The designed to handle the printing and avoid

Business, Management

The Single Responsibility Principle: Combine those things that changefor the same reason, and separate those things that change for differentreasons, this principle is called SRP. The SRP is a key principle in the design off a class withinan object-oriented programming language.

The SRP was formulated by Robert C. Martin in his first book, Agile Software Development (1), and states rathersimply that a class should have at most one responsibility and only ever onereason to change. As a more concrete example, consider a program that takes indata and performs a calculation on that data. The developer creating theprogram could consider one class that handles the input and calculation, but thisclearly violates the first portion of the SRP, because at a minimum we shouldseparate the program into 2 classes. The first class will handle taking in thedata and the second class will perform the calculations. Farther, consider a developer who has been given the task ofadding a printing feature to the program. When designing the feature, the codecould be added on to one of the existing classes, but this would violate thatthere is only one reason for a class to change.

In these cases, the first andsecond classes, our input and calculation classes, should only change ifadditional improvements are needed within that functionality. So a third classshould be designed to handle the printing and avoid violating the one reason tochange idea. In statically typed and compiledlanguages, several reasons may lead to several, unwanted redeployments. Ifthere are two different reasons to change, it is conceivable that two differentteams may work on the same code for two different reasons. Each will have todeploy its solution, which in the case of a compiled language (like C++, C# orJava), may lead to incompatible modules with other teams or other parts of theapplication. Even though you may not use a compiled language, you may need toretest the same class or module for different reasons. This means more QA work, time, and effort (2). Determining the one single responsibility a class or module should haveis much more complex than just looking at a checklist.

For example, one clue tofind our reasons for change is to analyze the audience for our class. The usersof the application or system we develop who are served by a particular modulewill be the ones requesting changes to it. Those served will ask for change. Here are a couple of modules and their possible audiences.

Persistence Module – Audience include DBAs and software architects. Reporting Module – Audience include clerks, accountants, and operations. Payment Computation Module for a Payroll System – Audience may include lawyers, managers, and accountants. Book Search Module for a Library Management System – Audience may include the librarian and/or the clients themselves. Let’s lookat a real example (2). Associating concrete persons to allof these roles may be difficult. In a small company a single person may need tosatisfy several roles while in a large company there may be several personsallocated to a single role. So it seems much more reasonable to think about theroles.

But roles by themselves are quite difficult to define. What is a role? How do we find it? It is much easier to imagine actors doing those roles andassociating our audience with those actors. So if ouraudience defines reasons for change, the actors define the audience. Thisgreatly helps us to reduce the concept of concrete persons like “ John thearchitect” to Architecture, or “ Mary the referent” to Operations(2). So a responsibility isa family of functions that serves one particular actor. (Robert C. Martin)

`Filters are another great example of SRP.`