Assignment 2

CSE2115: Software Engineering Methods

Group 06b

TU Delft

Assignment 2

by

Group 06b

Instructor:	Dr. A. Panichella & F. Mulder
Teaching Assistant:	M. Segers
Institution:	Delft University of Technology
Place:	Faculty of Electrical Engineering, Mathematics and Computer Science
Date:	16-01-2023

TU Delft

Contents

1 Metrics overview

Introduction

Code smells are warning signs of possible problems with a codebase, such as poor design or non-maintainability. Code smells in a project can be found and examined with the MetricsTree plugin. Developers may easily spot portions of the code that may need improvement and make modifications to raise the general caliber of the codebase by utilizing the MetricsTree plugin. Additionally, by identifying flaws early in the development process, the MetricsTree plugin can assist in preventing future problems. MetricsTree is a helpful plugin for maintaining and raising the caliber of a codebase, all things considered.

Method level metrics

1. LOC - Lines Of Code

Thresholds:

- Regular range upper bound: 11
- High range upper bound: 31
- Very-high range upper bound: 47

Justification: For method level metrics, we all decided that LOC is one of the best indicators there is to reflect the granularity of the methods. It will show us how maintainable this code will be for the future and how concentrated the functionality of the method is. Many of our methods violated this metric, so it was natural to consider it. We kept the thresholds provided by IntelliJ MetricsTree, as we felt it's a good representation of the complexity of the methods. A good amount of methods were found problematic in this aspect, such as: `cancelOrder`, `placeOrder`, `addTopping`, etc. To keep this metric to a minimum, we refactored our code with techniques like method extraction and logic simplification.

2. MCC - McCabe's Cyclomatic Complexity

Thresholds:

- Regular range upper bound: 3
- High range upper bound: 5
- Very-high range upper bound: 7

Justification: Cyclomatic complexity is quite literally a measure of the methods complexity(Wijendra and Hewagamage (2021)), it signifies the amount of possible branching paths in a method which fairly directly translates how difficult a method is to understand.

Henceforth we have chosen to use this metric to refactor our code. Based on the context of our project and preexisting sources(Filó et al. (2015)) we have determined that these thresholds are a good benchmark for our project. Due to the nature of our project we have a lot of simple if statements to check for exceptions, thereby we made the metrics a bit higher as those statements do not add to complexity too much. Anything under 3 is easily understandable, between 3 and 5 is normal, 5 and 7 warrants attention as it is higher than normal and may be confusing and anything above 7 needs to be refactored.

3. NOP - Number Of Parameters

Thresholds:

- Regular range upper bound: 3
- High range upper bound: 4
- Very-high range upper bound: 5

Justification: This metric measures the number of parameters of a method or constructor. We chose this metric because it would help us identify complex methods that are difficult to call and extend. According to CSE2115 professors methods with more than 4 parameters should be considered as Long parameters lists.

Class level metrics

1. WMC - Weighted Methods per Class

Thresholds:

- Regular range upper bound: 11
- High range upper bound: 34
- Very-high range upper bound: 44

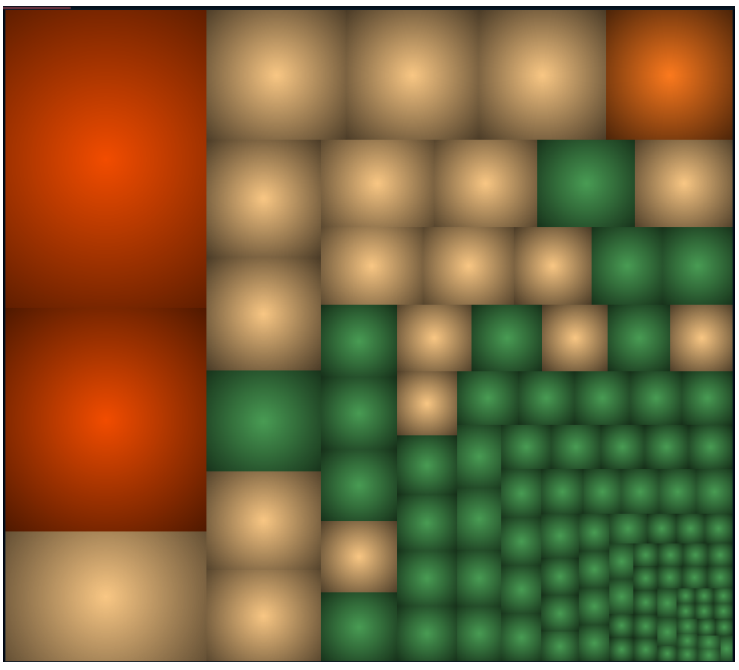


Figure 1.1: WMC before refactoring

Justification: We chose to refactor based on the Weighted Methods per Class since a few of our classes had really high WMC's compared to our highest threshold of 45. For example, the `MenuService` class that we refactored to lower the WMC had a count of 76 which we felt makes the class really bloated and hard to understand what was going on. We felt that the WMC is a good measure of readability of code since it relies on the cyclomatic complexity and the number of lines of code, both of which have a really high impact on the readability of the code since a high cyclomatic complexity is hard to figure out the logic of and the high lines of code can seem overwhelming and hard to navigate. We chose our thresholds to be similar to the thresholds from a paper(Filó et al., 2015), however, our high and very high range are slightly more forgiving as we feel that these ranges provide a better fit for the legibility of code in our application.

2. CBO - Coupling Between Objects

Thresholds:

- Regular range upper bound: 14
- High range upper bound: 17
- Very-high range upper bound: 23

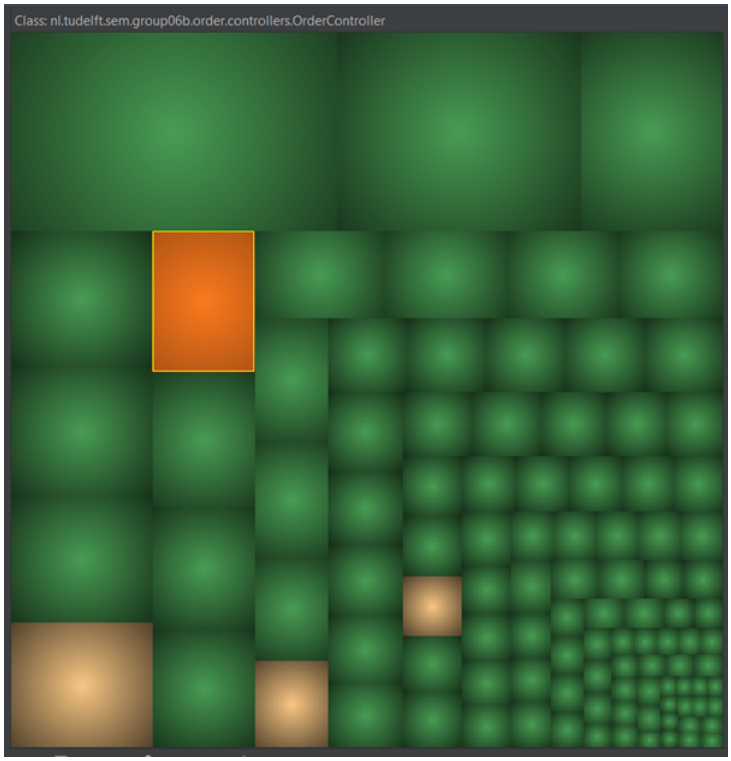


Figure 1.2: CBO before refactoring

Justification: CBO measures number of dependencies with other classes. It counts the unique number of reference types that occur through method calls, method parameters, return types, thrown exceptions and accessed fields. This metrics was chosen based on the advice of our professors since it can detect highly coupled classes, which could prevent reuse of existing components and hinder a modular, encapsulated software design. Furthermore, classes would be more sensitive to changes in other classes and thus less maintainable and error-prone. Additionally high coupling would make it harder to test in isolation. To improve this metric CBO should be kept to a minimum.

Only one class, `OrderController`, was identified as problematic with CBO of 18 which puts it in high range. The thresholds are based on research Sahraoui et al., 2000 where a maximum CBO value of 14 is recommended, because higher values have negative impacts on several quality aspects of a class, which includes the maintainability, stability and understandability.

3. RFC - Response For Class

Thresholds:

- Regular range upper bound: 30
- High range upper bound: 50
- Very-high range upper bound: 70

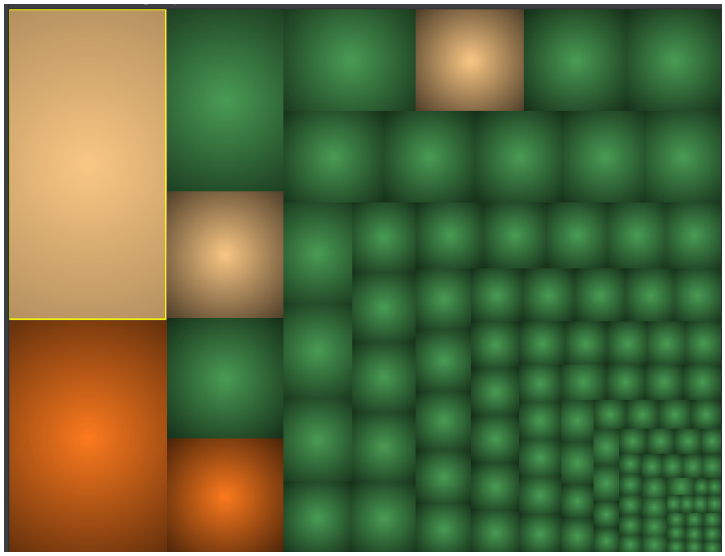


Figure 1.3: RFC before refactoring

Justification: The RFC (Response For Class) metric is an indicator that tells us how cohesive a class is. The higher the RFC value, the less cohesive the class. A class with low cohesion tells us that the class might be doing too many things and should be refactored into smaller chunks. For this reason, we decided to use this metric, as we believe that it is really important for having classes with good quality code and focused responsibility. A good range for RFC metric levels is below 30-45, based on IntelliJ MetricsTree default thresholds. Anything above 70 is really bad, and anything in between is something to look out for. The metrics tree shows that the classes `OrderProcessorImpl` and `OrderController` were the most problematic ones, with 73 and 68 RFC values respectively, which we decided it's something to improve on. Thus, these classes were chosen for refactoring.

4. LCOM - Lack of Cohesion of Methods

Thresholds:

- Regular range upper bound: 0.167
- High range upper bound: 0.558
- Very-high range upper bound: 0.725

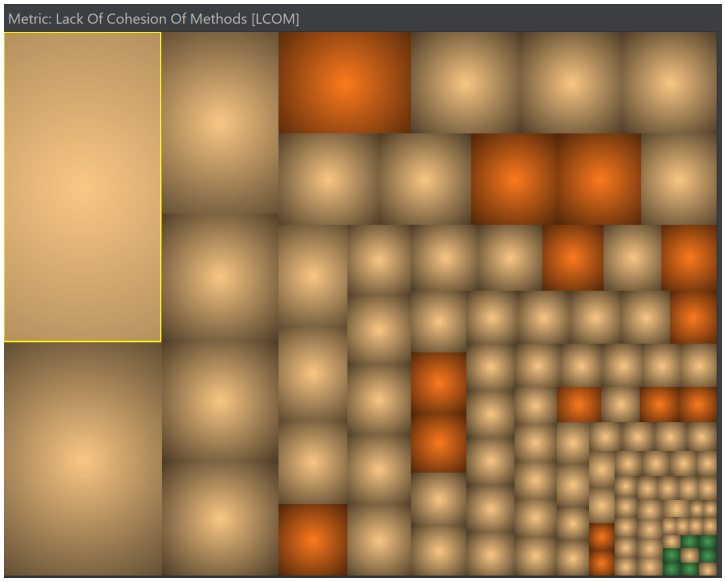


Figure 1.4: LCOM before refactoring

Justification: The LCOM(Lack of Cohesion in Methods) is a metric that measures the degree of cohesion inside a class. It counts the number of methods in a class that do not share data fields. As described by Filó et al. (2015) a good range for LCOM metric is below 0.167, anything above 0.725 should be refactored. When a class has low cohesion, developers may find it difficult to grasp its purpose and how it fits into the whole system. Additionally, bugs are more likely to appear in a class with low cohesion. Nevertheless, we have not chosen to refactor our class base on this metric. LCOM is based on the amount of properties shared among methods, which implies it does not consider the methods' functionality or purpose. Most classes in the MetricsTree analysis figure contain methods that perform related tasks but do not share any properties, this result in false alarms. LCOM also cannot distinguish between identical and dissimilar methods in the classes, which results in false positives.

5. DIT - Depth of Inheritance Tree

Thresholds:

- Regular range upper bound: 2
- High range upper bound: 4
- Very-high range upper bound: 6

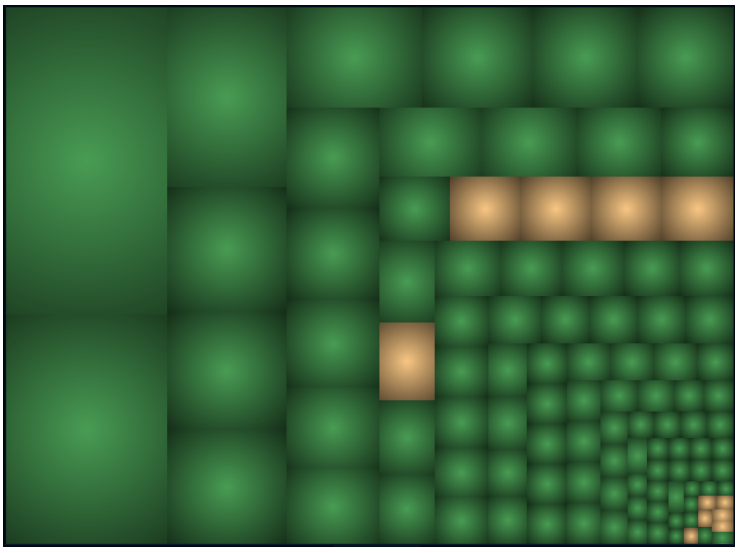


Figure 1.5: DIT before refactoring

Justification: The DIT measures the maximum depth of an inheritance tree when classes inherit each other. We chose not to refactor based on this metric as all of our classes (excluding the authentication classes) are in the recommended range. We chose these values for our thresholds as they are recommended by Filó et al. (2015) and we agree with these values as a team since we think that inheriting from 2 classes is acceptable but any more is questionable at best and could use refactoring.

6. NOC - Number Of Children

Thresholds:

- Regular range upper bound: 2
- High range upper bound: 4
- Very-high range upper bound: 7

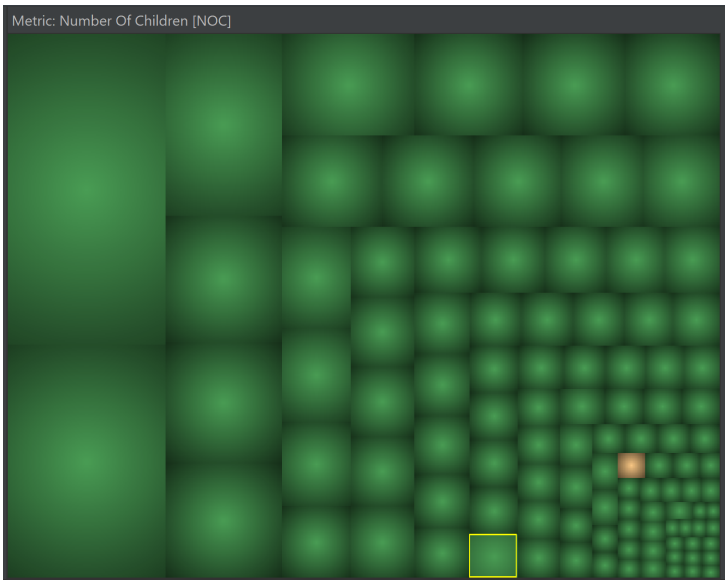


Figure 1.6: NOC before refactoring

Justification: The NOC(Number of Children) metric measures the number of direct children of a class. It is used to assess a class's inheritance structure. Improving the NOC measure, as suggested by the professors, can contribute to improving the overall quality and maintainability of our code. Based on IntelliJ MetricsTree, a good range for NOC metric is 0-2, anything above 7 should be refactored.Since it is nearly green everywhere from the MtricsTree analysis, we chose not to refactor based on this metric.

