What Is the Difference Between SOC and SRP?

What is the difference between SOC (Separation of Concerns) and SRP (Single Responsibility Principle)?
? Jump down to “Peter’s Thoughts” to get straight to the answer

This is a question I get a lot in my technical trainings, so here is the answer from my viewpoint.

Lets start with some definitions…

Definitions

SRP

Robert C. Martin, the originator of the term, expresses the principle as,
“**A class should have only one reason to change**“.
Because of confusion around the word “reason” he has also clarified saying that the “principle is about people.”[3] In some of his talks, he also argues that the principle is, in particular, about roles or actors. For example, while they might be the same person, the role of an accountant is different from a database administrator. Hence, each module should be responsible for each role.

Emphasis mine, from https://en.wikipedia.org/wiki/Single-responsibility_principle

SOC

In computer science, separation of concerns is a design principle for
separating a computer program into distinct sections.
Each section addresses a separate concern, a set of information that affects the code of a computer program.
A concern can be as general as “the details of the hardware for an application”, or as specific as “the name of which class to instantiate”.

A program that embodies SoC well is called a modular[1] program. Modularity, and hence separation of concerns, is achieved by encapsulating information inside a section of code that has a well-defined interface. Encapsulation is a means of information hiding.

Emphasis mine, from https://en.wikipedia.org/wiki/Separation_of_concerns

Peter’s Thoughts

The principles have a lot in common, but what I think is the biggest difference is
#1 the timing when you think about those.
The #2 big difference is the flight level of the two principles.
The #3 difference is that “concerns” could be anything really and that’s why the flight levels are more generic.

#1 Timing – When do you consider it?

SOC is a principle that you think hard **when you start** with something new or during refactoring. I think of Separation Of Concerns as a process.
SRP is a principle that applies **after the fact**. You have something (class, package, namespace, module) in front of you and you think: “Does this too much?”

#2 Flight Level

SOC is very generic that can be applied on all levels.
SRP applies to functions, classes and modules, with the focus on outside responsibility.

Examples for “SOC applied on all levels”:

- Where do we put validation logic in our code base?
    - Is that a concern that should be in a general place?

  - Is this done automagically by our framework that does this via annotations or attributes

  - Is this done in every outer boundary method manually and we have code reviews to ensure that concern is there?

- The Separation of Functional Core and Imperative Shell is applying the SOC principle

#3 The terms “responsibility” VS “concern”

- “**responsibility**” reminds me of OO (see flight levels)

- “**concern**” could be anything in my language world: activity, task, goal, responsibility, request, pure…
  It’s more about “where is this concern visible or delegatable to”

Here you go… 3 reasons how we might distinguish the two principles ….

? Additional Thoughts

From my developer community (Thanks Urs Enzler and Marko Markovi?!) I got the following additional inputs:

- I prefer to think about cohesion instead of SRP or SOC

- SRP as business behaviour and SOC as technical implementation

- SRP is an instance of SOC in software design. SOC is a broad principle.

- I view SoC as meta-principle (i.e. a more generic, principle-of-principles) whereas other princes, like SOLID, or DRY, even LoD, or SLAP. are specific principles.
  Specific instances (principles) that fall under the larger umbrella of SoC include:
  - Separate Interface from Implementation
  - Separate Data from Representation
  - Command-Query Separation (CQRS)
  - Interface Segregation (ISP)
  - Separate Construction from Use
  - Separate Business logic/layer/domain from <> (e.g., solution | presentation | …)

???? You?

How do you distinguish these 2 principles?
What makes you think of them differently?

We Go Beyond Agile

Beyond Agility
Zürich - Switzerland