- Analysera era klasser med avseende på Separation of Concern (SoC) och Single Responsibility Principle (SRP).
    - Vilka ansvarsområden har era klasser?
    - Vilka anledningar har de att förändras?
    - På vilka klasser skulle ni behöva tillämpa dekomposition för att bättre följa SoC och SRP?

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.

The **single responsibility principle** (**SRP**) is a computer programming principle that states that "A module should be responsible to one, and only one, actor."[1] The term actor refers to a group (consisting of one or more stakeholders or users) that requires a change in the module.

As an example, consider a module that compiles and prints a report. Imagine such a module can be changed for two reasons. First, the content of the report could change. Second, the format of the report could change. These two things change for different causes. The single responsibility principle says that these two aspects of the problem are really two separate responsibilities, and should, therefore, be in separate classes or modules. It would be a bad design to couple two things that change for different reasons at different times.

The reason it is important to keep a class focused on a single concern is that it makes the class more robust. Continuing with the foregoing example, if there is a change to the report compilation process, there is a greater danger that the printing code will break if it is part of the same class.

Vilka ansvarsområden har era klasser?

Position:

- Håller två värden, x och y, som beskriver positionen.
- Innehåller också getters och setters
- Samt funktionalitet för att beräkna avståndet mellan två points

Vehicle:

- Hanterar alla olika vehicles funktionalitet när det gäller position, hastighet, hästkrafter. (inte hur hastigheten ska ökas och minskas dock)
- Innehåller getters och setters för alla grundläggande attribut som hastighet, position och färg.
- Även funktion för att förflytta fordon och byta färdriktning.

Volvo240

- Innehåller funktionalitet för att öka och minska hastigheten med hjälp av en trimFactor

Saab95

- Innehåller funktionalitet för att öka och minska hastigheten
- Implementerar en turbofunktion

Truck:

- Truck bygger vidare på Vehicle
- Lägger till attribut och funktionalitet för att hantera platformen för lastbilar samt specifik funktionalitet för att öka och minska hastigheten
- Lägger också till begränsningar för när man kan gasa och inte, endast när plattformen är nere

VehicleTransport:

- Innehåller funktionalitet för hur flaket lastas och vilken kapacitet det har.
- Innehåller funktionalitet för att kolla vad som kan laddas på flaket samt när man kan lasta grejer på flaket
- Implementerar en egen move funktion som även uppdaterar alla fordon på flakets positioner

Scania:

- Scania bygger vidare på Truck.
- Innehåller funktionalitet för att höja och sänka ett flak.

Bilverkstad

DrawPanel

-

Refaktoriseringsplan

- Dela upp koden i olika package. Ex ett för fordon
- Från början höll vi koll på positionen på två olika ställen. Både i alla vehicles men också i drawpanel. Det är onödigt och den refaktoriseringen påbörjade vi redan innan vilket man ser i vårt första UML diagram. (Enligt single responsibility principle)
- Enligt principen composition over inheritance så vill vi även se till att inte använda arv i onödan för att minska complexiteten och öka flexibiliteten i koden.
- Drawpanel borde bara måla ut vad vår view innehåller men nu skapas det instanser av bilverkstad och nya points i modellen där. Det bryter mot SRP och bör fixas. Det är även där vi hämtar alla bilder, men eftersom bilden för en bil är något som rimligtvis bör tillhöra bilen själv så borde detta rimligtvis hanteras i de olika vehichle klasserna som Saab och Volvo. Enligt SOC är det rimligt då vi vill dela upp koden så att allt som har med en volvo att göra sköts på ett ställe, inte på flera olika ställen.
- Nu funkar det så att all input från användaren skickas till CarView innan det skickas vidare till CarController. Eftersom att det är carcontroller som ska ha hand om alla

inputs och sen påverka viewn utifrån det så kan denna funktionaliteten rimligtvis flyttas dit också

- Vi tycker också att CarController fungerar lite konstigt nu. Det bygger till exempel upp hela modellen med bilar osv men med ett MVC mönster borde Modellen skötas i en egen modul. Vilket är rimligt enligt SRP. En modul borde sköta själva modellen, en modul tar emot user input som i sin tur uppdaterar modellen och skickar datan till våran view. Fixar hur bilden ska se ut och DrawPanel målar upp viewn för användaren.