7. NOM - Number Of Methods

Thresholds:

- Regular range upper bound: 7
- High range upper bound: 15
- Very-high range upper bound: 25

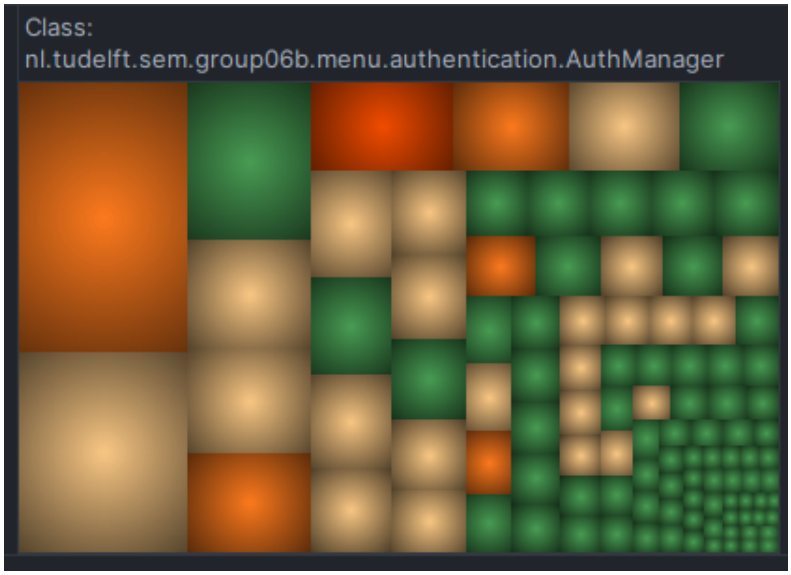


Figure 1.7: NOM before refactoring

Justification: We have chosen this metric as we believe that having an excessive number of methods in a single class can be overbearing to understand for someone who sees the class for the first time. Viewing the metric in the context of the project, we have a lot of routing/endpoint classes which simply redirect and delegate the actions to other classes, considering that, while the number of methods in those classes may be high it does not necessarily impact the readability of the class as each method is very short and simple. By looking at the classes we had, we had determined that anything below 7 methods is of no concern, anything between 7 and 15 should warrant some attention, anything between 15 and 25 definitely warrants refactoring as it is an unusually high amount of methods even if every method is short and readable individually. Anything above 25 was considered necessary to be refactored. These thresholds we have established by exploring established thresholds and tweaked that by exploring the NOM metric in the context of our project.

8. NOA - Number Of Attributes

Thresholds:

- Regular range upper bound: 4
- High range upper bound: 9
- Very-high range upper bound: 14

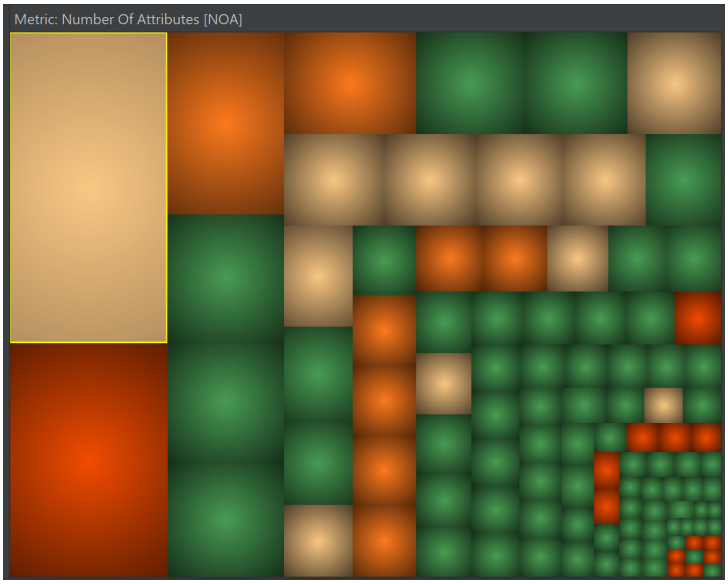


Figure 1.8: NOA before refactoring

Justification: The number of attributes of a class is a really good indicator of how the class is designed. A poorly designed class with a high number of attributes indicates that the class is too complex. Thus we decided that this metric is really important and should be taken into account. According to resources found on the internet and our own opinion, we decided that the thresholds currently set by IntelliJ MetricsTree were satisfactory. This means that any class with less than 4 global attributes is considered good, anything from 4 to 9 is acceptable, between 9 and 14 is a strong indicator to refactor, and above 14 is simply unmaintainable code. The metrics tree shows that the class OrderProcessorImpl was the most problematic one, with 14 attributes, which we considered unmaintainable. Thus, this class was chosen for refactoring.

2
Refactoring

Class level refactoring

1. OrderController

Identified metrics: CBO, NOM
Metric values before refactoring:

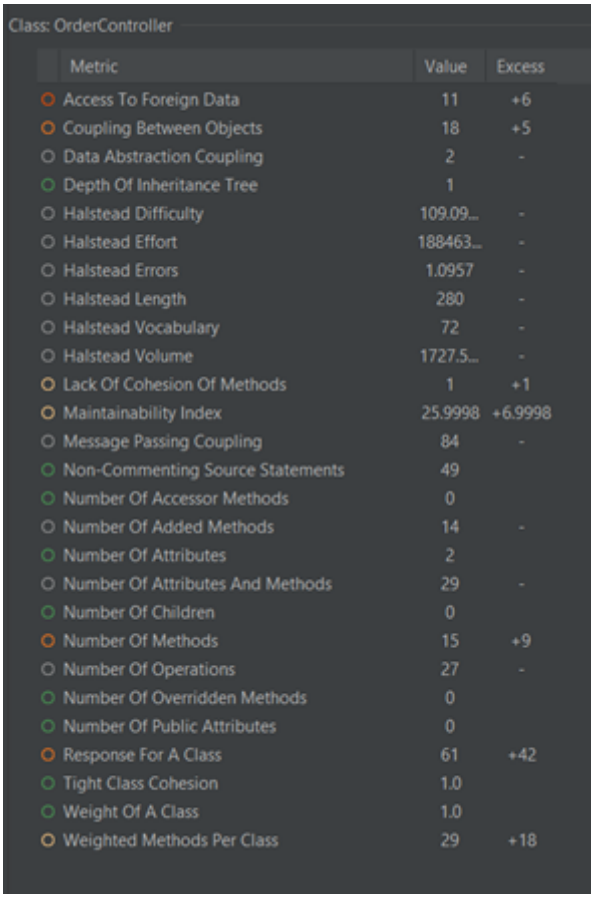


Figure 2.1: OrderController metrics before refactoring

Metric values after refactoring:

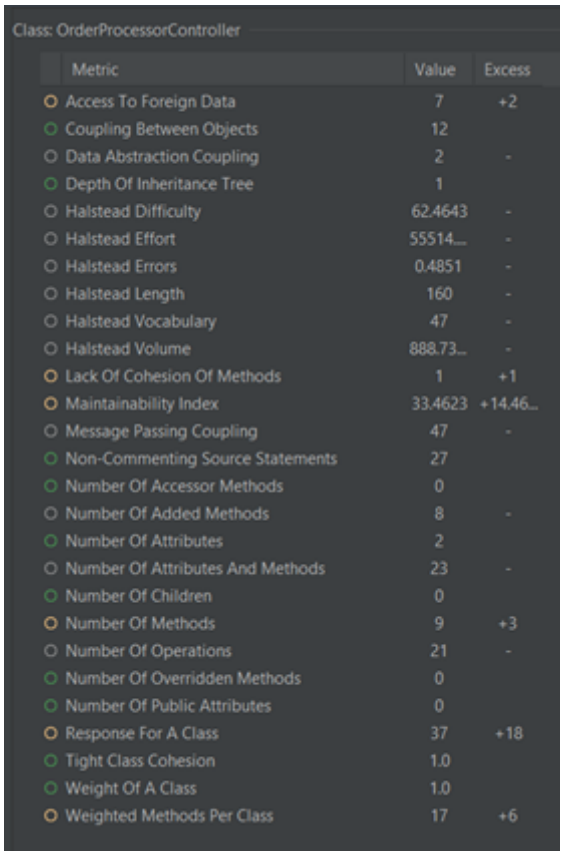


Figure 2.2: OrderProcessorController metrics after refactoring

Figure 2.3: OrderEditingController metrics after refactoring

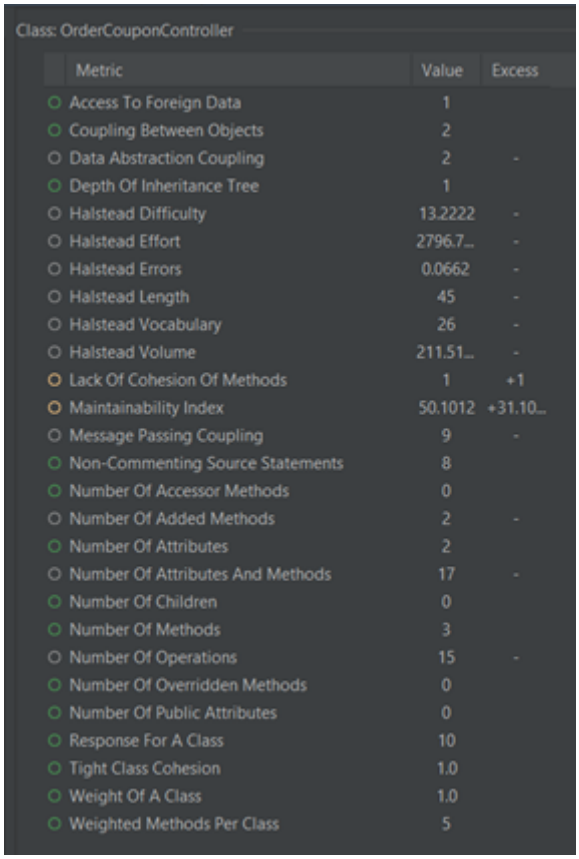


Figure 2.4: OrderCouponController metrics after refactoring

Code before refactoring:

```
public class OrderController {

    private final transient AuthManager authManager;

    private final transient OrderService orderService;

    /** Starts an empty order. ...*/
    @PostMapping("/start")
    public ResponseEntity<?> startOrder() {...}

    /** Changes the time of the order. ...*/
    @PutMapping("/change_time")
    public ResponseEntity<?> setOrderTime(@RequestBody SetOrderTimeRequestModel request) {...}

    /** Changes the location of the order. ...*/
    @PutMapping("/change_location")
    public ResponseEntity<?> setOrderLocation(@RequestBody SetOrderLocationRequestModel request) {...}

    /** Places an order. ...*/
    @PostMapping("/place")
    public ResponseEntity<PlaceOrderResponseModel> placeOrder(@RequestBody PlaceOrderRequestModel request) {...}

    /** Cancels an order. ...*/
    @PostMapping("/cancel")
    public ResponseEntity<?> cancelOrder(@RequestBody CancelOrderRequestModel request) {...}

    /** Gets an order. ...*/
    @GetMapping("/fetch_single")
    public ResponseEntity<FetchOrderResponseModel> fetchOrder(@RequestBody FetchOrderRequestModel request) {...}

    /** Gets all orders from a certain store. ...*/
    @GetMapping("/fetch_store")
    public ResponseEntity<FetchStoreOrdersResponseModel> fetchStoreOrders(
            @RequestBody FetchStoreOrdersRequestModel request) {...}
```

Figure 2.5: OrderController code before refactoring (1)

```
    /** Gets all orders from all stores. ...*/
    @GetMapping("/fetch_all")
    public ResponseEntity<FetchOrdersResponseModel> fetchOrders() {...}

    /** Adds a pizza to an order. ...*/
    @PostMapping("/add_pizza")
    public ResponseEntity<AddPizzaResponseModel> addPizza(@RequestBody AddPizzaRequestModel request) {...}

    /** Removes a pizza from an order. ...*/
    @DeleteMapping("/remove_pizza")
    public ResponseEntity<?> removePizza(@RequestBody RemovePizzaRequestModel request) {...}

    /** Adds a topping to a pizza. ...*/
    @PostMapping("/add_topping")
    public ResponseEntity<AddToppingResponseModel> addTopping(@RequestBody AddToppingRequestModel request) {...}

    /** Removes a topping from a pizza. ...*/
    @DeleteMapping("/remove_topping")
    public ResponseEntity<?> removeTopping(@RequestBody RemoveToppingRequestModel request) {...}

    /** Adds a coupon to the order. ...*/
    @PostMapping("/add_coupon/{orderId}/{couponId}")
    public ResponseEntity<?> addCoupon(@PathVariable long orderId, @PathVariable String couponId) {...}

    /** Removes a coupon from the order. ...*/
    @DeleteMapping("/remove_coupon/{orderId}/{couponId}")
    public ResponseEntity<?> removeCoupon(@PathVariable long orderId, @PathVariable String couponId) {...}
}
```

Figure 2.6: OrderController code before refactoring (2)

Code after refactoring:

```
public class OrderProcessorController {
    private final transient AuthManager authManager;
    private final transient OrderService orderService;
    /** Starts an empty order. ...*/
    @PostMapping("/start")
    public ResponseEntity<?> startOrder() {...}

    /** Places an order. ...*/
    @PostMapping("/place")
    public ResponseEntity<PlaceOrderResponseModel> placeOrder(@RequestBody PlaceOrderRequestModel request) {...}

    /** Cancels an order. ...*/
    @PostMapping("/cancel")
    public ResponseEntity<?> cancelOrder(@RequestBody CancelOrderRequestModel request) {...}

    /** Changes the time of the order. ...*/
    @PutMapping("/change_time")
    public ResponseEntity<?> setOrderTime(@RequestBody SetOrderTimeRequestModel request) {...}

    /** Changes the location of the order. ...*/
    @PutMapping("/change_location")
    public ResponseEntity<?> setOrderLocation(@RequestBody SetOrderLocationRequestModel request) {...}

    /** Gets an order. ...*/
    @GetMapping("/fetch_single")
    public ResponseEntity<FetchOrderResponseModel> fetchOrder(@RequestBody FetchOrderRequestModel request) {...}

    /** Gets all orders from a certain store. ...*/
    @GetMapping("/fetch_store")
    public ResponseEntity<FetchStoreOrdersResponseModel> fetchStoreOrders(
            @RequestBody FetchStoreOrdersRequestModel request) {...}

    /** Gets all orders from all stores. ...*/
    @GetMapping("/fetch_all")
    public ResponseEntity<FetchOrdersResponseModel> fetchOrders() {...}
}
```

Figure 2.7: OrderProcessorController code after refactoring

Process and result: NOM and CBO were improved in the same way. We extracted classes `OrderCouponController`, `OrderEditingController`, and `OrderProcessorController` from `OrderController` based on the semantically different functionality each one of them maps to. No changes in the tests were needed.

```java
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/order")
public class OrderEditingController {

    private final transient AuthManager authManager;

    private final transient OrderService orderService;

    /** Adds a pizza to an order. ...*/
    @PostMapping("/add_pizza")
    public ResponseEntity<AddPizzaResponseModel> addPizza(@RequestBody AddPizzaRequestModel request) {...}

    /** Removes a pizza from an order. ...*/
    @DeleteMapping("/remove_pizza")
    public ResponseEntity<?> removePizza(@RequestBody RemovePizzaRequestModel request) {...}

    /** Adds a topping to a pizza. ...*/
    @PostMapping("/add_topping")
    public ResponseEntity<AddToppingResponseModel> addTopping(@RequestBody AddToppingRequestModel request) {...}

    /** Removes a topping from a pizza. ...*/
    @DeleteMapping("/remove_topping")
    public ResponseEntity<?> removeTopping(@RequestBody RemoveToppingRequestModel request) {...}
}
```

Figure 2.8: OrderEditingController code after refactoring

```java
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/order")
public class OrderCouponController {

    private final transient AuthManager authManager;

    private final transient OrderService orderService;

    /** Adds a coupon to the order. ...*/
    @PostMapping("/add_coupon/{orderId}/{couponId}")
    public ResponseEntity<?> addCoupon(@PathVariable long orderId, @PathVariable String couponId) {
        try {
            orderService.addCoupon(authManager.getToken(), orderId, couponId);
            return ResponseEntity.ok().build();
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.EXPECTATION_FAILED, e.getMessage());
        }
    }

    /** Removes a coupon from the order. ...*/
    @DeleteMapping("/remove_coupon/{orderId}/{couponId}")
    public ResponseEntity<?> removeCoupon(@PathVariable long orderId, @PathVariable String couponId) {
        try {
            orderService.removeCoupon(orderId, couponId);
            return ResponseEntity.ok().build();
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.EXPECTATION_FAILED, e.getMessage());
        }
    }
}
```

Figure 2.9: OrderCouponController code after refactoring

2. CouponServiceImpl

Identified metrics: NOM, WMC, RFC
Metric values before refactoring:

