Understanding the Single Responsibility Principle

Dan Geabunea
PASSIONATE SOFTWARE DEVELOPER | BLOGGER

@romaniancoder www.romaniancoder.com

Overview

What is the Single Responsibility Principle (SRP)?

Identify multiple reasons to change

Danger of having multiple responsibilities

Demo: Refactor to SRP

Single Responsibility Principle

Every function, class or module should have one and only one reason to change.

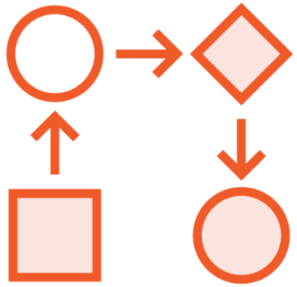
Examples of Responsibilities

Business logic

User interface

Persistence

Logging



Orchestration

Users

Always identify the reasons to change that your components have and reduce them to a single one.

Why Should You Use SRP?

It makes code easier to understand, fix, and maintain

Classes are less coupled and more resilient to change

More testable design

Identify Multiple Reasons to Change

If Statements

```
if(employee.getMonthlyIncome() > 2000){
    // some logic here
} else {
    // some other logic here
}
```

Switch Statements

```
switch(employee.getNbHoursPerWork()){
    case 40: {
        // logic for full time
    }
    case 20: {
        // logic for part time
    }
}
```

Monster Method

```
Income getIncome(Employee e){
    Income income = employeeRepository.getIncome(e.id);
    StateAuthorityApi.send(income, e.fullName);
    Payslip payslip = PayslipGenerator.get(income);
    JsonObject payslipJson = convertToJson(payslip);
    EmailService.send(e.email, payslipJson);
    …
    return income;
}
```

God Class

```
class Utils{
    void saveToDb(Object o){…}
    void convertToJson(Object o){…}
    byte[] serialize(Object o){…}
    void log(String msg){…}
    String toFriendlyDate(LocalDateTime date){…}
    int roundDoubleToInt(double val){…}
}
```

People

```
Report generate(){
    // method used by HR and Management actors
    // each one will want different features at some point
    // in time
}
```

SRP Example

```
class ConsoleLogger{
    void logInfo(String msg){
        System.out.println(msg);
    }

    void logError(String msg, Exception e){…}
}
```

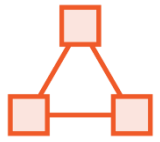
Danger of Multiple Responsibilities

Symptoms of Not Using SRP

Code is more difficult to read and reason about

Decreased quality due to testing difficulty

Side effects



High coupling

Coupling

The level of inter-dependency between various software components

Example

```
Income getIncome(Employee e){
    RepositoryImpl repo = new RepositoryImpl(srv,port,db);

    Income income = repo.getIncome(e.id);

    return income;
}
```

Example

```
Income getIncome(Employee e, Repository repo){
    Income income = repo.getIncome(e.id);

    return income;
}
```

If Module A knows too much about Module B, changes to the internals of Module B may break functionality in Module A.

Demo

Refactor a component with many responsibilities

- Identify the responsibilities
- Extract them out of the method by applying the SRP

Summary

- Correctly identify reasons to change
- The link between high coupling and code fragility
- Refactor responsibilities out to specialized components

“We want to design components that are self-contained: independent, and with a single, well-defined purpose”

Andrew Hunt & David Thomas, The Pragmatic Programmer