Group 5A / Assignment 2

Refactoring Report

After the project was concluded, we spent a week focusing on improving the project and the codebase. One of the ways to reach this goal was to use an external tool to see the refactoring needs of the code, more specifically, any work we can do on our methods or whole classes to establish a more maintainable and cleaner codebase.

Our team decided to use the Intellij plug-in MetricsTree, which gives an elaborate analysis over any class / method within our project. The team examined the project as a whole, and identified a few methods and classes as refactorable. Below are the methods and classes:

Methods:

- **applyCouponToBasket()**
- **overview()**
- **cancelOrder()**
- **registerUser()**
- **addCustomPizzaToBasket()**

Classes:

- **Basket**
- **OrderController**
- **RepoController**
- **BasketController**
- **RegistrationService**

This report is an in-depth overview of each refactoring, with specifications on what changed, how the changes were made, and how the statistics were improved through these changes.

Method refactoring #1: applyCouponToBasket

(lab-template/basket-microservice/src/main/java/nl/tudelft/sem/template/basket/controllers/basket/BasketCouponController.java)

```java
@PostMapping("/applyCoupon")
public ResponseEntity<String> applyCouponToBasket(@RequestBody String code) {
    String customerId = authManager.getNetId();
    if (basketService.getBasket(customerId) == null) {
        return ResponseEntity.badRequest().body("Your basket is empty!");
    }
    Basket basket = basketService.getBasket(customerId);
    Coupon coupon = couponService.getByCode(code);
    if (coupon == null) {
        return ResponseEntity.badRequest().body("Coupon code: " + code + " is invalid.");
    }
    if (basket.getBasketInfo().getCoupon() != null && basket.getBasketInfo().getCoupon().getCode().equalsIgnoreCase(code)) {
        return ResponseEntity.badRequest().body("This coupon is already applied.");
    }


    boolean applied = basketService.applyCouponToBasket(customerId, coupon);
    StringBuilder sb = new StringBuilder();
    DecimalFormat df = new DecimalFormat( pattern: "0.00");

    if (applied) {
        // If the coupon is applied, the details of the coupon is displayed.
        if (coupon.getType() == 'D') {
            sb.append(coupon.getRate()).append("% discount coupon has been applied.\n");
            sb.append("Current price: €").append(df.format(basket.getBasketInfo().getPrice()));
        } else if (coupon.getType() == 'F') {
            sb.append("Buy-one-get-one-free coupon has been applied.\n");
            sb.append("Current price: €").append(df.format(basket.getBasketInfo().getPrice()));
        } else {
            sb.append("Coupon code: ").append(coupon.getCode()).append(" has been applied.\n");
            sb.append("Current price: €").append(df.format(basket.getBasketInfo().getPrice()));
        }
        return ResponseEntity.ok(sb.toString());
    } else {
        // If the coupon is not applied because the previous coupon is cheaper, it tells that it has not been applied.
        sb.append("Coupon has not been applied because there is a cheaper coupon that has been applied already.");
        return ResponseEntity.badRequest().body(sb.toString());
    }
}
```

Method: applyCouponToBasket(String)

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	8	[0..3)
NOL		Number Of Loops	0	
LOC		Lines Of Code	46	[0..11)
NOPM		Number Of Parameters	1	[0..3)

For CND, the code structure for the problematic part of the method was:

- If coupon is applied
    - If coupon type is discount
    - Else if coupon type is buy-one-get-one-free
    - Else
- Else

The nested conditions increased the condition nested depth: we fixed this by moving the else block which accounted for when the coupon is not applied to the top. This way, we did not need nested condition blocks. The fixed code structure is:

- If coupon is not applied
- If coupon type is discount
- Else if coupon type is buy-one-get-one-free
- Else

Even after fixing CND, there were too many conditions which resulted in high cyclomatic complexity. In order to fix this, we used two refactoring techniques: ‘extract method’ and ‘replace conditional with polymorphism’.

We have extracted the second and third conditionals and added them to a new method couponChecker, as they served a similar purpose: checking if the coupon is eligible prior to applying it to the basket.

We have also extracted the last four conditionals and added them to the method couponApplier, as they are only reached after the applyCouponToBasket method is called. Additionally, in order to further reduce CC, we employed polymorphism. In the strategy interface, we have created the getMessage method, which is then overridden by strategy classes that represent different types of the coupon.