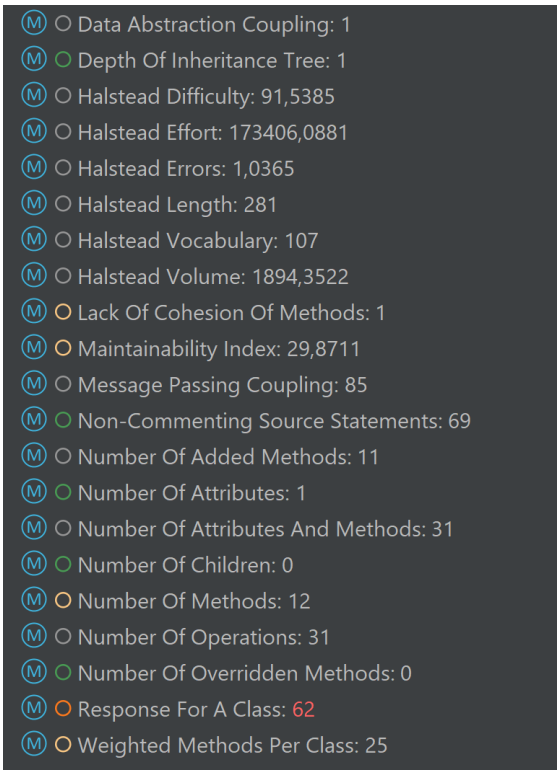


Figure 2.10: CouponServiceImpl before refactoring

Metric values after refactoring:

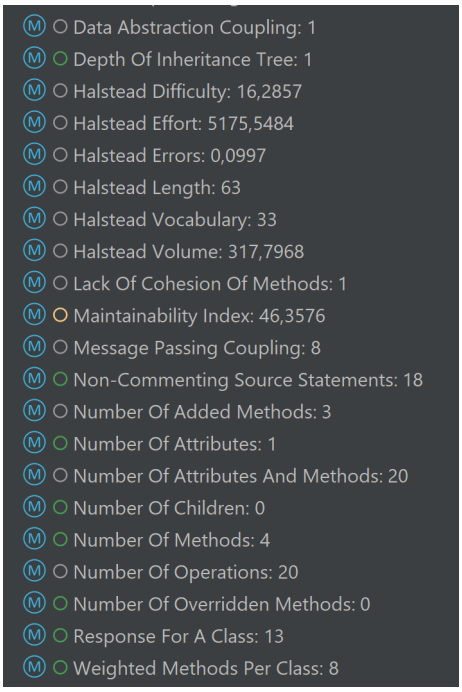


Figure 2.11: CouponsManagementServiceImpl after refactoring

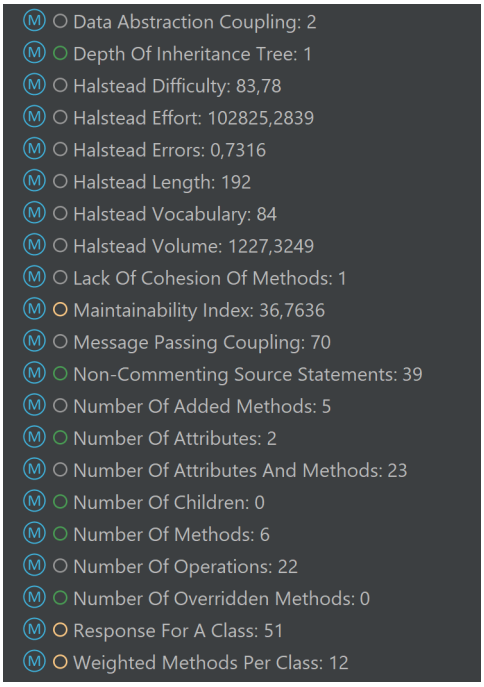


Figure 2.12: CouponsOperationsServiceImpl after refactoring

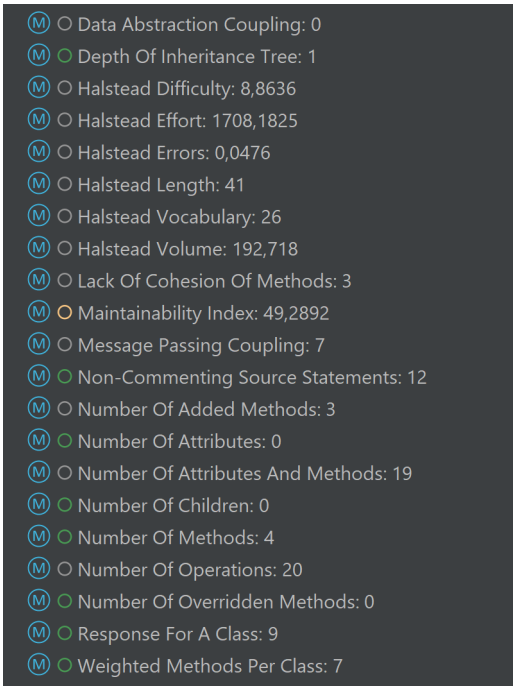


Figure 2.13: BasketValidationService after refactoring

Code before refactoring:

```
public interface CouponsService {

    /**
     * Adds a new coupon to the database.
     *
     * @param couponId       the id of the coupon
     * @param couponType     the type of the coupon
     * @param discount       the discount of the coupon
     * @param expirationDate the expiration date of the coupon
     */
    void addCoupon(String couponId, String couponType, double discount, Date expirationDate);
    /**
     * Removes the coupon from the database if it exists.
     *
     * @param couponId the id of the coupon
     */
    void removeCoupon(String couponId);
    /**
     * Queries the database for all coupons.
     *
     * @return all the coupons
     */
    List<Coupon> queryAllCoupons();
    /**
     * Checks if a coupon is in the repository and is available.
     *
     * @param couponId the id of the coupon
     * @return true if the coupon is in the repository
     */
    boolean isCouponAvailable(String couponId, String memberId);

    void useCoupon(String couponId, String memberId);

    ApplyCouponsRequestModel calculatePrice(ApplyCouponsRequestModel pizzasAndCoupons);
}
```

Figure 2.14: CouponServiceImpl code before refactoring

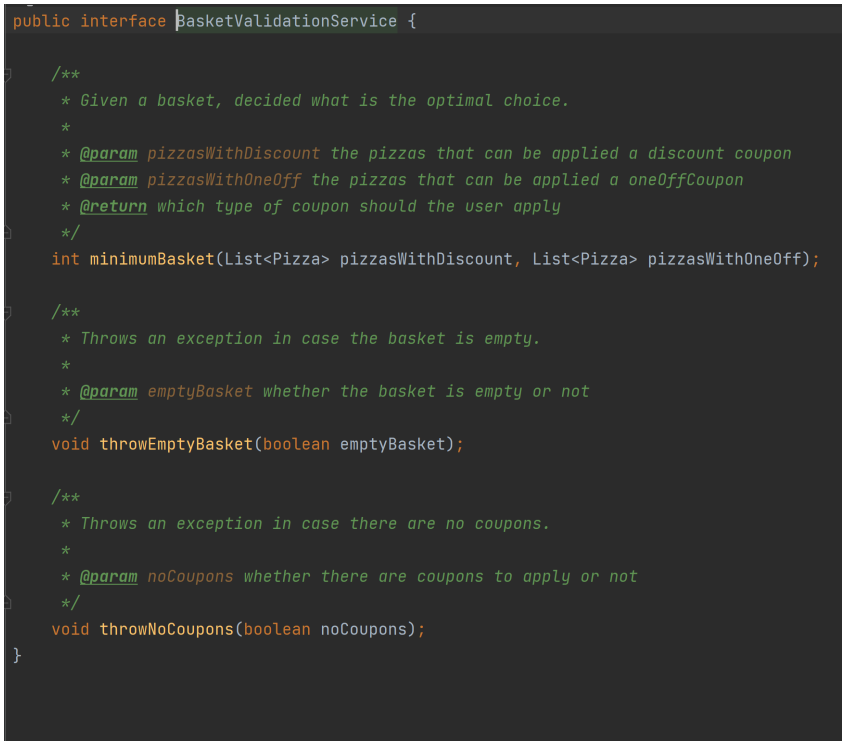
Code after refactoring:

```java
public interface CouponManagementService {

    /**
     * Adds a new coupon to the database.
     *
     * @param couponId       the id of the coupon
     * @param couponType     the type of the coupon
     * @param discount       the discount of the coupon
     * @param expirationDate the expiration date of the coupon
     */
    void addCoupon(String couponId, String couponType, double discount, Date expirationDate);

    /**
     * Removes the coupon from the database if it exists.
     *
     * @param couponId the id of the coupon
     */
    void removeCoupon(String couponId);

    /**
     * Queries the database for all coupons.
     *
     * @return all the coupons
     */
    List<Coupon> queryAllCoupons();
}
```

Figure 2.15: CouponsManagementServiceImpl code after refactoring

```java
public interface CouponOperationsService {

    /**
     * Checks if a coupon is in the repository and is available.
     *
     * @param couponId the id of the coupon
     * @return true if the coupon is in the repository
     */
    boolean isCouponAvailable(String couponId, String memberId);

    /**
     * Remembers that a coupon has been used by a customer.
     *
     * @param couponId the id of the coupon
     * @param memberId the id of the customer
     */
    void useCoupon(String couponId, String memberId);

    /**
     * Applies the coupons to the pizzas, chooses the optimal one to obtain the lowest price.
     *
     * @param pizzasAndCoupons the request containing the pizzas and the coupons
     * @return the pizzas with the applied coupons
     */
    ApplyCouponsRequestModel calculatePrice(ApplyCouponsRequestModel pizzasAndCoupons);
}
```

Figure 2.16: CouponsOperationsServiceImpl code after refactoring



```java
public interface BasketValidationService {

    /**
     * Given a basket, decided what is the optimal choice.
     *
     * @param pizzasWithDiscount the pizzas that can be applied a discount coupon
     * @param pizzasWithOneOff the pizzas that can be applied a oneOffCoupon
     * @return which type of coupon should the user apply
     */
    int minimumBasket(List<Pizza> pizzasWithDiscount, List<Pizza> pizzasWithOneOff);

    /**
     * Throws an exception in case the basket is empty.
     *
     * @param emptyBasket whether the basket is empty or not
     */
    void throwEmptyBasket(boolean emptyBasket);

    /**
     * Throws an exception in case there are no coupons.
     *
     * @param noCoupons whether there are coupons to apply or not
     */
    void throwNoCoupons(boolean noCoupons);
}
```

Figure 2.17: BasketValidationService code after refactoring

Process and result: To fix the listed problems, we used class extraction to extract the CouponsService (with interface) class based on different responsibilities. CouponsManagementService, CouponsOperationsService, and BaskedValidationService are the classes that were created as the result of the extraction. The name of the new classes indicates that they are responsible for different activities involving coupon management by regional managers/store managers, and operations involve the actions of the users and on the menu, while the latter takes care of logic that allows validation of certain coupons. This fixes the following issues: We improved NOM with a high value of 12 down to regular values of 4, 6, and 4 in `CouponsManagementServiceImpl`, `CouponsOperationsServiceImpl` and `BasketValidationService`, respectively. WMC was improved from a high value of 25 to 8, 12, 7 in `CouponsManagementServiceImpl`, `CouponsOperationsServiceImpl` and `BasketValidationService`, respectively. No changes in the tests were needed.

OrderProcessorImpl

Identified metrics: NOA

Metric values before refactoring:

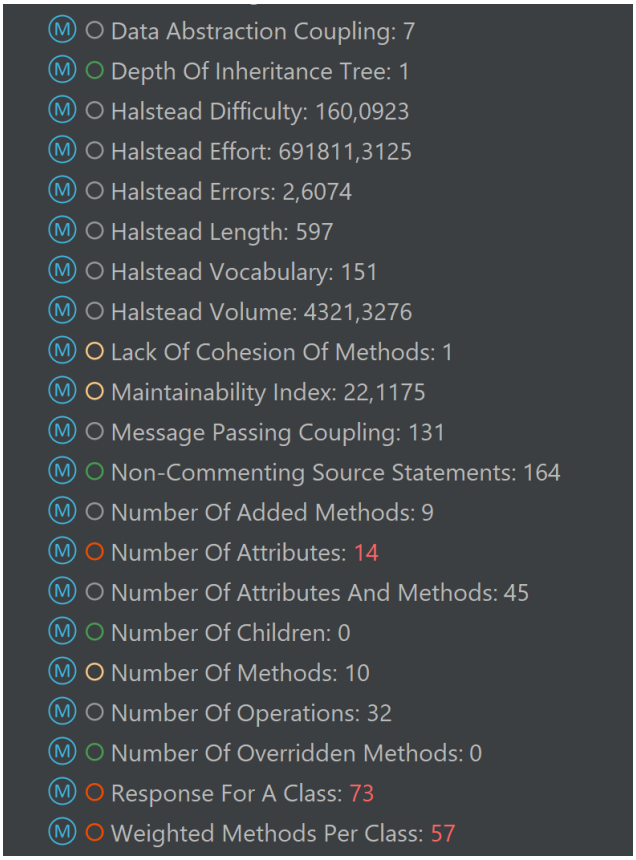


Figure 2.18: OrderProcessorImpl before refactoring

Metric values after refactoring:

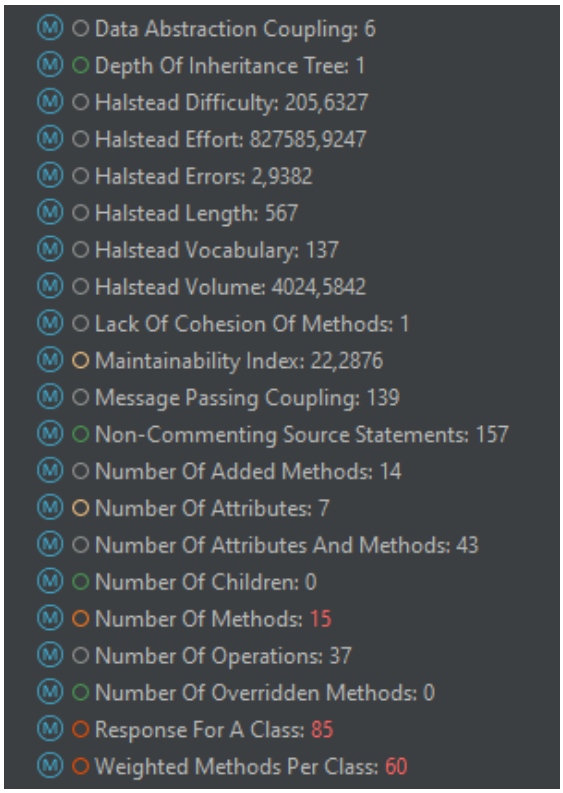


Figure 2.19: OrderProcessorImpl after refactoring

Code before refactoring:

```
@Service
public class OrderProcessorImpl implements OrderProcessor {

    private static final transient String INVALID_MEMBER_ID_MESSAGE = "Invalid member ID";
    private static final transient String INVALID_ORDER_ID_MESSAGE = "Invalid order ID";
    private static final transient String INVALID_ORDER_CONTENTS_MESSAGE = "Invalid order contents";
    private static final transient String INVALID_TIME_MESSAGE =
            "Invalid time. The time should be at least 30 minutes after the current time";
    private static final transient String INVALID_LOCATION_MESSAGE = "Invalid store location";
    private static final transient String NO_ACTIVE_ORDER_MESSAGE = "No active order with this ID";
    private static final transient String INVALID_TOKEN_MESSAGE = "Invalid token";
```

Figure 2.20: OrderProcessorImpl code before refactoring

Code after refactoring:

```
private void validateOrderAttributes(Order order) throws Exception {
    if (order.getPizzas() == null || order.getPizzas().isEmpty()) {
        throw new InvalidOrderContentsException();
    }

    if (order.getSelectedTime() == null || order.getSelectedTime().isEmpty()) {
        throw new InvalidOrderTimeException();
    }

    if (order.getStatus() == null || order.getStatus() != Status.ORDER_ONGOING) {
        throw new NoActiveOrderException();
    }

    if (order.getLocation() == null || order.getLocation().isEmpty()) {
        throw new InvalidOrderLocationException();
    }
}
```

Figure 2.21: OrderProcessorImpl code after refactoring

Process and result: To fix the number of attributes problem, we applied custom exception refactoring. We have noticed that several attributes in the class represent exception messages, so we created a new exception for every single message attribute, reducing the number of attributes from 14 (red) to 7 (yellow). This also improves the semantic correctness of the code and maintainability by providing more specific errors. Even though RFC and NOC also seem problematic at first, we decided that we shouldn't fix these metrics to preserve the structure of the design pattern used. If we split the design pattern that would worsen the maintainability even more. There were tests that needed to change because the exception message types have changed in all cases, so they needed to be replaced with the new exception types.

