Package principles

In computer programming, **package principles** are a way of organizing classes in larger systems to make them more organized and manageable. They aid in understanding which classes should go into which packages (package cohesion) and how these packages should relate with one another (package coupling). Package principles also includes software package metrics, which help to quantify the dependency structure, giving different and/or more precise insights into the overall structure of classes and packages.

Contents

Overview

Principles of package cohesion

Reuse-release Equivalence Principle (REP)

REP essentially means that the package must be created with reusable classes – “Either all of the classes inside the package are reusable, or none of them are”. The classes must also be of the same family. Classes that are unrelated to the purpose of the package should not be included. A package constructed as a family of reusable classes tends to be most useful and reusable.

Common-Reuse Principle (CRP)

The CRP states that classes that tend to be reused together belong in the same package together. It is a way of helping to decide which classes belong in which package.
When depending on a package, one wants to make sure that the classes are inseparable and interdependent, which is also handy when culling classes that do not belong in the package.

Common-Closure Principle (CCP)

CCP states that the package should not have more than one reason to change. If changes were to happen in an application dependent on a number of packages, ideally one only want changes to occur in one package, rather than in a number of them.
This helps to identify classes that are likely to change, and package them together for the same reasons. If the classes are tightly coupled, they belong to the same package.

Principles of package coupling

Acyclic Dependencies Principle (ADP)

In a development cycle with multiple developers, cooperation and integration needs to happen in small incremental releases. The ADP states that there can be no cycles in the dependency structure, and that when an incremental release is made, the other developers can adopt and build upon it.

Stable-Dependencies Principle (SDP)

Designs, by nature of the environments they are used in or by, are changing. Thus, package design needs to support change as well. The SDP states that any packages one wants to be volatile should not be depended upon by a package that is difficult to change.

Stable-Abstractions Principle (SAP)

The SAP says that a stable package should also be abstract, so that its stability does not prevent it from being extended.
It also states that an unstable package should be concrete, since its instability allows the concrete code within it to be easily changed.

See also

- SOLID
- Robert Cecil Martin

References

- "Principles of OOD" (http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod).
- Martin , Robert C. (2002). *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall. ISBN 978-0135974445.
- "Granularity in Package Principles" (http://www.objectmentor.com/resources/articles/granularity.pdf) (PDF).
- "Stability in Package Principles" (http://www.objectmentor.com/resources/articles/stability.pdf) (PDF).