Separation of Concern vs Single Responsibility Principle (SoC vs SRP)

Monday, January 26, 2009

.NET (/arturtrosin/Tags/.NET) C# (/arturtrosin/Tags/C%23) Design Principles (/arturtrosin/Tags/Design%20Principles)

These two great principals that stands on the base of many design and architecture decisions. We very often meet these principals in book, articles, blogs, etc... And main question which risen in my head was what they really all about and how they relate to each other? These two principles are totally discrete from each other or their core principle is the same?
Just to remember what these two principles means lets read following statements which defines the each of them:

Separation of Concerns (SoC) – is the process of breaking a computer program into distinct features that overlap in functionality as little as possible. A concern is any piece of interest or focus in a program. Typically, concerns are synonymous with features or behaviors.
http://en.wikipedia.org/wiki/Separation_of_concerns (http://en.wikipedia.org/wiki/Separation_of_concerns)

Single Responsibility Principle (SRP) – every object should have a single responsibility, and that all its services should be narrowly aligned with that responsibility. On some level Cohesion is considered as synonym for SRP.
http://en.wikipedia.org/wiki/Single_responsibility_principle
(http://en.wikipedia.org/wiki/Single_responsibility_principle)

From the first sight very simple and easy to understand. The principles really have a lot in common and in general they both talk about decoupling in distinct Logical Units with well defined Boundaries (responsibilities). Logical Unit and Boundaries could be very abstract or concrete, and the boundaries depend on the **context** of the problem which you are trying to solve.

The separation allows:

- To allow people to work on individual pieces of the system in isolation;
- To facilitate reusability;
- To ensure the maintainability of a system;
- To add new features easily;

- To enable everyone to better understand the system;
- Etc...;

And of course, SoC is not limited to Architecture Layers, it's also applied on many other things, such as: an object that could represent a concern from language point view, SOA can separate concerns into services, separating behaviour as concern in logical units, etc...
If to discuss further about similarities, SRP mostly means the same thing as SoC for layered architecture example.

However, SRP was re-interpreted by Uncle Bob with the definition "THERE SHOULD NEVER BE MORE THAN ONE REASON FOR A CLASS TO CHANGE". By the definition, SRP was narrowed down to class level.

The "narrowed" SRP definition also could leave a lot of questions, how we can make sure that our new class really has one responsibility and there is only one reason to change? Similar question we could raise for SoC… What is for sure is that is not one answer, but in general the answer is that there is not a Rule and all depends on the context of the problem. You can find an answer by yourself by answering next questions:

1. What is really the **boundary** of the Responsibility that you are trying to separate?
2. What is really the **boundary** of the Concern that you are trying separate?

In real-world applications is mostly impossible to implement an ideal solution, trying to solve by one principle you can break another, such as YAGNI or over design…
Is very important to notice that: do not shift all the principles to extremes, because in real cases is impossible to achieve them from all point of views. So is very important to apply them from a certain and most meaningful angle.
The principles on certain level have similarities but on another level they are different things, let's find out it by an example.
Let's suppose we have following user story, "*user should be able set screen background color because the color should be customizable for any logged on user, so user access must be checked before color is set*".

Here is very raw implementation (just for demonstration purpose) of the requirement:

```
internal class UserSettingsService
{
    public void SetBackgroundColor(ConsoleColor color)
    {
        CheckAccess();

        Console.BackgroundColor = color;
        Console.WriteLine("- Color is changed...");
```

```
    }

    private static void CheckAccess()
    {
        if (IsCurrentUserLogedIn())
        {
            throw new SecurityException("Can't change color."
                                        + "The User is not Authenticated in the
system");
        }
    }

    private static bool IsCurrentUserLogedIn()
    {
        return Thread.CurrentPrincipal.Identity.IsAuthenticated;
    }
}
```

Main purpose of the SetBackgroundColor method is to set new background color but before to set a new color we check if the user has access. Security access checking is encapsulated by CheckAccess method. After short explanation, let's go back to our lovely SoC and SRP, on method level each method has its own concern and single responsibility. However on class level they are not, the class is far from to be a cohesive one, it encapsulates Security checks and changing screen color logic. But what if we need to do same check in another classes? then the Check Access logic is duplicated OR we can change the Check Access to public and use the class. Of course, both ways are not the best practices.
Let's do next step of refactoring and separate each class with its own responsibilities:

```
internal class UserSettingsService
{
    public void SetBackgroundColor(ConsoleColor color)
    {
        SecurityService.CheckAccess();

        Console.BackgroundColor = color;
        Console.WriteLine("- Color is changed...");
    }
}

internal class SecurityService
```

```
{
    public static void CheckAccess()
    {
        if (IsCurrentUserLogedIn())
        {
            throw new SecurityException("Can't change color."
                                + "The User is not Authenticated in the system");
        }
    }

    private static bool IsCurrentUserLogedIn()
    {
        return Thread.CurrentPrincipal.Identity.IsAuthenticated;
    }
}
```

Now the classes are separated so each class has its own responsibility:

1. UserSettingsService - to set background
2. SecurityService - to check security

which increases their reusability and therefore maintainability.
So in the last example SRP is not broken because each class has its own well defined responsibility. From another perspective if we take the security checks as infrastructure related concerns and changing color as domain concerns then SoC is broken, because set color invokes security checks instead just setting the color. We can avoid pollution of the domain logic with infrastructure stuff, by applying Virtual Proxy Pattern which will wrap *UserSettingsService* class and will perform security checks before “real” *SetBackgroundColor* method is invoked.

The security checks such in the example above are considered Cross-Cutting Concerns and that is why **AOP** (Aspect Oriented Programming) was invented for. AOP tries to separate such concerns in separate component(s) and apply and reuse the component(s) without introducing dependencies on the components making easier maintenance.
Conclusion, the SRP and SoC principles have something in common but in the same time they have some differences, also they should be viewed as an advice and not a rule, a developer should feel when and how to apply and do abstract concerns and responsibilities for a problem correctly.

[**Note**: I assume that provided example could not be ideal or the best one but I hope at least it shows my idea.]

Thank you,

Artur Trosin

6 Comments

Well articulated explanation. Thanks for knowledge sharing Artur!

— **Chitra** - Wednesday, September 22, 2010 6:37:37 PM (/arturtrosin/separation-of-concern-vs-single-responsibility-principle-soc-vs-srp#comment-108)

thanks for the article

— **Huy** - Friday, March 18, 2011 11:09:43 AM (/arturtrosin/separation-of-concern-vs-single-responsibility-principle-soc-vs-srp#comment-109)

Nice article....learned some new stuff. Thank you very much :-)

— **Sayem (http://ellidanreize.wordpress.com/)** - Wednesday, May 18, 2011 2:57:41 PM (/arturtrosin/separation-of-concern-vs-single-responsibility-principle-soc-vs-srp#comment-110)

Separation of concern vs single responsibility principle soc vs srp.. OMG! :)

— **weblogs.asp.net (http://massdents.info/weblogs.asp.net/)** - Wednesday, May 25, 2011 4:12:40 PM (/arturtrosin/separation-of-concern-vs-single-responsibility-principle-soc-vs-srp#comment-111)

Great post. I especially like how you concluded it.

— **Andrew** - Friday, March 1, 2013 3:35:40 PM (/arturtrosin/separation-of-concern-vs-single-responsibility-principle-soc-vs-srp#comment-113)

v.nice explaination

— **jamali** - Saturday, March 30, 2013 11:49:44 AM (/arturtrosin/separation-of-concern-vs-single-responsibility-principle-soc-vs-srp#comment-114)

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.