MenuController class refactoring

The `MenuController` class is located in the menu microservice. The output of the metrics tool before refactoring:

Class: MenuController

Met...	Description	Value
ATFD	Access To Foreign Data	7
CBO	Coupling Between Objects	8
DAC	Data Abstraction Coupling	2
DIT	Depth Of Inheritance Tree	1
CHD	Halstead Difficulty	59,0
CHEF	Halstead Effort	120261,...
CHER	Halstead Errors	0,8121
CHL	Halstead Length	300
CHVC	Halstead Vocabulary	111
CHVL	Halstead Volume	2038,3...
LCOM	Lack Of Cohesion Of Methods	1
CMI	Maintainability Index	24,0542
MPC	Message Passing Coupling	82
NCSS	Non-Commenting Source Statements	60
NOAC	Number Of Accessor Methods	0
NOA...	Number Of Added Methods	18
NOA	Number Of Attributes	2
SIZE2	Number Of Attributes And Methods	33
NOC	Number Of Children	0
NOM	Number Of Methods	19
NOO	Number Of Operations	31
NO...	Number Of Overridden Methods	0
NOPA	Number Of Public Attributes	0
RFC	Response For A Class	68
TCC	Tight Class Cohesion	0,9216
WMC	Weighted Methods Per Class	32

Figure 2.22: MenuController before refactoring

Code smell: long class. We used NOM (number of methods) and WPM(weighted methods per class) to determine that.

To fix the NOM problem, we used class extraction to extract the `MenuController` class based on different responsibilities. `MenuPizzaController`, `MenuToppingController`, and `MenuAllergyControllert` are the classes that were created as a result of the extraction. The name of the new classes indicates that they are responsible for different activities involving allergies, pizzas, and toppings on the menu. To fix the WPM issue, we first refactored each method in the `MenuController` class to directly reduce the weighted method complexity in the class, then we further reduced the complexity by extracting the `MenuController` class.
No changes in the tests were needed.
The class extraction results in the following improvement:

1. The number of methods decreased from the original MenuController class with 19 methods to 7(MenuAllergyController), 7(MenuPizzaController), and 6(MenuToppingController) methods in each class.
2. The weighted methods per class decreased from the original MenuController class with 32 to 9(MenuAllergyController), 7(MenuPizzaController) and 6(MenuToppingController) in each class.

Class: MenuAllergyController

Met...	Metrics...	Description	Value
ATFD	Lanza-...	Access To Foreign Data	1
CBO	Chidam...	Coupling Between Objects	8
DAC	Li-Henr...	Data Abstraction Coupling	4
DIT	Chidam...	Depth Of Inheritance Tree	1
CHD	Halstea...	Halstead Difficulty	26,25
CHEF	Halstea...	Halstead Effort	16290,2...
CHER	Halstea...	Halstead Errors	0,2142
CHL	Halstea...	Halstead Length	113
CHVC	Halstea...	Halstead Vocabulary	45
CHVL	Halstea...	Halstead Volume	620,5794
LCOM	Chidam...	Lack Of Cohesion Of Methods	1
CMI	Maintai...	Maintainability Index	39,1241
MPC	Li-Henr...	Message Passing Coupling	34
NCSS	Chr. Cle...	Non-Commenting Source Statements	16
NOAC	Lanza-...	Number Of Accessor Methods	0
NOA...	Lorenz-...	Number Of Added Methods	6
NOA	Lorenz-...	Number Of Attributes	4
SIZE2	Li-Henr...	Number Of Attributes And Methods	23
NOC	Chidam...	Number Of Children	0
NOM	Li-Henr...	Number Of Methods	7
NOO	Lorenz-...	Number Of Operations	19
NO...	Lorenz-...	Number Of Overridden Methods	0
NOPA	Lanza-...	Number Of Public Attributes	0
RFC	Chidam...	Response For A Class	22
TCC	Bieman...	Tight Class Cohesion	1,0
WMC	Chidam...	Weighted Methods Per Class	9
WOC	Lanza-...	Weight Of A Class	1,0

(a) MenuAllergyController after refactoring

Class: MenuPizzaController

Met...	Metrics...	Description	Value
ATFD	Lanza-...	Access To Foreign Data	3
CBO	Chidam...	Coupling Between Objects	4
DAC	Li-Henr...	Data Abstraction Coupling	1
DIT	Chidam...	Depth Of Inheritance Tree	1
CHD	Halstea...	Halstead Difficulty	17,4167
CHEF	Halstea...	Halstead Effort	4918,27...
CHER	Halstea...	Halstead Errors	0,0964
CHL	Halstea...	Halstead Length	57
CHVC	Halstea...	Halstead Vocabulary	31
CHVL	Halstea...	Halstead Volume	282,3892
LCOM	Chidam...	Lack Of Cohesion Of Methods	1
CMI	Maintai...	Maintainability Index	43,3308
MPC	Li-Henr...	Message Passing Coupling	18
NCSS	Chr. Cle...	Non-Commenting Source Statements	7
NOAC	Lanza-...	Number Of Accessor Methods	0
NOA...	Lorenz-...	Number Of Added Methods	6
NOA	Lorenz-...	Number Of Attributes	1
SIZE2	Li-Henr...	Number Of Attributes And Methods	20
NOC	Chidam...	Number Of Children	0
NOM	Li-Henr...	Number Of Methods	7
NOO	Lorenz-...	Number Of Operations	19
NO...	Lorenz-...	Number Of Overridden Methods	0
NOPA	Lanza-...	Number Of Public Attributes	0
RFC	Chidam...	Response For A Class	20
TCC	Bieman...	Tight Class Cohesion	1,0
WMC	Chidam...	Weighted Methods Per Class	7
WOC	Lanza-...	Weight Of A Class	1,0

(b) MenuPizzaController after refactoring

Class: MenuToppingController

Met...	Metrics...	Description	Value
ATFD	Lanza-...	Access To Foreign Data	0
CBO	Chidam...	Coupling Between Objects	2
DAC	Li-Henr...	Data Abstraction Coupling	1
DIT	Chidam...	Depth Of Inheritance Tree	1
CHD	Halstea...	Halstead Difficulty	10,0
CHEF	Halstea...	Halstead Effort	1800,85...
CHER	Halstea...	Halstead Errors	0,0493
CHL	Halstea...	Halstead Length	41
CHVC	Halstea...	Halstead Vocabulary	21
CHVL	Halstea...	Halstead Volume	180,085
LCOM	Chidam...	Lack Of Cohesion Of Methods	1
CMI	Maintai...	Maintainability Index	46,7187
MPC	Li-Henr...	Message Passing Coupling	11
NCSS	Chr. Cle...	Non-Commenting Source Statements	5
NOAC	Lanza-...	Number Of Accessor Methods	0
NOA...	Lorenz-...	Number Of Added Methods	5
NOA	Lorenz-...	Number Of Attributes	1
SIZE2	Li-Henr...	Number Of Attributes And Methods	19
NOC	Chidam...	Number Of Children	0
NOM	Li-Henr...	Number Of Methods	6
NOO	Lorenz-...	Number Of Operations	18
NO...	Lorenz-...	Number Of Overridden Methods	0
NOPA	Lanza-...	Number Of Public Attributes	0
RFC	Chidam...	Response For A Class	12
TCC	Bieman...	Tight Class Cohesion	1,0
WMC	Chidam...	Weighted Methods Per Class	6
WOC	Lanza-...	Weight Of A Class	1,0

(c) MenuToppingController after refactoring

Figure 2.23: MenuController after refactoring

MenuService

Identified metrics: NOM, WMC, RFC

Metric values before refactoring:

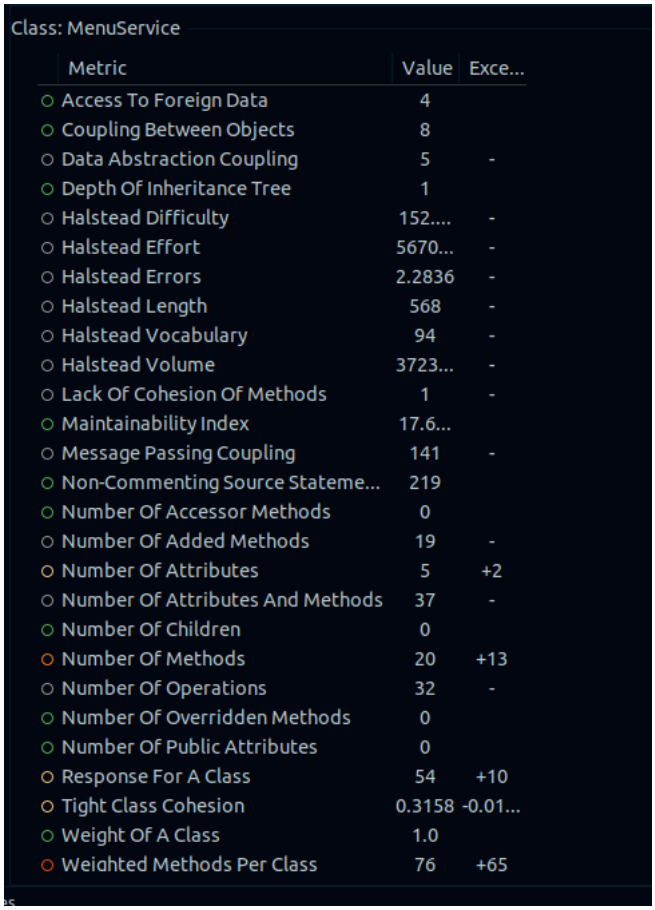


Figure 2.24: MenuService before refactoring

Metric values after refactoring:

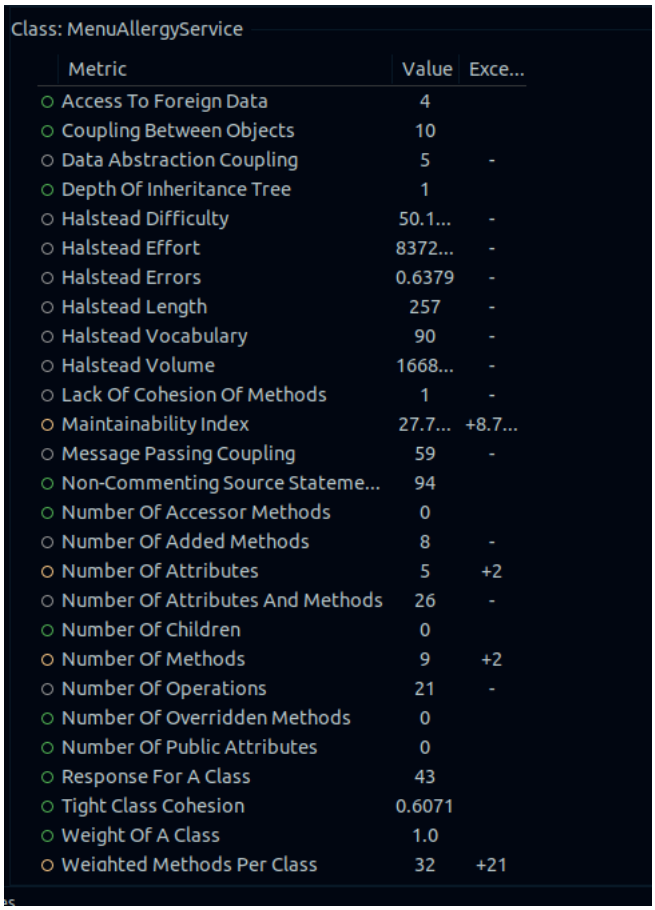
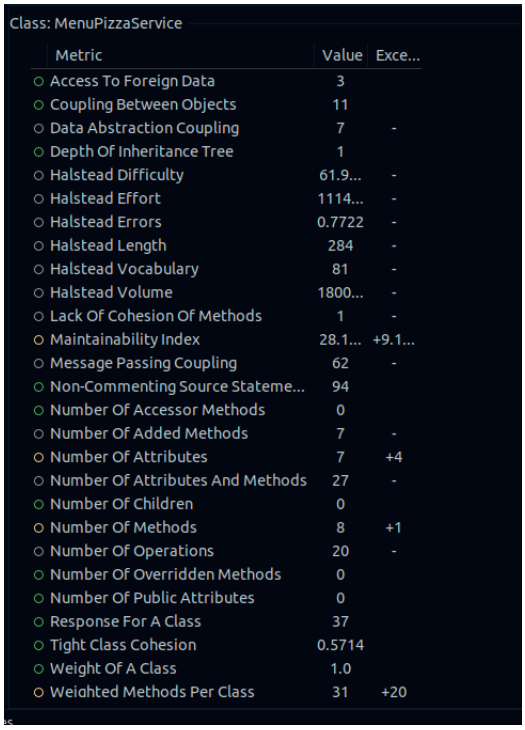
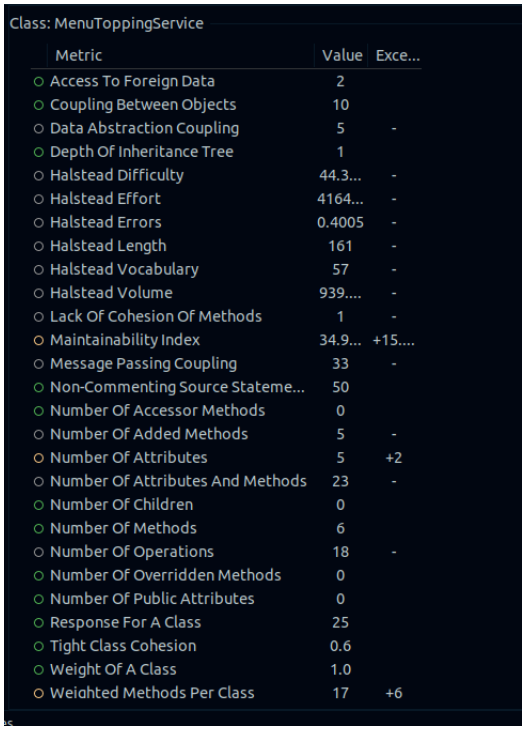


Figure 2.25: MenuAllergyService after refactoring


Class: MenuPizzaService

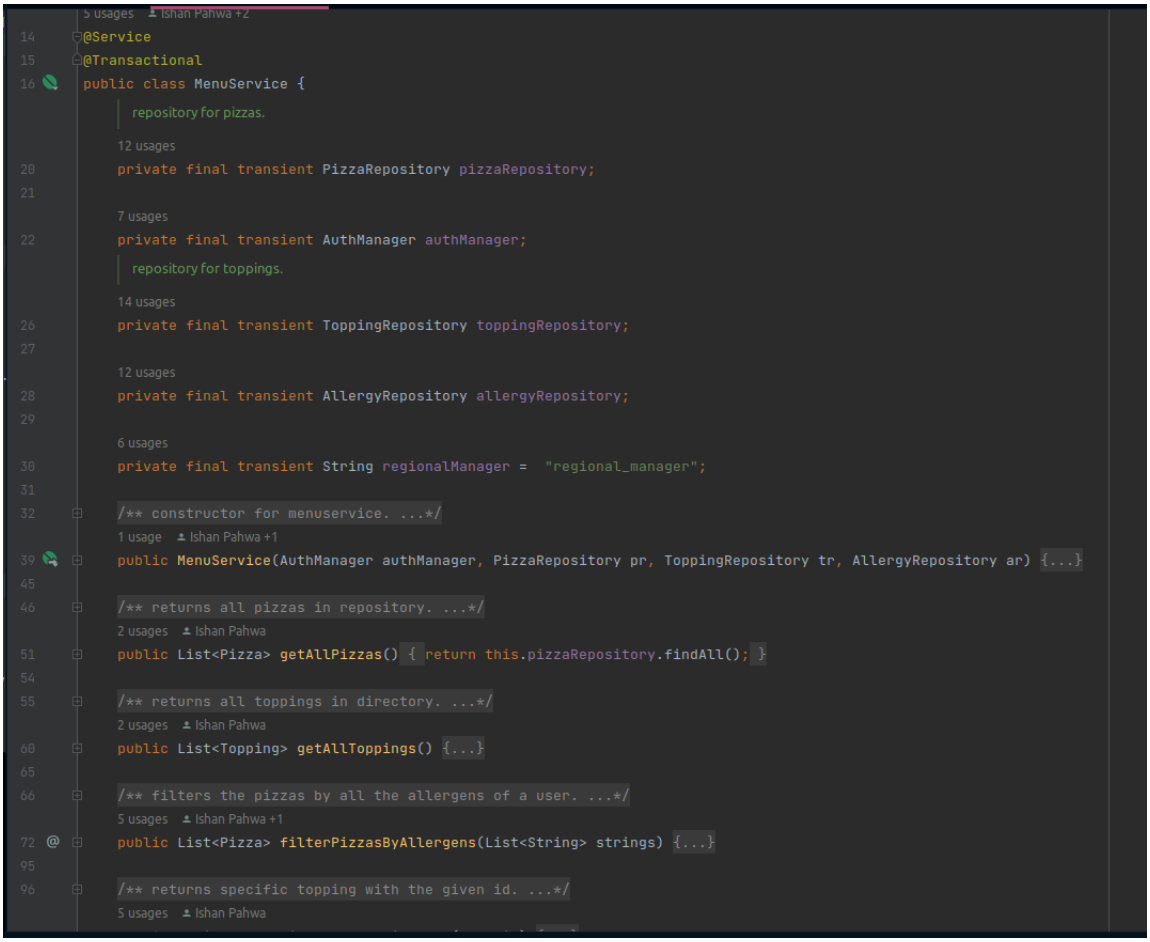
Metric	Value	Exce...
Access To Foreign Data	3	
Coupling Between Objects	11	
Data Abstraction Coupling	7	-
Depth Of Inheritance Tree	1	
Halstead Difficulty	61.9...	-
Halstead Effort	1114...	-
Halstead Errors	0.7722	-
Halstead Length	284	-
Halstead Vocabulary	81	-
Halstead Volume	1800...	-
Lack Of Cohesion Of Methods	1	-
Maintainability Index	28.1...	+9.1...
Message Passing Coupling	62	-
Non-Commenting Source Stateme...	94	
Number Of Accessor Methods	0	
Number Of Added Methods	7	-
Number Of Attributes	7	+4
Number Of Attributes And Methods	27	-
Number Of Children	0	
Number Of Methods	8	+1
Number Of Operations	20	-
Number Of Overridden Methods	0	
Number Of Public Attributes	0	
Response For A Class	37	
Tight Class Cohesion	0.5714	
Weight Of A Class	1.0	
Weighted Methods Per Class	31	+20

