reports. Used to measure progress in completing testing, number of potential defects found, and efficiency in resolving them. Can be used to estimate quality and time needed to complete tests.	• Most projects experience failed tests and schedule slips • Increase in Software problem (SP) and Change Reports (CR) may indicate poor code quality or inadequate testing at the unit and component levels • Problems in unit integration testing may also indicate that problems will be encountered during system testing
10. Incremental Release Content	The estimated and actual release date and the estimated and actual components in each	• Plots of units integrated into each build thread should rise

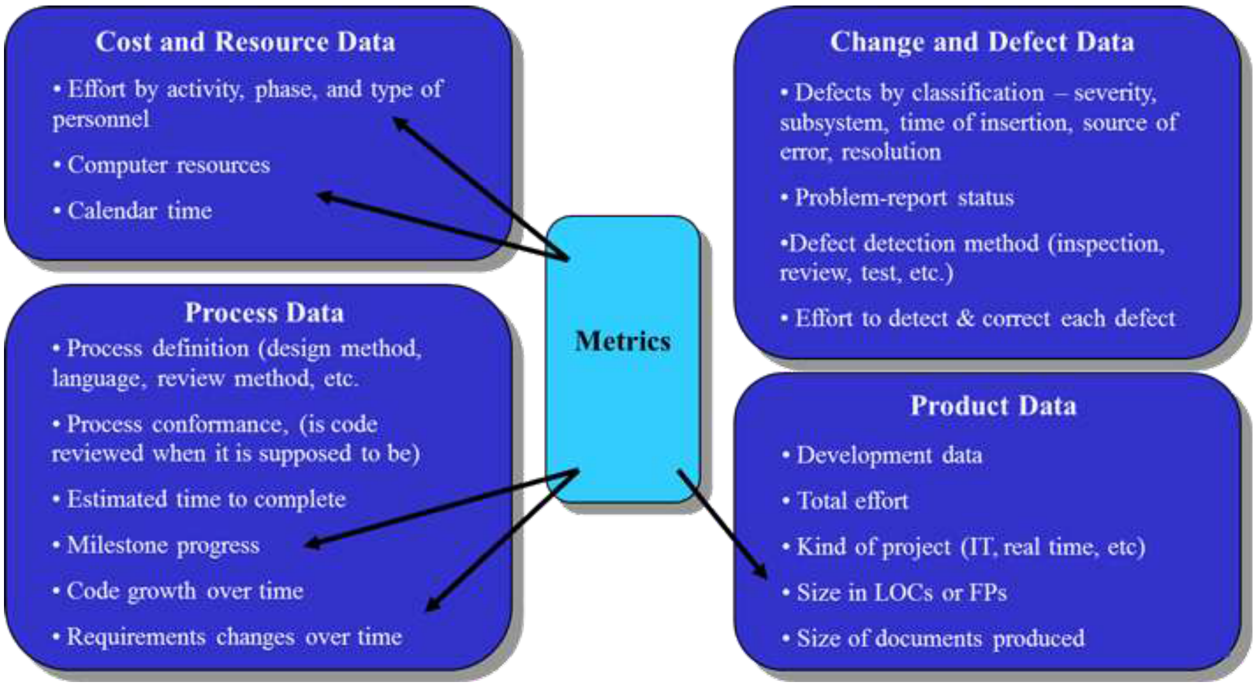
MITRE Metric	Description	Interpret the Metric
	release. Used to measure progress and requirement changes used to meet schedule.	with a fairly constant slope • Constant continuous slippage in unite integration can be caused by unexpected personnel losses, under-experienced personnel, or late requirements changes

The most frequently used metrics are:

- Size
- Personnel
- Computer Use
- Unit Progress
- Testing Progress

The bottom line is your metrics program should collect four types of data:

1. Cost and resource data
2. Process data
3. Change and defect data, and
4. Product data


Cost and Resource Data
• Effort by activity, phase, and type of personnel
• Computer resources
• Calendar time
Process Data
• Process definition (design method, language, review method, etc.
• Process conformance, (is code reviewed when it is supposed to be)
• Estimated time to complete
• Milestone progress
• Code growth over time
• Requirements changes over time
Metrics
Change and Defect Data
• Defects by classification – severity, subsystem, time of insertion, source of error, resolution
• Problem-report status
•Defect detection method (inspection, review, test, etc.)
• Effort to detect & correct each defect
Product Data
• Development data
• Total effort
• Kind of project (IT, real time, etc)
• Size in LOCs or FPs
• Size of documents produced