Calculating Design Quality Metrics with

JDepend

Overview

According to the creators of JDepend, JDepend traverses Java class file directories and generates design quality metrics for each Java package. JDepend allows you to automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to manage package dependencies effectively.

JDepend is a tool for measuring the design of a programming systems product programmed in the Java programming language.

JDepend Calculated Design Quality Metrics

The Java classes are tested against three metrics, that determine the overall quality of the design of the product. These metrics are as follows, extensibility, reusability and maintainability. JDepend analyzes the number of classes and the interfaces, along with coupling, abstractness, instability, and the dependency of cycles. In JDepend, the design quality is key to a well functioning programming systems product.

Meaning of the Calculated Metrics

JDepend tests if a programming systems product can be updated without causing any complications in other parts of the programming systems product. In order to have a design with extensibility, different parts of the program have to be independent of other parts of the system. Also, the system needs to be written in a way that allows for the user to extend the system without having to write the system again from scratch. This leads right into the next metric, reusability. The system needs to be written with reuse constantly in mind, or in other words, the source code needs to be modular in nature, rather than a single main method. Finally, maintainability refers to the ability to maintain the system without having to re-write the entire system. Maintenance should be easy and should not require the chief programmer to break the current system in order to fix another part of the system.

The specific metrics that JDepend calculates—provided by JDepend's website—are as follows:

- Number of Classes and Interfaces

- The number of concrete and abstract classes (and interfaces) in the package is an indicator of the extensibility of the package.
- Afferent Couplings (Ca)
- The number of other packages that depend upon classes within the package is an indicator of the package's responsibility.
- Efferent Couplings (Ce)
- The number of other packages that the classes in the package depend upon is an indicator of the package's independence.
- Abstractness (A)
- The ratio of the number of abstract classes (and interfaces) in the analyzed package to the total number of classes in the analyzed package.
- The range for this metric is 0 to 1, with A=0 indicating a completely concrete package and A=1 indicating a completely abstract package.
- Instability (I)
- The ratio of efferent coupling (Ce) to total coupling (Ce + Ca) such that I = Ce / (Ce + Ca). This metric is an indicator of the package's resilience to change.
- The range for this metric is 0 to 1, with I=0 indicating a completely stable package and I=1 indicating a completely instable package.
- Distance from the Main Sequence (D)
- The perpendicular distance of a package from the idealized line A + I =
- This metric is an indicator of the package's balance between abstractness and stability.
- A package squarely on the main sequence is optimally balanced with respect to its abstractness and stability. Ideal packages are either completely abstract and stable (x=0, y=1) or completely concrete and instable (x=1, y=0).
- The range for this metric is 0 to 1, with D=0 indicating a package that is coincident with the main sequence and D=1 indicating a package that is as far from the main sequence as possible.
- Package Dependency Cycles
- Package dependency cycles are reported along with the hierarchical paths of packages participating in package dependency cycles.

Better the Design of a System

The three aforementioned metrics are all very important in designing a high-quality and easily maintained programming systems product. Often, you are not continuously working on the same system, therefore, you need to make sure that you system makes sense and is written in a way that makes sense to people other than yourself. If you strive to achieve extensibility, reusability, and maintainability, the higher quality your programming systems product will be because it will last the test of time.