Figure 2.26: MenuPizzaService after refactoring


Class: MenuToppingService

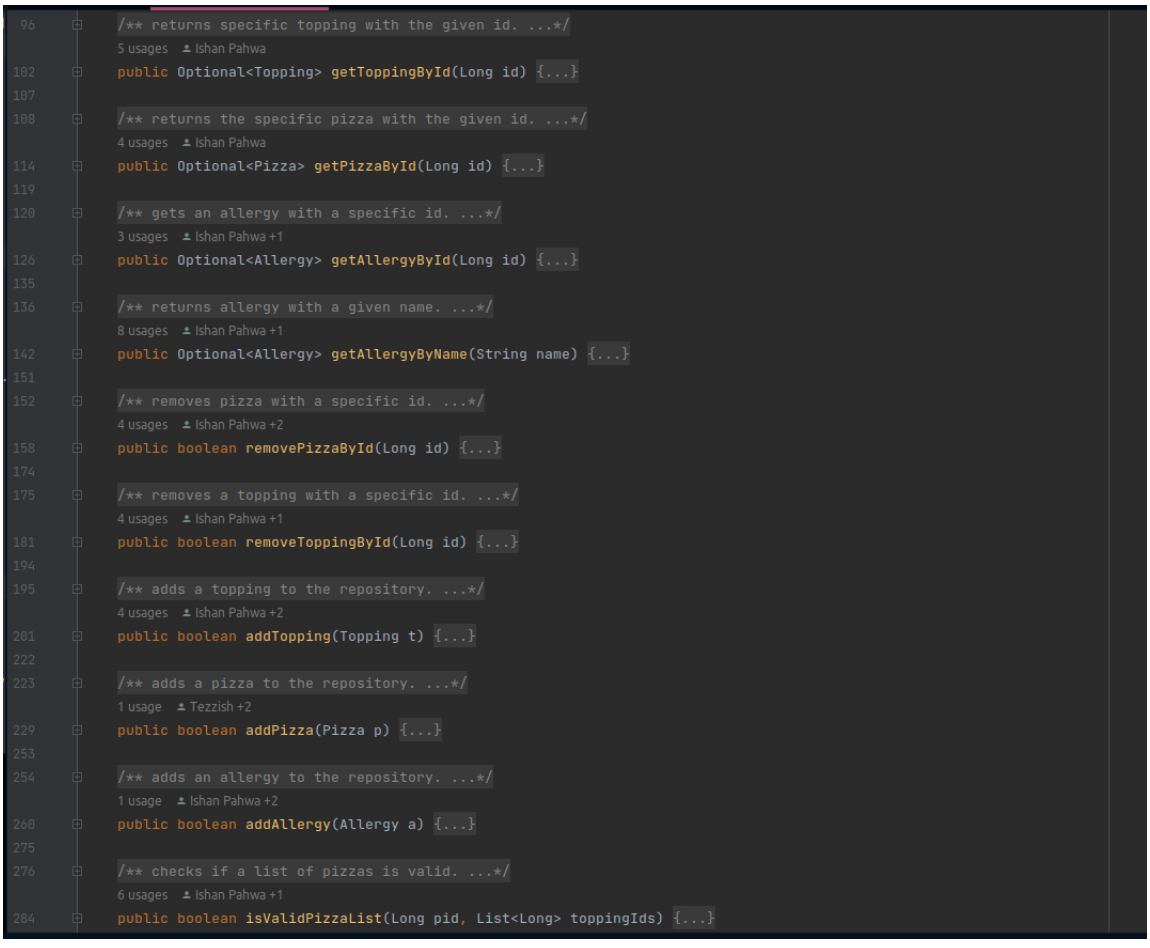
Metric	Value	Exce...
Access To Foreign Data	2	
Coupling Between Objects	10	
Data Abstraction Coupling	5	-
Depth Of Inheritance Tree	1	
Halstead Difficulty	44.3...	-
Halstead Effort	4164...	-
Halstead Errors	0.4005	-
Halstead Length	161	-
Halstead Vocabulary	57	-
Halstead Volume	939....	-
Lack Of Cohesion Of Methods	1	-
Maintainability Index	34.9...	+15....
Message Passing Coupling	33	-
Non-Commenting Source Stateme...	50	
Number Of Accessor Methods	0	
Number Of Added Methods	5	-
Number Of Attributes	5	+2
Number Of Attributes And Methods	23	-
Number Of Children	0	
Number Of Methods	6	
Number Of Operations	18	-
Number Of Overridden Methods	0	
Number Of Public Attributes	0	
Response For A Class	25	
Tight Class Cohesion	0.6	
Weight Of A Class	1.0	
Weighted Methods Per Class	17	+6

Figure 2.27: MenuToppingService after refactoring

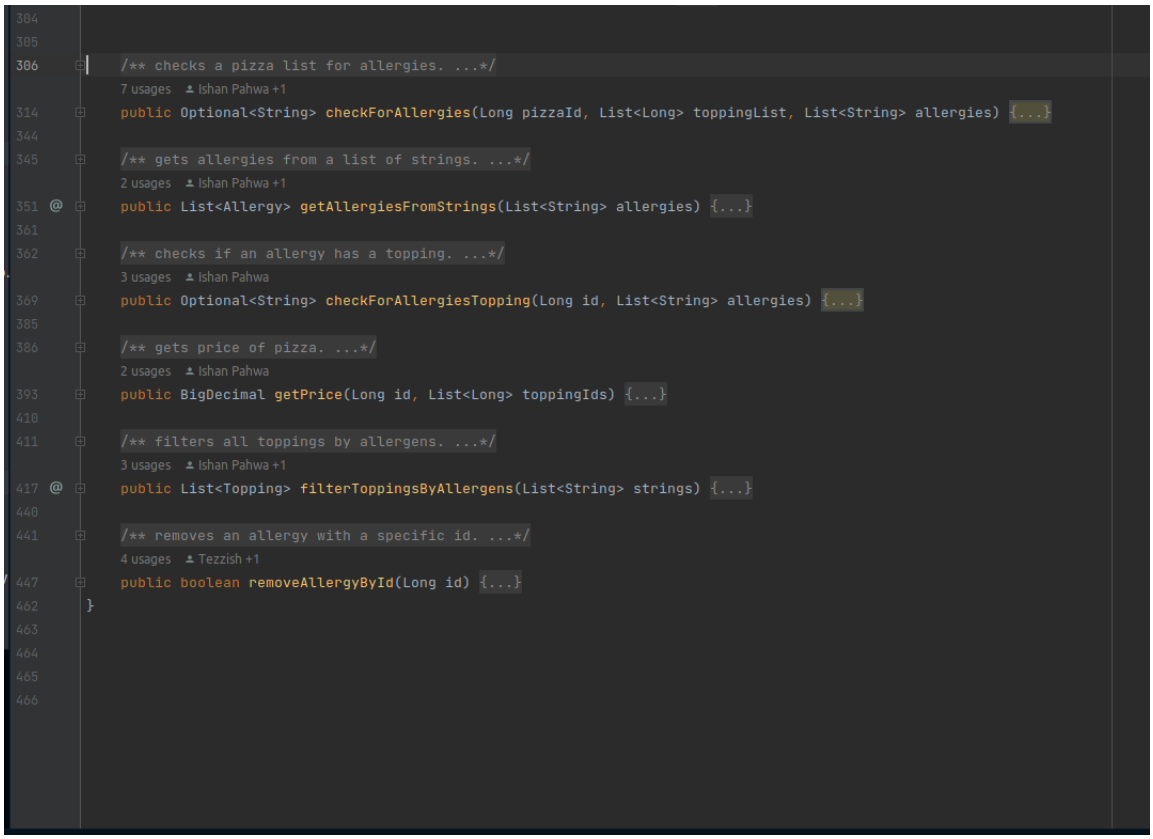


```
5 usages  Ishan Pahwa +2
@Service
@Transactional
public class MenuService {
    repository for pizzas.
    12 usages
    private final transient PizzaRepository pizzaRepository;

    7 usages
    private final transient AuthManager authManager;
    repository for toppings.
    14 usages
    private final transient ToppingRepository toppingRepository;

    12 usages
    private final transient AllergyRepository allergyRepository;

    6 usages
    private final transient String regionalManager =  "regional_manager";

    /** constructor for menuservice. ...*/
    1 usage  Ishan Pahwa +1
    public MenuService(AuthManager authManager, PizzaRepository pr, ToppingRepository tr, AllergyRepository ar) {...}

    /** returns all pizzas in repository. ...*/
    2 usages  Ishan Pahwa
    public List<Pizza> getAllPizzas() { return this.pizzaRepository.findAll(); }

    /** returns all toppings in directory. ...*/
    2 usages  Ishan Pahwa
    public List<Topping> getAllToppings() {...}

    /** filters the pizzas by all the allergens of a user. ...*/
    5 usages  Ishan Pahwa +1
    public List<Pizza> filterPizzasByAllergens(List<String> strings) {...}

    /** returns specific topping with the given id. ...*/
    5 usages  Ishan Pahwa
```

Figure 2.28: MenuService before refactoring

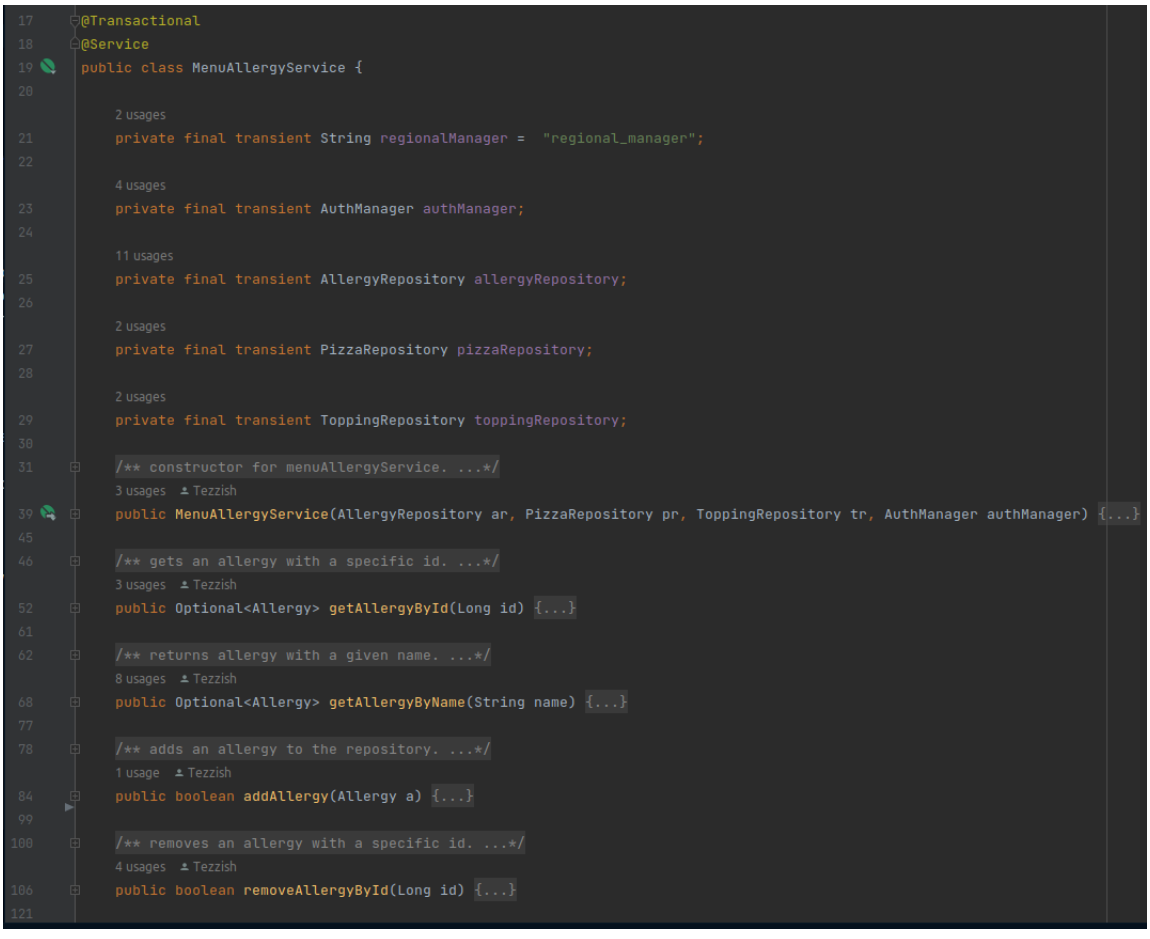


```
    /** returns specific topping with the given id. ...*/
    5 usages  Ishan Pahwa
    public Optional<Topping> getToppingById(Long id) {...}

    /** returns the specific pizza with the given id. ...*/
    4 usages  Ishan Pahwa
    public Optional<Pizza> getPizzaById(Long id) {...}

    /** gets an allergy with a specific id. ...*/
    3 usages  Ishan Pahwa +1
    public Optional<Allergy> getAllergyById(Long id) {...}

    /** returns allergy with a given name. ...*/
    8 usages  Ishan Pahwa +1
    public Optional<Allergy> getAllergyByName(String name) {...}

    /** removes pizza with a specific id. ...*/
    4 usages  Ishan Pahwa +2
    public boolean removePizzaById(Long id) {...}

    /** removes a topping with a specific id. ...*/
    4 usages  Ishan Pahwa +1
    public boolean removeToppingById(Long id) {...}

    /** adds a topping to the repository. ...*/
    4 usages  Ishan Pahwa +2
    public boolean addTopping(Topping t) {...}

    /** adds a pizza to the repository. ...*/
    1 usage  Tezzish +2
    public boolean addPizza(Pizza p) {...}

    /** adds an allergy to the repository. ...*/
    1 usage  Ishan Pahwa +2
    public boolean addAllergy(Allergy a) {...}

    /** checks if a list of pizzas is valid. ...*/
    6 usages  Ishan Pahwa +1
    public boolean isValidPizzaList(Long pid, List<Long> toppingIds) {...}
```



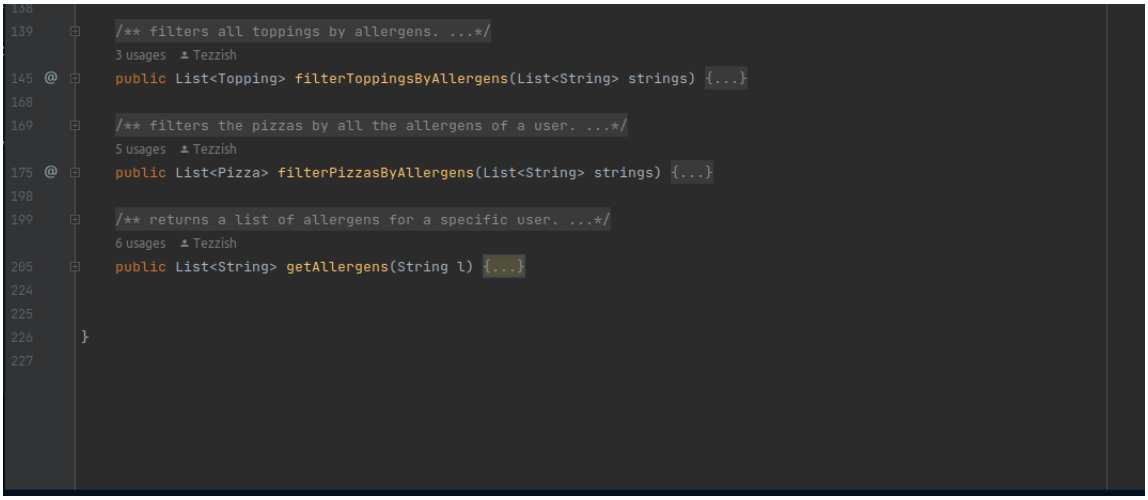
```
304
305
306     /** checks a pizza list for allergies. ...*/
        7 usages  Ishan Pahwa +1
314     public Optional<String> checkForAllergies(Long pizzaId, List<Long> toppingList, List<String> allergies) {...}
344
345     /** gets allergies from a list of strings. ...*/
        2 usages  Ishan Pahwa +1
351 @   public List<Allergy> getAllergiesFromStrings(List<String> allergies) {...}
361
362     /** checks if an allergy has a topping. ...*/
        3 usages  Ishan Pahwa
369     public Optional<String> checkForAllergiesTopping(Long id, List<String> allergies) {...}
385
386     /** gets price of pizza. ...*/
        2 usages  Ishan Pahwa
393     public BigDecimal getPrice(Long id, List<Long> toppingIds) {...}
410
411     /** filters all toppings by allergens. ...*/
        3 usages  Ishan Pahwa +1
417 @   public List<Topping> filterToppingsByAllergens(List<String> strings) {...}
440
441     /** removes an allergy with a specific id. ...*/
        4 usages  Tezzish +1
447     public boolean removeAllergyById(Long id) {...}
462 }
463
464
465
466
```

