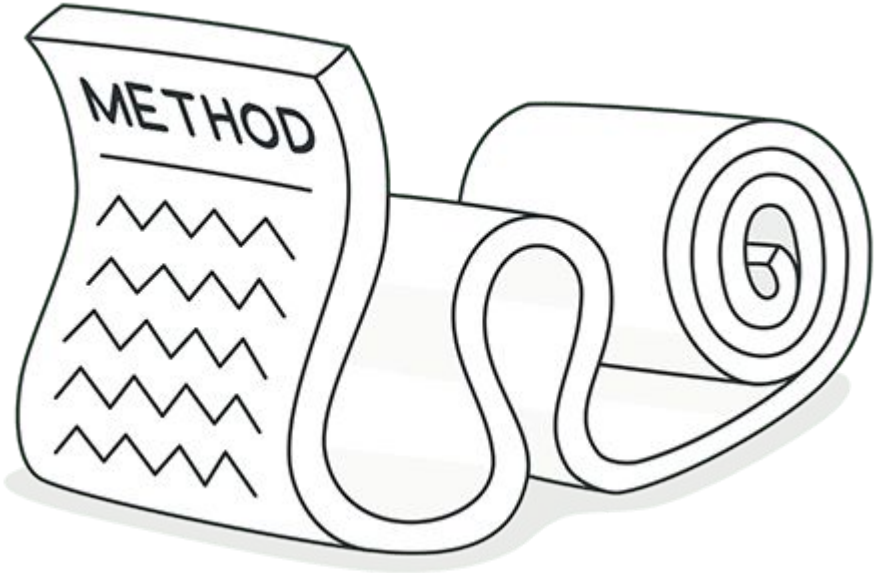
Refactoring

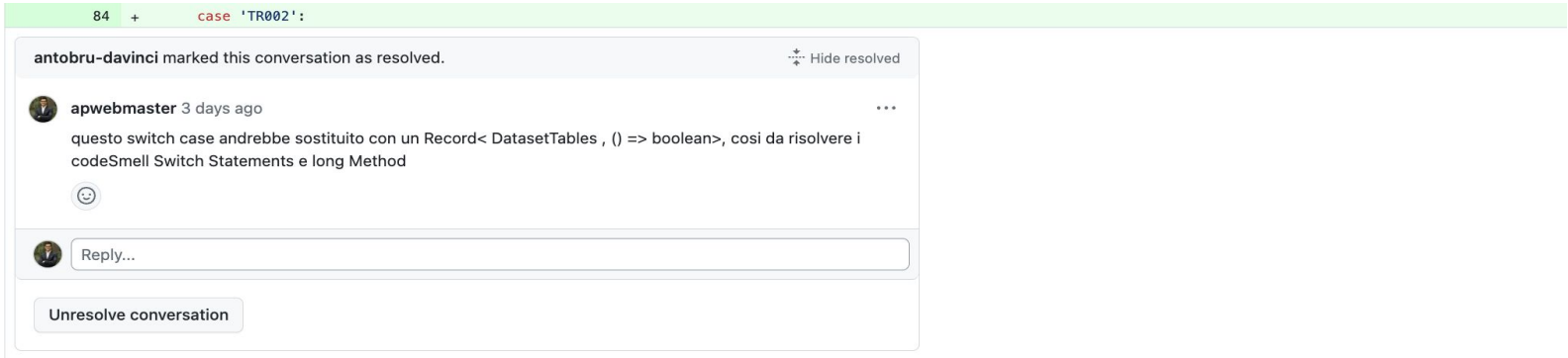
Durante una CR vedo che è stato necessario intervenire su uno switch…

Switch enorme (165 righe)….

Lo switch si trovava in uno script di importazioni di tabelle da farmadati e sostanzialmente aveva l'obiettivo, in base al nome della tabella, di effettuare delle operazioni e ritornare un booleano.



Decido dunque di prendere l’occasione di questa modifica per effettuare un piccolo refactoring…



Tra i vari Code smell individuati vi mostrerò:

- Switch Statements
- Long method
- Long Class

Successivamente al primo refactoring:

- Long Parameter List
- Data Clumps

1) Per prima cosa isolo lo switch in una funzione separata

```
Private processRecords (dataset, service1,service2, data, logger) {
     switch(dataset){...}
}
```

2) Scrivo i test per non rischiare di rompere la BL durante il refactoring.
3) Trasformo lo switch in un Record:

```
type DatasetTables = "XXXX" | "YYYY" | ... 18 more ... | "ZZZZZ"
type TablesUpdaterValue = (service1,service2, data, logger) => Promise<boolean>;


private tablesUpdater =Record<DatasetTables, TablesUpdaterValue> = {
  XXXX: (service1,service2, data, logger) => {..},
  YYYY: (dataset, service1,service2, data, logger) => {..},
  ZZZZ: (dataset, service1,service2, data, logger) => {..}
}
Private processRecords (dataset, service1,service2, data, logger) {
     return this.tablesUpdater[tableName](service1,service2, data, logger)
}
```

4) A quel punto creo una funzione separata per ogni BL inclusa dentro il case dello switch.
In questa maniera si è risolto il problema del “long method” ma praticamente ho dovuto introdurre:

- Data clumps
- Long parameter list

Perchè in ogni funzione ho dovuto passare 3 services + i dati…

```
private tablesUpater: Record<DatasetTables, TablesUpdaterValue> = {
  XXXX: this.functionXXXX()
  YYYY: this.functionYYYY()
  ZZZZ: this.functionZZZZ()
}
private functionXXXX() {
  return (service1, service2, data, logger) => {...}
}
Private functionYYYY() {
   return (service1, service2, data, logger) => {...}
}
```

5) Decido quindi di estrarre il metodo e il Record in una classe che ho chiamato TablesUpdaterValue e inietto i services necessari nel costruttore.

```
Export Class TablesUpaterClass {
  constructor(service1, service2){}
  private readonly logger = new Logger(TablesUpaterValue.name) }
  private tablesUpdater = Record<DatasetTables, TablesUpdaterValue> = {...}

  Private processRecords (dataset, data) {
    return this.tablesRecords[dataset](data)
  }

  private functionXXXX() {
     return (data) => {...}
  }
```

Come si può vedere il metodo `processRecords` prende in ingresso solo i dati e utilizza i services iniettati nel costruttore risolvendo quindi il “long parameter list” e il “data clumps”

Il risultato finale è che lo switch di 165 righe nella classe principale è diventata una chiamata di un rigo…

```
const result = await this.tablesUpaterClass.processRecords(dataset, data);
```

Inoltre è stato possibile togliere dalla classe principale le dipendenze spostate nel costruttore di `tablesUpaterClass`

Sono soddisfatto?

No, al momento la classe processa dati che gli vengono passati dall’esterno…

mi piacerebbe completare il refactoring passando i dati nel costruttore e poi chiamare un metodo senza dover passare nessun parametro

Se la domanda è: chi ha scritto questo codice? Non fatemela… altrimenti ditemi pure XD

Bibliografia:

- https://sourcemaking.com/refactoring/smells
- Vari immagini prese da google