Extracting methods and using polymorphism lowered CC and LOC, reducing the code smells significantly. To reduce LOC further, there was also some minor code refactoring. Below are the changed methods and the metrics for methods applyCouponToBasket, couponChecker, and couponApplier after refactoring.

```
@PostMapping("/applyCoupon")
public ResponseEntity<String> applyCouponToBasket(@RequestBody String code) {
    if (basketService.getBasket(authManager.getNetId()) == null)
        return ResponseEntity.badRequest().body("Your basket is empty!");

    return couponChecker(authManager.getNetId(), code);
}

1개 사용 위치  JustinBJo *
public ResponseEntity<String> couponChecker(String customerId, String code) {
    Basket basket = basketService.getBasket(customerId);

    if (couponService.getByCode(code) == null) {
        return ResponseEntity.badRequest().body("Coupon code: " + code + " is invalid.");
    }
    if (basket.getBasketInfo().getCoupon() != null
            && basket.getBasketInfo().getCoupon().getCode().equalsIgnoreCase(code)) {
        return ResponseEntity.badRequest().body("This coupon is already applied.");
    }

    return couponApplier(customerId, couponService.getByCode(code));
}

1개 사용 위치  JustinBJo *
public ResponseEntity<String> couponApplier(String customerId, Coupon coupon) {
    boolean applied = basketService.applyCouponToBasket(customerId, coupon);
    if (!applied) return ResponseEntity.badRequest().body(
            "Coupon has not been applied because there is a cheaper coupon that has been applied already.");

    Basket basket = basketService.getBasket(customerId);
    DecimalFormat df = new DecimalFormat( pattern: "0.00");
    return ResponseEntity.ok( body: coupon.getMessage()
            + "Current price: €" + df.format(basket.getBasketInfo().getPrice()));
}
```

Method: applyCouponToBasket(String)

Metric	Metrics Set	Description	Value
CND		Condition Nesting Depth	1
LND		Loop Nesting Depth	0
CC		McCabe Cyclomatic Complexity	2
NOL		Number Of Loops	0
LOC		Lines Of Code	10
NOPM		Number Of Parameters	1

Method: couponChecker(String, String)

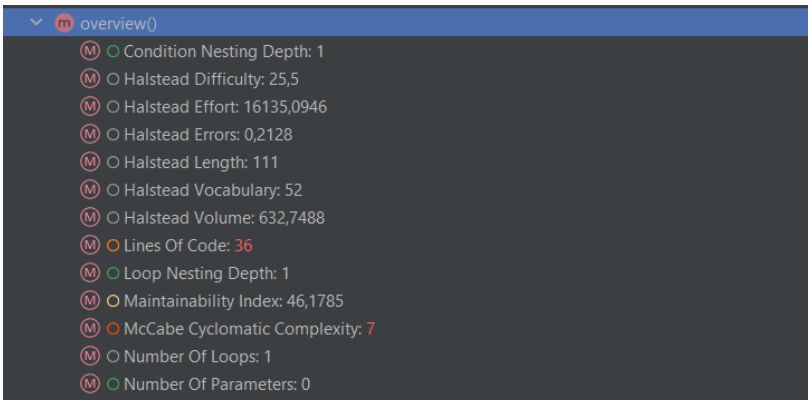
Metric	Metrics Set	Description	Value
CND		Condition Nesting Depth	1
LND		Loop Nesting Depth	0
CC		McCabe Cyclomatic Complexity	4
NOL		Number Of Loops	0
LOC		Lines Of Code	10
NOPM		Number Of Parameters	2

Method: couponApplier(String, Coupon)

Metric	Metrics Set	Description	Value
CND		Condition Nesting Depth	1
LND		Loop Nesting Depth	0
CC		McCabe Cyclomatic Complexity	2
NOL		Number Of Loops	0
LOC		Lines Of Code	8
NOPM		Number Of Parameters	2

Method refactoring #2: overview

(lab-template/basket-microservice/src/main/java/nl/tudelft/sem/template/basket/controllers/basket/BasketController.java)



The problem with the method was that it was way too big and it had a lot of if-checks to see what attributes from the coupon and basket to print. The way that we dealt with it was by using the '*extract method*' refactoring technique.

We successfully extracted 2 methods from the initial version — **printCoupon() and basketCalculatePrice()** — one that prints out the coupon attributes and one that prints out the basket attributes. Now we call those two methods in the overview method and we pass the coupon and the basket respectively. By doing so we improved the Cyclomatic complexity and the lines of code.