Figure 2.30: MenuService (2) before refactoring



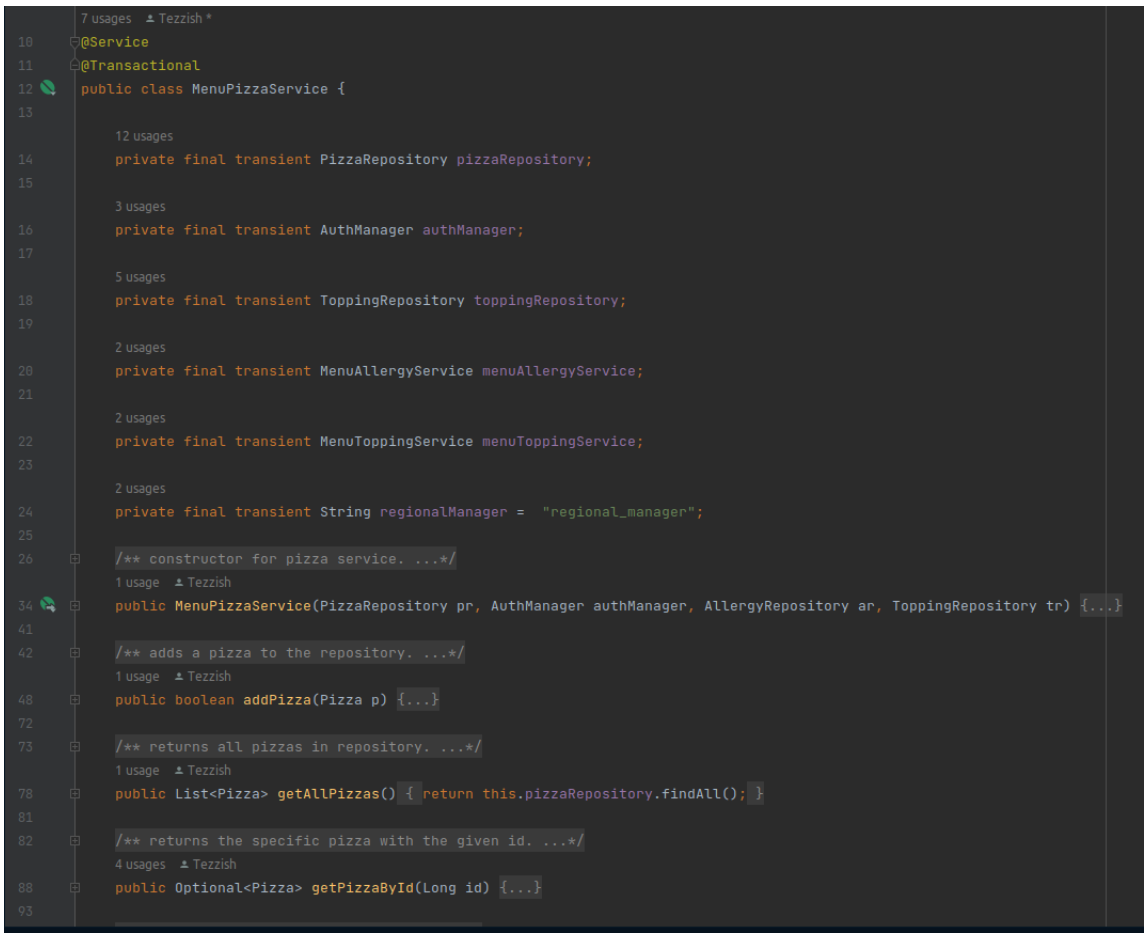
```
17  @Transactional
18  @Service
19  public class MenuAllergyService {
20
        2 usages
21      private final transient String regionalManager =  "regional_manager";
22
        4 usages
23      private final transient AuthManager authManager;
24
        11 usages
25      private final transient AllergyRepository allergyRepository;
26
        2 usages
27      private final transient PizzaRepository pizzaRepository;
28
        2 usages
29      private final transient ToppingRepository toppingRepository;
30
31      /** constructor for menuAllergyService. ...*/
        3 usages  Tezzish
39      public MenuAllergyService(AllergyRepository ar, PizzaRepository pr, ToppingRepository tr, AuthManager authManager) {...}
45
46      /** gets an allergy with a specific id. ...*/
        3 usages  Tezzish
52      public Optional<Allergy> getAllergyById(Long id) {...}
61
62      /** returns allergy with a given name. ...*/
        8 usages  Tezzish
68      public Optional<Allergy> getAllergyByName(String name) {...}
77
78      /** adds an allergy to the repository. ...*/
        1 usage  Tezzish
84      public boolean addAllergy(Allergy a) {...}
99
100     /** removes an allergy with a specific id. ...*/
        4 usages  Tezzish
106     public boolean removeAllergyById(Long id) {...}
121
```

Figure 2.31: MenuAllergyService after refactoring



```
    /** filters all toppings by allergens. ...*/
    3 usages  Tezzish
    public List<Topping> filterToppingsByAllergens(List<String> strings) {...}

    /** filters the pizzas by all the allergens of a user. ...*/
    5 usages  Tezzish
    public List<Pizza> filterPizzasByAllergens(List<String> strings) {...}

    /** returns a list of allergens for a specific user. ...*/
    6 usages  Tezzish
    public List<String> getAllergens(String l) {...}


}
```

Figure 2.32: MenuAllergyService (2) after refactoring



```
7 usages  Tezzish *
@Service
@Transactional
public class MenuPizzaService {

    12 usages
    private final transient PizzaRepository pizzaRepository;

    3 usages
    private final transient AuthManager authManager;

    5 usages
    private final transient ToppingRepository toppingRepository;

    2 usages
    private final transient MenuAllergyService menuAllergyService;

    2 usages
    private final transient MenuToppingService menuToppingService;

    2 usages
    private final transient String regionalManager =  "regional_manager";

    /** constructor for pizza service. ...*/
    1 usage  Tezzish
    public MenuPizzaService(PizzaRepository pr, AuthManager authManager, AllergyRepository ar, ToppingRepository tr) {...}

    /** adds a pizza to the repository. ...*/
    1 usage  Tezzish
    public boolean addPizza(Pizza p) {...}

    /** returns all pizzas in repository. ...*/
    1 usage  Tezzish
    public List<Pizza> getAllPizzas() { return this.pizzaRepository.findAll(); }

    /** returns the specific pizza with the given id. ...*/
    4 usages  Tezzish
    public Optional<Pizza> getPizzaById(Long id) {...}
```

Figure 2.33: MenuPizzaService after refactoring

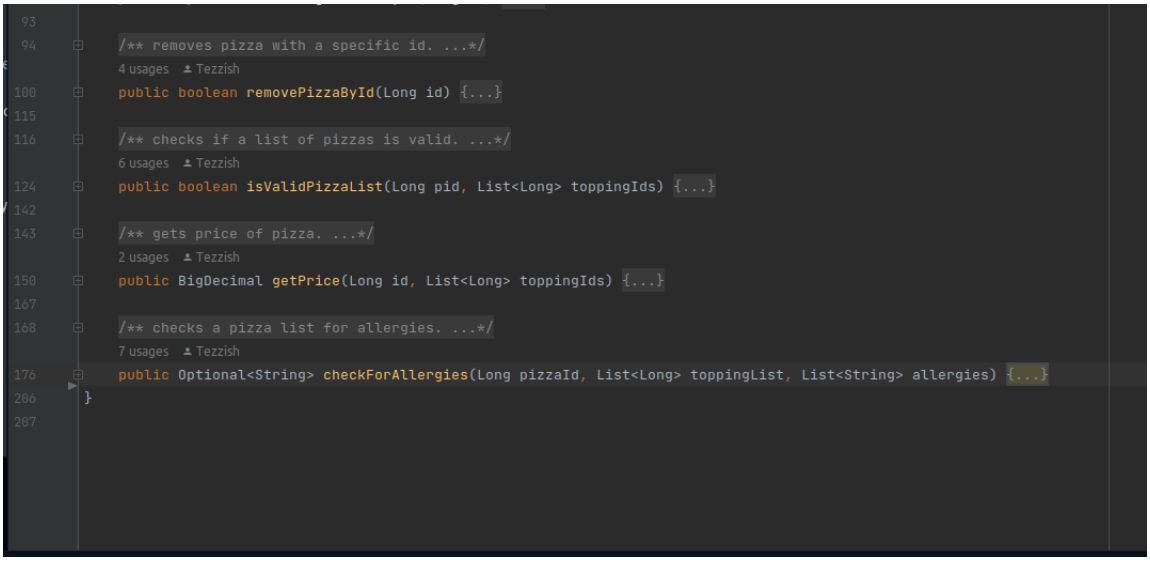


Figure 2.34: MenuPizzaService (2) after refactoring

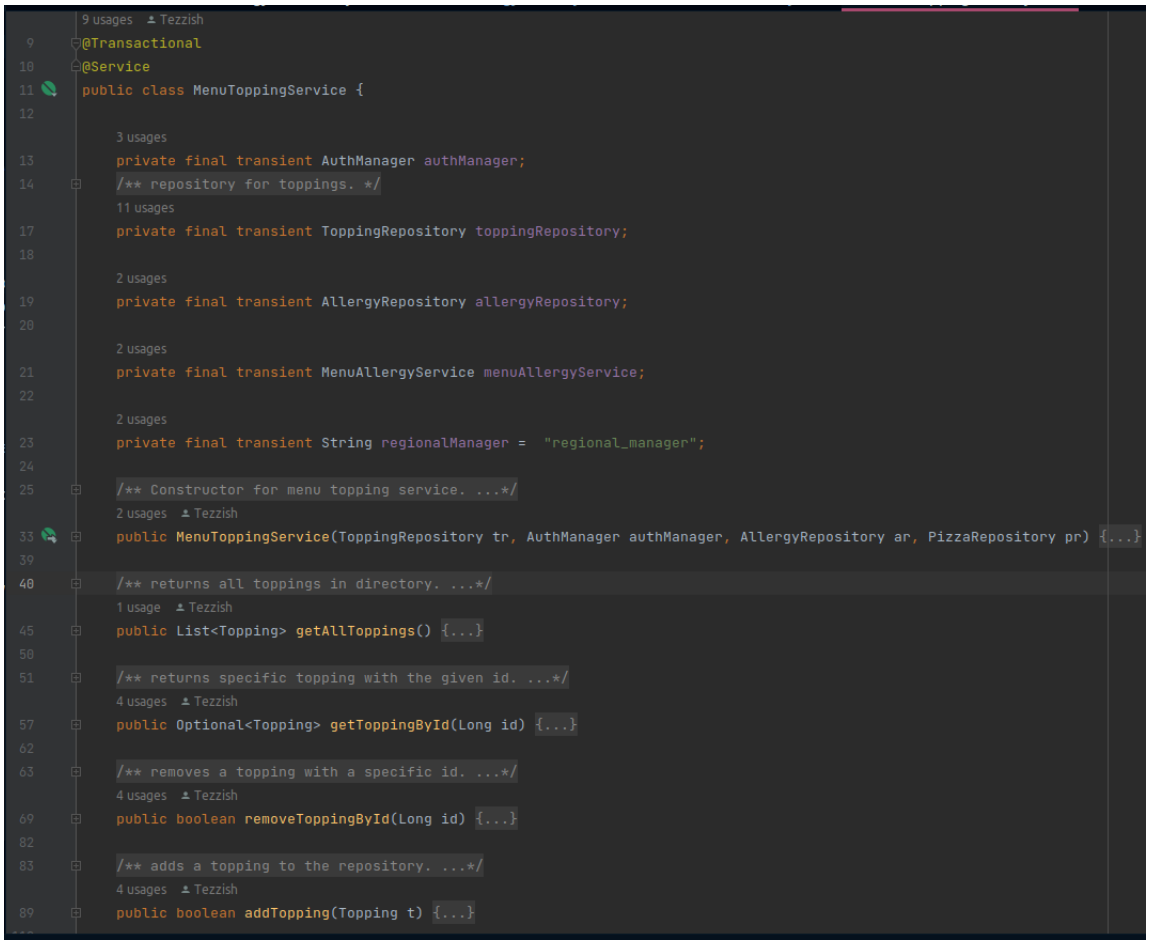
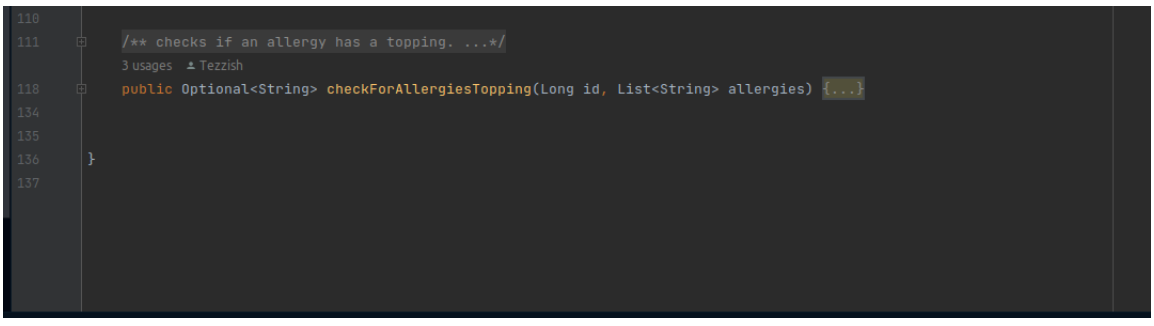


Figure 2.35: MenuToppingService after refactoring



Process and result:

To fix the problem, we used class extraction to split the class between three service classes, `MenuPizzaService`, `MenuToppingService`, `MenuAllergyService`. We chose these three classes since all of the methods in the menu service correspond to either allergies, toppings, or pizzas, and there is an even distribution of methods across the three. Splitting the methods up into three classes also splits up the cyclomatic complexity leading to lower weighted methods per class for each of the new classes than the original class. The number of methods went down from a single class with 20 methods to three classes with 8, 7, and 5 methods (not including constructors). We feel that this is far more legible and has a better structure than the original class. Furthermore, the response for a class metric decreased from orange to green in all three of our classes (from 71 to 43, 37, and 25) since the number of methods decreased which is a factor it relies on. We also changed the names of the objects in the tests to the new classes.

