Maintainability

- During the maintenance phase, the software system, is corrected, adapted, perfected and enhanced perhaps following the planning, execution and control as well as review and evaluation guidelines recommended for the Maintenance phase in ISO/IEC 12207.
- To avoid maintenance cost overruns, it is imperative that the software is maintainable.
- Software maintainability as a software quality attribute is defined as the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, adapt to a changed environment or accommodate new requirements.
- Maintainability of a software is assessed in terms of Analyzability, Modularity, Reusability, Modifiability and Testability.

Analyzability

- ease of effort required to understand, decompose and/or modify a particular source code
- While consistency is the key, following efforts help improve analyzabiliy:
  - Enforcing established coding conventions for names, comments and format of the software.
  - Utilizing well known coding patterns, design patterns and reference architectures as the foundation of the software.
  - Maintaining mandated documentation and any accompanying change logs, and making sure that these are accessible to all stakeholders.
  - Establishing a set criteria and format for error reporting and logs.
  - Conducting code reviews on a regular basis or integrating static code analysis tools such as lint in the build process to monitor that the above conventions are being followed.
  - Collecting metrics e.g. how long it took to fix the bug from the time it was reported so that the above conventions could be improved upon.

Modularity

- Modularity is the extent of partitioning of the source code and its refactoring into separate functions, classes, packages and components. Modularization shall aim at:
  - Avoiding code duplication
  - Advancing cohesion but minimizing coupling among functions, classes, packages and components so that a change in one of these sections of the software does not affect other sections
  - Ensuring that modules could be tested independently

Reusability

- Reusability is the level of molarity in the source code. In other words, it is reflection of how well each of the individual modules hide information e.g. implementation details and variables etc. to other modules.
- As a rule of thumb, if a piece of code is not usable outside of the original system then it is not reusable at all.
- Following OO paradigm religiously and taking advantage of its properties e.g. abstraction, encapsulation, inheritance and encapsulation inherently improve reusability, as an example.

Modifiability

- Modifiability is the ease of effort required to change a section of the software and thereafter test all other affected pieces.
- Modifiability improves with decoupling among functions, classes, packages and components.
- Presence of structural constructs identified with OO Paradigm such as interfaces and abstract classes naturally implies extendibility and modifiability.
- How much a module interfaces with other modules will impact its modifiability.
- Modifiability of pieces of software can be deduced in terms of cyclomatic complexity.
- Impact on the regression testing following a change in the code is another measure of modifiability.

Testability

It is the ease of effort required to determine if each of the software requirements had been satisfied. Whether these tests were written before or after the code was written would influence the testability of the test suite.

From another perspectives, it is also measured in terms of how many unit tests have been written and how much test coverage these unit tests provide.

Maintainability Measures

- Given two implementations of an application, it should be possible to quantitatively decide which implementation is more maintainable than the other.
- Since the seminal work by Halstead several maintainability measures have been defined to measure maintainability. The MI (Maintainability Index) presented below was originally defined by Paul Oman and Jack Hagemeister.

  - MI = 171-5.2ln(HV)-0.23(CC)-16.2ln(LoC)+50.0sin*sqrt(2.46*COM).

- MI increases with COM (comments) but decreases with HV (Halstead Volume), CC (Cyclomatic Complexity) or LOC (Lines of Code).

  - Cyclomatic Complexity is a quantitative measure of the complexity of the code [Thomas J. McCabe in 1976]. It is determined by counting number of linearly independent paths through a program's source code.
  - Calculation of Halstead Volume requires determination of number of distinct operators (e.g. mathematical or Boolean operators etc.), number of distinct operands (e.g. variables and constants etc.), number of operator instances in the code and number of operand instances in the code [Maurice Halstead]

Other metrics ?

- how equitably the class functionality is distributed among its methods;
- how uniformly the instance variables are being referenced in the instance methods;
- how coupled is the class to other classes ;
- how long is the hierarchy of super-classes;
- how deep is the hierarchy of sub-classes;
- how big is the class in terms of its instance variables and methods total etc.
- how big is the package
- how many interfaces and abstract classes in the package
- coupling among the classes in the package

- Maintainability Analysis Tools:
  - Metrics Reloaded (Android)
  - Radon (Python)