--- Defect Removal Efficiency

**Definition** : The defect removal efficiency (DRE) gives a measure of the development team ability to remove defects prior to release. It is calculated as a ratio of defects resolved to total number of defects found. It is typically measured prior and at the moment of release.

**Calculation**

To be able to calculate that metric, it is important that in your defect tracking system you track:

* **affected version**, version of software in which this defect was found.
* **release date**, date when version was released

DRE = Number of defects resolved by the development team / total number of defects at the moment of measurement.

DRE is typically measured at the moment of version release, the best visualization is just to show current value of DRE as a number.

--- Test Effectiveness Ratio

The **Test Effectiveness Ratio** in software metrics is a measure used to evaluate the efficiency of a testing process. It is calculated by comparing the number of defects found by testing to the total number of defects in the system (including those found after release). The formula is:

Test Effectiveness Ratio= Defects found during / Total defects (defects found during testing + after release) \* 100

A higher ratio indicates that a larger proportion of defects were identified during testing, implying more effective testing.

--- McCabe’s Cyclomatic Complexity

Introduced by Thomas McCabe in 1976, cyclomatic complexity measures the number of linearly-independent paths through a program module. This measure provides a single ordinal number that can be compared to the complexity of other programs. Cyclomatic complexity is often referred to simply as program complexity or as McCabe's complexity.

Methods with a high complexity tend to be more difficult to understand and maintain. In general the more complex the methods of an application are, the more difficult it is to test the application, which adversely affects the application's reliability. V(G) counts the number of branches in the body of the method defined as:

* if statements
* conditions such as && and ||
* for statements
* while statements

This metric is computed as follows:

* Each function has a base complexity of 1
* Each atomic condition adds 1
* Each case block of switch adds 1

If V(G) is larger than 10, consider to split the method up. The more complex the program the harder it is to test and comprehend. This metric can additionally be interpreted as the cost of producing test cases for the code.

--- Validation

In software metrics, **validation** refers to the process of ensuring that the metrics being collected are accurate, reliable, and reflective of the software's actual performance or quality. It involves verifying that the data being measured aligns with the intended goals and that the results can be trusted for decision-making. Validation ensures that the metrics are meaningful and not misleading, confirming that the measurement approach and the tools used are effective and appropriate for the project's needs.

--- Mean Time To Change

1. **Important to**: Everyone, especially the CEO, CIO and CTO.
2. **Definition**:   
   How long does it take a new average feature, idea, fix or any other kind of change to get into a paying customer's hands, in production, from the moment of inception in someone's mind. MTTC is what it takes you from the moment you see an opportunity until you can actually utilize it. The faster MTTC is, the faster you can react to market changes.
3. **How to measure:**  
    We start counting from moment of the change's inception in someone's head (Imagine a marketing person coming up with a competing idea to that of a competitor's product, or a bug being reported by a customer)
4. One way to capture and measure mean time to change is by doing a value streaming exercise, as we will touch on in a later chapter in this book.
5. **Expected Outcome:**  MTTC should become shorter and shorter as Dev maturity grows.
6. **Common Misunderstandings:**  
   MTTC is not the same as the often cited "**Change lead time**" as proposed in multiple online publications.
7. Change Lead time only counts the time from the start of development of a feature, when real coding begins.
8. MTTC will measure everything that leads up to the coding as well, which might include design reviews, change committees, budgeting meetings, resource scheduling and everything that stands in the way of an idea as it makes its way into the development team, all the way through to production and the customer.
9. From a CEO, CIO and CTO view , MTTC is one of the most important key metrics to capture. Unfortunately, many organizations today do not measure this.

--- SW Failure

"SW Failure" in software metrics refers to a software failure, which occurs when a system or application does not perform its intended function correctly or crashes. In the context of software metrics, it typically relates to the measurement of failure rates, bug occurrences, and the impact of these failures on system performance or user experience. These metrics help developers understand the reliability and quality of software, and guide efforts to improve its robustness and reduce failure occurrences.

--- Purpose of metrics validation

The purpose of metrics validation is to identify both product and process metrics that can predict specified quality factor values, which are quantitative representations of quality requirements. Metrics shall indicate whether quality requirements have been achieved or are likely to be achieved in the future. When it is possible to measure quality factor values at the desired point in the life cycle, direct metrics shall be used to evaluate software quality. At some points in the life cycle, certain quality factor values (e.g., reliability) are not available. They are obtained after delivery or late in the project. In these cases, validated metrics shall be used early in a project to predict quality factor values.

--- Process measures

Process measures quantify behavior, strategies, and execution of the process used to develop the product. One general category of process measures is event counts, such as the number of defects found in test, requirement changes, or milestones met. Another general category concerns time measures, such as cycle time: time to complete a project. In highly competitive markets, cycle time, or deployment, may be more important than reducing development costs

--- Tools for requirements management

Tools for requirements management store and manage information about requirements. They allow testers to prioritize requirements and to follow their implementation status.

In a narrow sense, they are not testing tools, but they are very useful for defining a test based on requirements (see section 3.7.1) and for planning the test; for example, the test could be oriented on the implementation status of a requirement. For this purpose, requirements management tools are usually able to **exchange information** with **test management tools.**

Thus, it is possible to seamlessly interconnect requirements, tests, and test results. For every requirement the corresponding tests can be found and vice versa. The tools also help to find inconsistencies or holes in the requirements, and they can identify requirements without test cases; that is, requirements that otherwise might go untested.

Requirements and specification documents generally combine text, graphs, and special mathematical diagrams and symbols. The nature of the presentation depends on the particular style, method, or notation used. When measuring code or design size, you can identify atomic entities to count (lines, statements, bytes, classes, and methods, for example).

However, a requirements or specification document can consist of a mixture of text and diagrams. For example, a use case analysis may consist of a UML use case diagram along with a set of use case scenarios that may be expressed as either text or as UML activity diagrams. Because a requirements analysis often consists of a mix of document types, it is difficult to generate a single size measure.

--- Design phases

* Architectural design - Identify sub-systems
* Abstract specification - Specify sub-systems
* Interface design - Describe sub-system interfaces
* Component design - Decompose sub-systems into components
* Data structure design - Design data structures to hold problem data
* Algorithm design - Design algorithms for problem functions