Method level refactoring

CouponsServiceImpl.calculatePrice()

Identified metrics: LOC, MCC

Code before refactoring:

```
public ApplyCouponsRequestModel calculatePrice(ApplyCouponsRequestModel pizzasAndCoupons) {
    List<Pizza> pizzas = pizzasAndCoupons.getPizzas();
    if (!pizzas.isEmpty()) {
        List<Coupon> couponsList = couponRepository.findAllById(pizzasAndCoupons.getCoupons());
        if (couponsList.isEmpty()) {
            throw new IllegalArgumentException("No coupons found");
        }

        Coupon discountCoupon = couponsList.stream() Stream<Coupon>
                .filter(coupon -> coupon.getType() == CouponType.DISCOUNT)
                .max(Comparator.comparing(Coupon::getDiscount)) Optional<Coupon>
                .orElse( other: null);
        Coupon oneOffCoupon = couponsList.stream() Stream<Coupon>
                .filter(coupon -> coupon.getType() == CouponType.ONE_OFF)
                .findFirst() Optional<Coupon>
                .orElse( other: null);

        BigDecimal totalPrice = pizzas.stream() Stream<Pizza>
                .map(Pizza::getPrice) Stream<BigDecimal>
                .reduce(BigDecimal.ZERO, BigDecimal::add);
        BigDecimal bigPrice = pizzas.stream() Stream<Pizza>
                .map(Pizza::getPrice) Stream<BigDecimal>
                .max(BigDecimal::compareTo) Optional<BigDecimal>
                .orElse(BigDecimal.ZERO);
        // Find the index of the pizza with the highest price
        int index = pizzas.indexOf(pizzas.stream() Stream<Pizza>
                .filter(pizza -> pizza.getPrice().equals(bigPrice))
                .findFirst() Optional<Pizza>
                .orElse( other: null));

        if (discountCoupon != null && oneOffCoupon != null) {
            if (totalPrice.multiply(BigDecimal.valueOf(1 - discountCoupon.getDiscount()))
                    .compareTo(totalPrice.subtract(bigPrice)) < 0) {
                pizzasAndCoupons.setCoupons(List.of(discountCoupon.getCode()));
                pizzasAndCoupons.getPizzas()
                        .forEach(
                                pizza -> pizza.setPrice(
                                        pizza.getPrice().multiply(BigDecimal.valueOf(1 - discountCoupon.getDiscount()))
                                ));
            } else {
                pizzasAndCoupons.setCoupons(List.of(oneOffCoupon.getCode()));
                pizzas.get(index).setPrice(BigDecimal.ZERO);
            }
        } else if (discountCoupon != null) {
            pizzasAndCoupons.setCoupons(List.of(discountCoupon.getCode()));
            pizzasAndCoupons.getPizzas()
                    .forEach(
                            pizza -> pizza.setPrice(
                                    pizza.getPrice().multiply(BigDecimal.valueOf(1 - discountCoupon.getDiscount()))
                            ));
        } else if (oneOffCoupon != null) {
            pizzasAndCoupons.setCoupons(List.of(oneOffCoupon.getCode()));
            pizzas.get(index).setPrice(BigDecimal.ZERO);
        }
        return pizzasAndCoupons;
    }

    throw new IllegalArgumentException("The basket is empty");
}
```

Figure 2.37: calculatePrice code before refactoring

Metric values before refactoring:

Figure 2.38: calculatePrice before refactoring

Code after refactoring:

```
public ApplyCouponsRequestModel calculatePrice(ApplyCouponsRequestModel pizzasAndCoupons) {
    List<Pizza> pizzas = pizzasAndCoupons.getPizzas();
    List<Coupon> couponsList = couponRepository.findAllById(pizzasAndCoupons.getCoupons());
    throwEmptyBasket(pizzas.isEmpty());
    throwNoCoupons(couponsList.isEmpty());

    Coupon discountCoupon = couponsList.stream().filter(coupon -> coupon.getType() == CouponType.DISCOUNT)
            .max(Comparator.comparing(Coupon::getDiscount)).orElse(new Coupon());
    Coupon oneOffCoupon = couponsList.stream().filter(coupon -> coupon.getType() == CouponType.ONE_OFF)
            .findFirst().orElse(new Coupon());

    Pizza pizza = pizzas.stream().max(Comparator.comparing(Pizza::getPrice)).get();
    int maxIndex = pizzas.indexOf(pizza);

    List<Coupon> coupons = List.of(discountCoupon, oneOffCoupon);
    List<List<Pizza>> pizzasList = List.of(calculateDiscountBasket(pizzas, discountCoupon),
            calculateOneOffBasket(pizzas, oneOffCoupon, maxIndex));
    int bestIndex = minimumBasket(pizzasList.get(0), pizzasList.get(1));

    return new ApplyCouponsRequestModel(pizzasList.get(bestIndex), List.of(coupons.get(bestIndex).getCode()));
}
```

Figure 2.39: calculatePrice Code after refactoring

Metric values after refactoring:

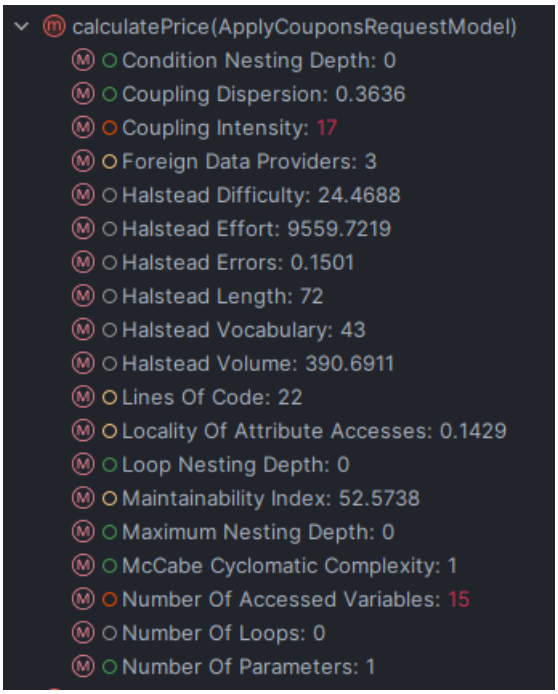


Figure 2.40: calculatePrice after refactoring

Process and result: The algorithm through which this method arrives at the results was significantly reworked and optimized which initially led to a significant reduction in cyclomatic complexity and effective length of code. Method extraction was also performed, by creating private helper methods, to further reduce the cyclomatic complexity of the method. No changes in the tests were needed.

MenuAllergyService.filterPizzasByAllergens()

Identified metrics: CC, LOC, Loop Nesting Depth

Metric values before refactoring:

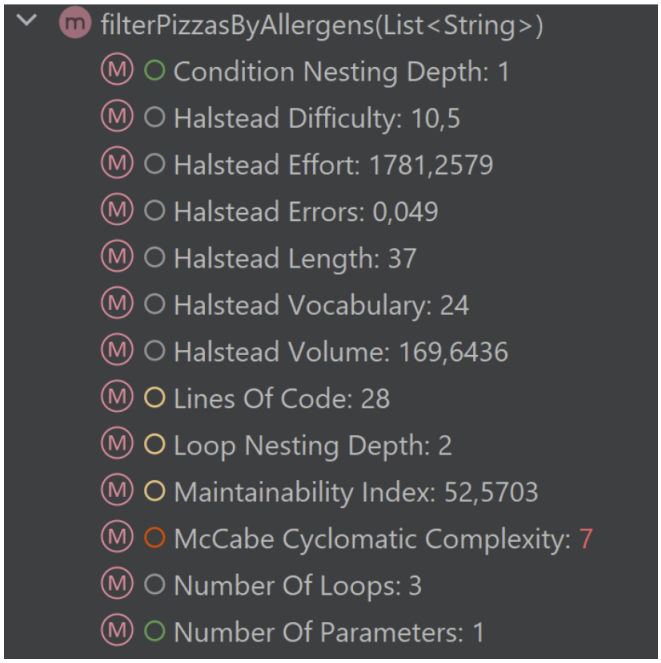


Figure 2.41: filterPizzaByAllergens before refactoring

Metric values after refactoring:

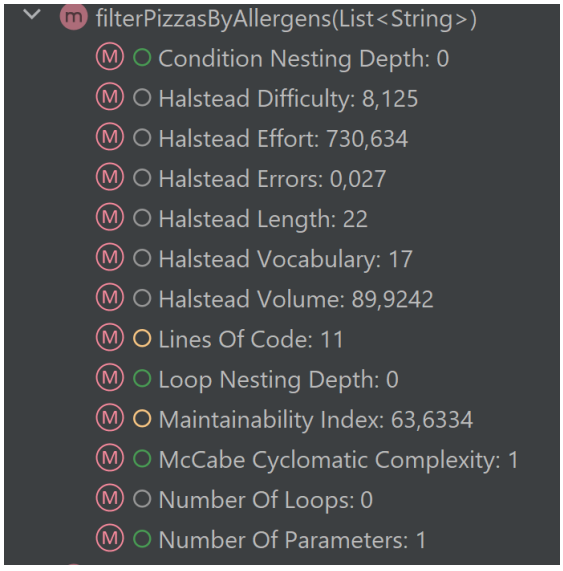


Figure 2.42: filterPizzaByAllergens after refactoring

Code before refactoring:

```
public List<Pizza> filterPizzasByAllergens(List<String> strings) {
    ArrayList<Allergy> allergyList = new ArrayList<>();
    for (String s : strings) {
        if (getAllergyByName(s).isPresent()) {
            allergyList.add(getAllergyByName(s).get());
        }
    }

    ArrayList<Pizza> ret = new ArrayList<>();
    for (Pizza p : getAllPizzas()) {
        boolean add = true;
        for (Allergy a : allergyList) {
            if (p.containsAllergen(a).isPresent()) {
                add = false;
            }
        }
        if (add) {
            ret.add(p);
        }
    }
    this.allergyRepository.flush();
    return ret;
}
```

Figure 2.43: filterPizza code before refactoring

Code after refactoring:

```
public List<Pizza> filterPizzasByAllergens(List<String> strings) {
    return getAllPizzas().stream().filter(pizza -> strings.stream()
                    .filter(x -> getAllergyByName(x).isPresent()).map(x -> getAllergyByName(x).get())
                    .noneMatch(allergy -> pizza.containsAllergen(allergy).isPresent())).collect(Collectors.toList());
}
```

Figure 2.44: filterPizza code after refactoring

Process and result: To fix this problem we used “logic simplification” to replace unnecessary code fragments and remove all unneeded loops and branches. The code was improved by using functional programming. Functional programming uses pure functions without side effects, so this can be used to reduce metrics like Cyclomatic Complexity and Loop Nesting Depth. No test needed updating. No changes in the tests were needed.

OrderProcessorImpl.placeOrder()

Identified metrics: LOC

Metric values before refactoring:

Method: placeOrder(String, Long)

Metric	Metrics Set	Description	Value	Regular Range
CND		Condition Nesting Depth	2	[0..2)
LND		Loop Nesting Depth	1	[0..2)
CC		McCabe Cyclomatic Complexity	16	[0..3)
NOL		Number Of Loops	1	
LOC		Lines Of Code	47	[0..11)
NOPM		Number Of Parameters	2	[0..3)
HVL	Halstead M...	Halstead Volume	926.4784	
HD	Halstead M...	Halstead Difficulty	67.0	
HL	Halstead M...	Halstead Length	155	
HEF	Halstead M...	Halstead Effort	62074.052	
HVC	Halstead M...	Halstead Vocabulary	63	
HER	Halstead M...	Halstead Errors	0.5226	
MMI	Maintainabi...	Maintainability Index	42.3797	[0.0..19.0]

Figure 2.45: placeOrder method metrics before refactoring

Metric values after refactoring:

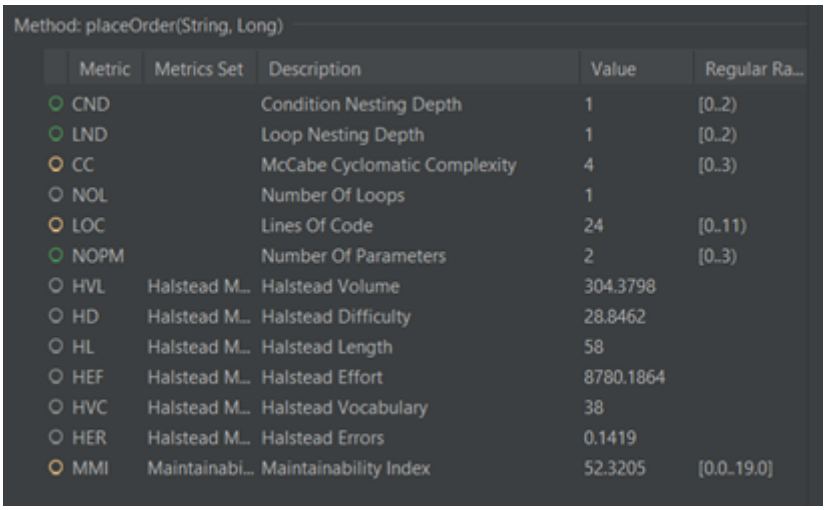


Figure 2.46: placeOrder method metrics after refactoring

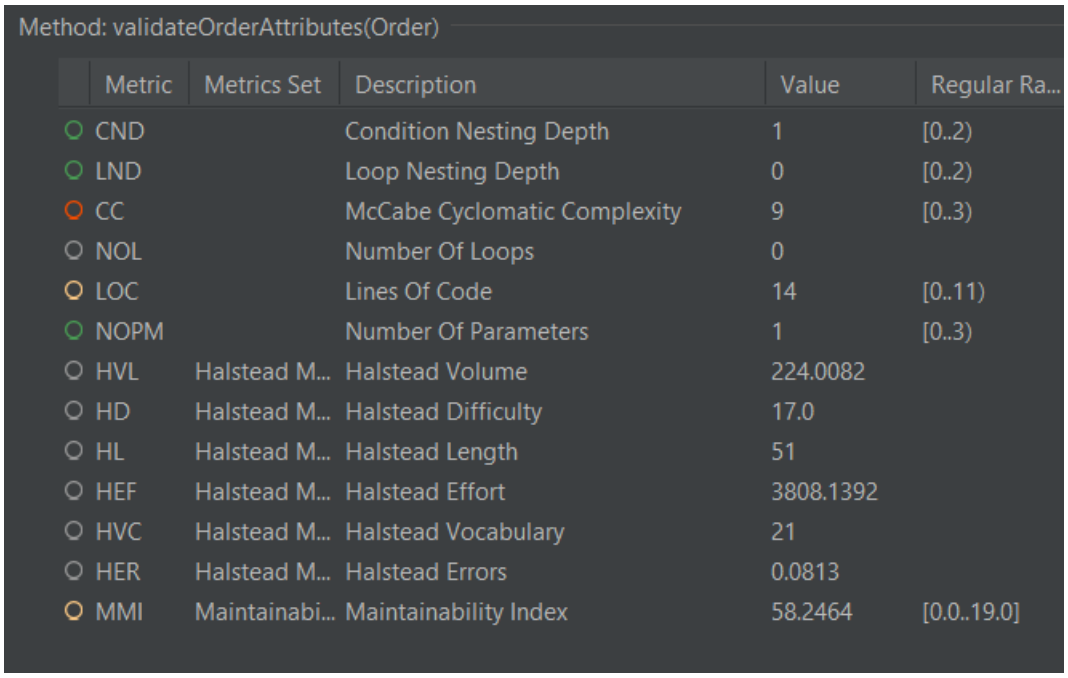
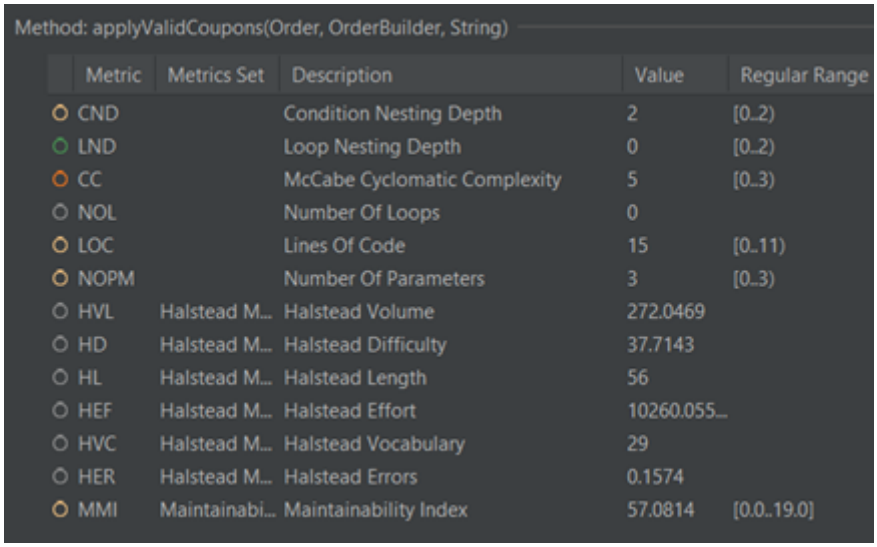


Figure 2.47: validateOrderAttributes method metrics



Metric	Metrics Set	Description	Value	Regular Range
CND		Condition Nesting Depth	2	[0..2)
LND		Loop Nesting Depth	0	[0..2)
CC		McCabe Cyclomatic Complexity	5	[0..3)
NOL		Number Of Loops	0	
LOC		Lines Of Code	15	[0..11)
NOPM		Number Of Parameters	3	[0..3)
HVL	Halstead M...	Halstead Volume	272.0469	
HD	Halstead M...	Halstead Difficulty	37.7143	
HL	Halstead M...	Halstead Length	56	
HEF	Halstead M...	Halstead Effort	10260.055...	
HVC	Halstead M...	Halstead Vocabulary	29	
HER	Halstead M...	Halstead Errors	0.1574	
MMI	Maintainabi...	Maintainability Index	57.0814	[0.0..19.0]

Figure 2.48: applyValidCoupons method metrics

Code before refactoring:

```java
@Override
public Order placeOrder(String token, Long orderId) throws IllegalArgumentException {
    if (token.isEmpty()) {
        throw new IllegalArgumentException(INVALID_TOKEN_MESSAGE);
    }

    if (orderId == null) {
        throw new IllegalArgumentException(INVALID_ORDER_ID_MESSAGE);
    }

    Order order = orderRepository.getOne(orderId);

    if (order.getPizzas() == null || order.getPizzas().isEmpty()) {
        throw new IllegalArgumentException(INVALID_ORDER_CONTENTS_MESSAGE);
    }

    if (order.getSelectedTime() == null || order.getSelectedTime().isEmpty()) {
        throw new UnsupportedOperationException("No order time is selected");
    }

    if (order.getStatus() == null || order.getStatus() != Status.ORDER_ONGOING) {
        throw new IllegalArgumentException(NO_ACTIVE_ORDER_MESSAGE);
    }

    if (order.getLocation() == null || order.getLocation().isEmpty()) {
        throw new UnsupportedOperationException("No store location is selected");
    }

    for (Pizza pizza : order.getPizzas()) {
        pizza.setPrice(menuCommunication.getPizzaPriceFromMenu(pizza, token));
    }

    OrderBuilder orderBuilder = Builder.toBuilder(order);

    if (order.getCoupons() != null && !order.getCoupons().isEmpty()) {
        ApplyCouponsToOrderModel applyCouponsToResponse = couponCommunication.applyCouponsToOrder(order.getPizzas(),
                new ArrayList<>(order.getCoupons()), token);
        if (applyCouponsToResponse.getCoupons() == null || applyCouponsToResponse.getCoupons().isEmpty()) {
            orderBuilder.setAppliedCoupon(null);
            order.setAppliedCoupon(null);
        } else {
            orderBuilder.setAppliedCoupon(applyCouponsToResponse.getCoupons().get(0));
            order.setAppliedCoupon(applyCouponsToResponse.getCoupons().get(0));
        }
        orderBuilder.setPizzas(applyCouponsToResponse.getPizzas());
        order.setPizzas(applyCouponsToResponse.getPizzas());
    }

    orderBuilder.setPrice(order.calculateTotalPrice());
    orderBuilder.setOrderStatus(Status.ORDER_PLACED);
    scheduleOrderCompletion(orderId);

    Order newOrder = orderBuilder.build();
    orderRepository.save(newOrder);

    // notify the store
    storeCommunication.sendEmailToStore(order.getStoreId(), order.formatEmail(), token);

    return order;
}
```

Figure 2.49: placeOrder code before

Code after refactoring:

```java
@Override
public Order placeOrder(String token, Long orderId) throws IllegalArgumentException {
    if (token.isEmpty()) {
        throw new IllegalArgumentException(INVALID_TOKEN_MESSAGE);
    }
    if (orderId == null) {
        throw new IllegalArgumentException(INVALID_ORDER_ID_MESSAGE);
    }

    Order order = orderRepository.getOne(orderId);

    validateOrderAttributes(order);

    for (Pizza pizza : order.getPizzas()) {
        pizza.setPrice(menuCommunication.getPizzaPriceFromMenu(pizza, token));
    }

    OrderBuilder orderBuilder = Builder.toBuilder(order);

    applyValidCoupons(order, orderBuilder, token);

    orderBuilder.setPrice(order.calculateTotalPrice());
    orderBuilder.setOrderStatus(Status.ORDER_PLACED);
    scheduleOrderCompletion(orderId);

    Order newOrder = orderBuilder.build();
    orderRepository.save(newOrder);

    // notify the store
    storeCommunication.sendEmailToStore(order.getStoreId(), order.formatEmail(), token);

    return order;
}
```

Figure 2.50: placeOrder code after

```
private void validateOrderAttributes(Order order) {
    if (order.getPizzas() == null || order.getPizzas().isEmpty()) {
        throw new IllegalArgumentException(INVALID_ORDER_CONTENTS_MESSAGE);
    }

    if (order.getSelectedTime() == null || order.getSelectedTime().isEmpty()) {
        throw new UnsupportedOperationException("No order time is selected");
    }

    if (order.getStatus() == null || order.getStatus() != Status.ORDER_ONGOING) {
        throw new IllegalArgumentException(NO_ACTIVE_ORDER_MESSAGE);
    }

    if (order.getLocation() == null || order.getLocation().isEmpty()) {
        throw new UnsupportedOperationException("No store location is selected");
    }
}
```

Figure 2.51: validateOrderAttributes code after

```
private void applyValidCoupons(Order order, OrderBuilder orderBuilder, String token) {
    if (order.getCoupons() != null && !order.getCoupons().isEmpty()) {
        ApplyCouponsToOrderModel applyCouponsToResponse = couponCommunication.applyCouponsToOrder(order.getPizzas(),
                new ArrayList<>(order.getCoupons()), token);
        if (applyCouponsToResponse.getCoupons() == null || applyCouponsToResponse.getCoupons().isEmpty()) {
            orderBuilder.setAppliedCoupon(null);
            order.setAppliedCoupon(null);
        } else {
            orderBuilder.setAppliedCoupon(applyCouponsToResponse.getCoupons().get(0));
            order.setAppliedCoupon(applyCouponsToResponse.getCoupons().get(0));
        }
        orderBuilder.setPizzas(applyCouponsToResponse.getPizzas());
        order.setPizzas(applyCouponsToResponse.getPizzas());
    }
}
```

Figure 2.52: applyValidCoupons code after

Process and result: To fix this problem we used method extraction to group code fragments that can be grouped together and move them into new methods. We named the methods after intent, namely `validateOrderAttributes` to validate order attributes and `applyValidCoupons` to validate and apply coupons. We copied the extracted code from the source method into the new target methods and added parameters to new methods. Then we replaced the extracted code in the source method with a call to the target methods. No test needed updating.

OrderProcessorImpl.cancelOrder()

Identified metrics: LOC, CC

Metric values before refactoring:

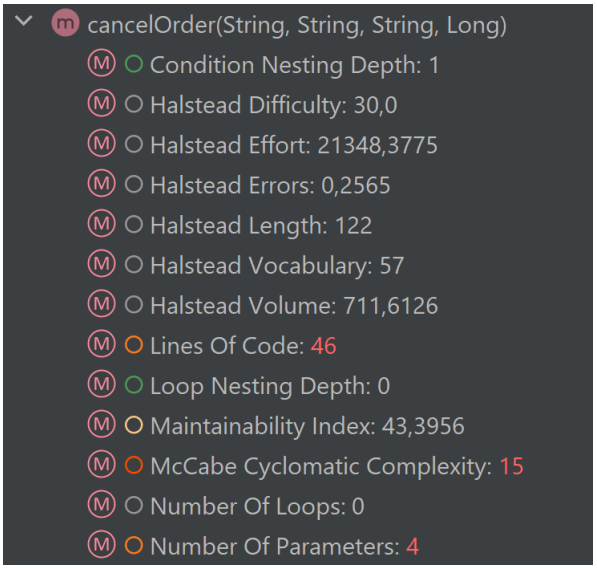


Figure 2.53: cancelOrder before refactoring

Metric values after refactoring:

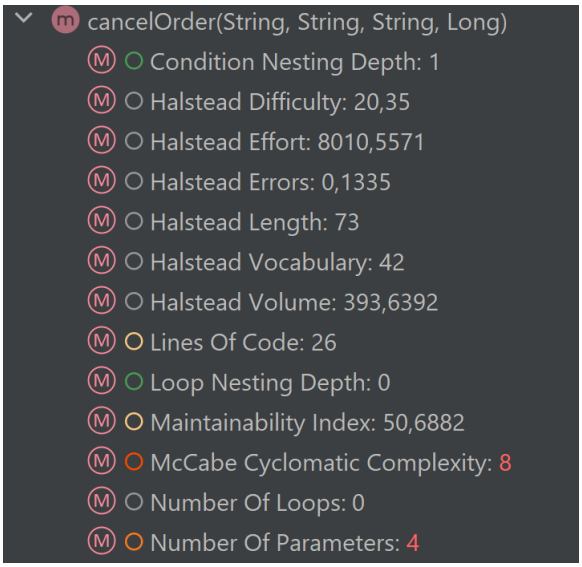


Figure 2.54: cancelOrder after refactoring

Process and result: To fix this problem we used method extraction to group code fragments that can be grouped together and move them into new methods. We named the methods after intent, namely `checkRoleAndOrderStatus` to validate if the user is authorized to cancel the order, `checkStoreManager` to check whether the current store manager is the manager of the store the order is part of, and `checkCancelTime` to check if the time of canceling the order is valid or it is too late to cancel the order. We have performed small refactoring of the logic of the code to improve readability, but mostly we copied the logic from the main method to multiple submethods. As there can be seen the number of Lines of Code per method has improved significantly, going from 46 to 26 (from orange to yellow) and the Cyclomatic Complexity has decreased from 15 to 8, which is also a significant improvement. We tried to lower it even more but we found out that there exists intrinsic complexity which cannot be lowered.

Code before refactoring:

```
@Override
public void cancelOrder(String token, String memberId, String roleName, Long orderId) throws Exception {
    if (orderId == null) {
        throw new IllegalArgumentException(INVALID_ORDER_ID_MESSAGE);
    }

    Order order = orderRepository.getOne(orderId);

    if (storeCommunication.validateManager(memberId, token)) {
        roleName = "store_admin";
    }

    switch (roleName) {
        case "regional_manager":
            if (order.getStatus() != Status.ORDER_ONGOING && order.getStatus() != Status.ORDER_PLACED) {
                throw new IllegalArgumentException(NO_ACTIVE_ORDER_MESSAGE);
            }
            break;
        case "store_admin":
            if (order.getStatus() != Status.ORDER_ONGOING && order.getStatus() != Status.ORDER_PLACED) {
                throw new IllegalArgumentException(NO_ACTIVE_ORDER_MESSAGE);
            }
            if (!orderRepository.getOne(orderId).getStoreId()
                    .equals(storeCommunication.getStoreIdFromManager(memberId, token))) {
                throw new UnsupportedOperationException("Not the store manager of this store");
            }
            break;
        case "customer":
            if (!order.getMemberId().equals(memberId)) {
                throw new UnsupportedOperationException("Access denied.");
            }
            if (order.getStatus() != Status.ORDER_ONGOING) {
                throw new IllegalArgumentException(NO_ACTIVE_ORDER_MESSAGE);
            }
            if (!timeValidation.isTimeValid(order.getSelectedTime(), DEADLINE_OFFSET)) {
                throw new Exception("You can no longer cancel the order");
```

Figure 2.55: cancelOrder code before refactoring

Code after refactoring:

```
public void cancelOrder(String token, String memberId, String roleName, Long orderId) throws Exception {
    if (orderId == null) {
        throw new IllegalArgumentException(INVALID_ORDER_ID_MESSAGE);
    }

    Order order = orderRepository.getOne(orderId);

    if (storeCommunication.validateManager(memberId, token)) {
        roleName = "store_admin";
    }

    if (!checkRoleAndOrderStatus(roleName, memberId, order)) {
        throw new IllegalArgumentException(NO_ACTIVE_ORDER_MESSAGE);
    }

    if (roleName.equals("store_admin") && !checkStoreManager(memberId, token, orderId)) {
        throw new UnsupportedOperationException("Not the store manager of this store");
    }

    if (roleName.equals("customer") && !checkCancelTime(order)) {
        throw new Exception("You can no longer cancel the order");
    }
```

Figure 2.56: cancelOrder code after refactoring

OrderEditorImpl.addTopping()

Identified metrics: NOM

Metric values before refactoring:

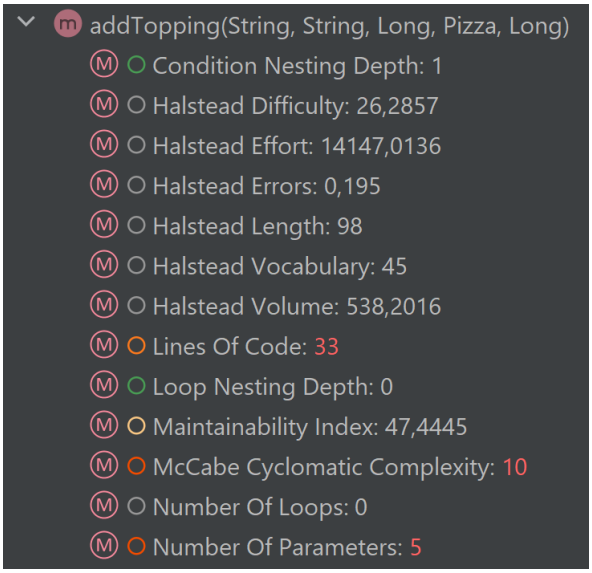


Figure 2.57: addTopping before refactoring

Metric values after refactoring:

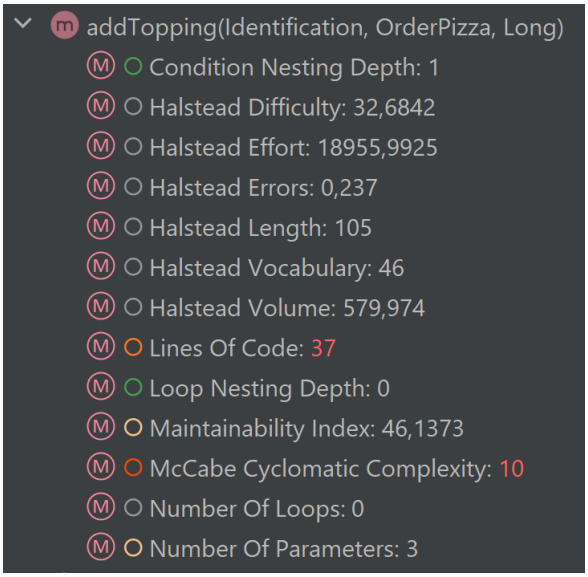


Figure 2.58: addTopping after refactoring

Code before refactoring:

```
@Override
public Allergens addTopping(String token, String memberId,
                            Long orderId, Pizza pizza, Long toppingId) throws Exception {
    if (orderId == null) {
        throw new Exception(invalidOrderId);
    } else if (orderRepository.getOne(orderId) == null
            || orderRepository.getOne(orderId).getStatus() != Status.ORDER_ONGOING) {
        throw new Exception(noActiveOrderMessage);
    } else if (toppingId == null) {
        throw new Exception("Invalid topping");
    } else if (pizza == null) {
        throw new Exception(invalidPizza);
    } else if (token == null) {
        throw new Exception(invalidToken);
    }
    Allergens allergens = new Allergens( allergensContent: "No allergens");
```

Figure 2.59: addTopping code before

Code after refactoring:

```
@Override
public Allergens addTopping(Identification identification, OrderPizza orderPizza, Long toppingId) throws Exception {
    if (orderPizza.getOrderId() == null) {
        throw new Exception(invalidOrderId);
    }

    if (orderRepository.getOne(orderPizza.getOrderId()) == null
            || orderRepository.getOne(orderPizza.getOrderId()).getStatus() != Status.ORDER_ONGOING) {
        throw new Exception(noActiveOrderMessage);
    }

    if (toppingId == null) {
        throw new Exception("Invalid topping");
    }

    if (orderPizza.getPizza() == null) {
        throw new Exception(invalidPizza);
    }

    if (identification.getToken() == null) {
        throw new Exception(invalidToken);
    }
```

Figure 2.60: addTopping code after

Process and result: To fix this problem we used parameter object refactoring. We observed that the parameters token and memberId are strongly connected, so there was decided to create a new model class called Identity. Also, the pizza and orderId are connected since a pizza is part of an order, so there was decided to have OrderPizza class that encapsulates the orderId and the Pizza that will be affected by the topping added. We made sure that we don't create these classes just for this method, but they can be used to improve most of the methods in the project, but for the purpose of the assignment we have changed just this method. There can be observed that the number of parameters went from a red threshold (5) to a yellow one (3).

3
Conclusion

The results of refactoring 5 classes and 5 methods in our software are shown in the report. By simplifying the system and using multiple refactoring approaches, we aimed to improve our code quality using Software Metrics to guide us. The report lists the metrics that were used as an indicator, along with their pre and post-refactoring values. The code before and after the refactoring is also included. The report shows how we detected the parts that have to be refactored (by setting and using thresholds), why we chose to refactor those metrics over others, and how we refactored the code to improve those metric levels. By looking at the metrics values after the refactorings, it is evident that the system's maintainability and quality have both increased while the complexity has decreased as a result of the restructuring process.

References

Filó, T. G., Bigonha, M., & Ferreira, K. (2015). A catalogue of thresholds for object-oriented software metrics. *Proc. of the 1st SOFTENG*, 48–55.

Sahraoui, H., Godin, R., & Miceli, T. (2000). Can metrics help to bridge the gap between the improvement of oo design quality and its automation? *Conference on Software Maintenance*, 154–162. https://doi.org/10.1109/ICSM.2000.883034

Wijendra, D., & Hewagamage, K. (2021). Analysis of cognitive complexity with cyclomatic complexity metric of software. *International Journal of Computer Applications*, *174*, 14–19. https://doi.org/10.5120/ijca2021921066

List of Figures