Here are the before and after of the classes:

Before:

```
@GetMapping("/overview")
public ResponseEntity<String> overview() {
    String customerId = authManager.getNetId();
    if (basketService.getBasket(customerId) == null) {
        return ResponseEntity.ok("Your basket is empty!");
    }

    Basket basket = basketService.getBasket(customerId);
    StringBuilder sb = new StringBuilder();
    sb.append("Pizzas:\n");
    if (basket.getPizzas().isEmpty()) sb.append("Nothing is in the basket yet!\n");

    DecimalFormat df = new DecimalFormat( pattern: "0.00");
    for (Pizza p : basket.getPizzas()) {
        sb.append(p.getName()).append(" | EUR ").append(df.format(p.getPrice())).append('\n');
    }

    sb.append("\nCoupon applied: ");
    Coupon coupon = basket.getCoupon();
    if (coupon == null) {
        sb.append("None");
    } else if (coupon.getType() == 'D') {
        sb.append(coupon.getCode()).append(" (").append(coupon.getRate()).append("% discount coupon)");
    } else if (coupon.getType() == 'F') {
        sb.append(coupon.getCode()).append(" (Buy-one-get-one-free coupon)");
    } else {
        sb.append(coupon.getCode());
    }

    sb.append("\n\nTotal: EUR ").append(df.format(basket.getPrice()));
```

```
    sb.append("\n\nTotal: EUR ").append(df.format(basket.getPrice()));

    sb.append("\n\nYour order will be ready at ").append(basket.timeToString()).append(".");

    return ResponseEntity.ok(sb.toString());
```

After:

```
@GetMapping("/overview")
public ResponseEntity<String> overview() {
    String customerId = authManager.getNetId();
    if (basketService.getBasket(customerId) == null) {
        return ResponseEntity.ok("Your basket is empty!");
    }
    Basket basket = basketService.getBasket(customerId);
    StringBuilder sb = new StringBuilder();
    sb.append("Pizzas:\n");
    if (basket.getBasketInfo().getPizzas().isEmpty()) sb.append("Nothing is in the basket yet!\n");
    sb.append(basketCalculatePrice(basket));
    sb.append("\nCoupon applied: ");
    sb.append(printCoupon(basket.getBasketInfo().getCoupon()));
    sb.append("\n\nYour order will be ready at ").append(basket.timeToString()).append(".");

    return ResponseEntity.ok(sb.toString());
}
```

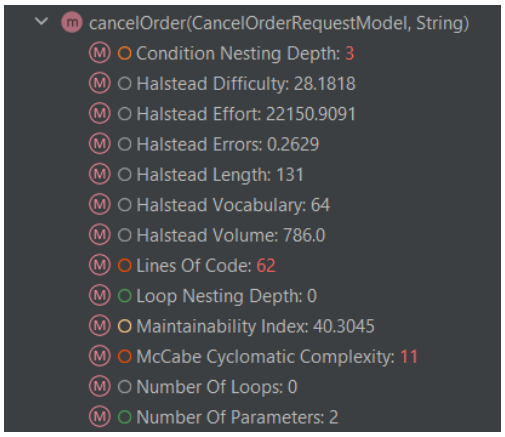
New methods:

```
 */
public String basketCalculatePrice(Basket basket) {
    StringBuilder sb = new StringBuilder();
    DecimalFormat df = new DecimalFormat( pattern: "0.00");
    for (Pizza p : basket.getBasketInfo().getPizzas()) {
        sb.append(p.getName()).append(" | EUR ").append(df.format(p.getPrice())).append('\n');
    }

    sb.append("\n\nTotal: EUR ").append(df.format(basket.getBasketInfo().getPrice()));
    return sb.toString();
}

/**
 * Prints the coupon
 * @param coupon - the coupon that will be printed
 * @return the string representation of the coupon
 */
public String printCoupon(Coupon coupon) {
    StringBuilder sb = new StringBuilder();
    if (coupon == null) {
        sb.append("None");
    } else if (coupon.getType() == 'D') {
        sb.append(coupon.getCode()).append(" (").append(coupon.getRate()).append("% discount coupon)");
    } else if (coupon.getType() == 'F') {
        sb.append(coupon.getCode()).append(" (Buy-one-get-one-free coupon)");
    } else {
        sb.append(coupon.getCode());
    }
    return sb.toString();
}
```

(Note that the LOC is still high but that is because the TreeMetrics plugin counts the JavaDocs of the methods as well)

Method refactoring #3: cancelOrder

(lab-template\order-microservice\src\main\java\nl\tudelft\sem\template\order\controllers\OrderController.java)

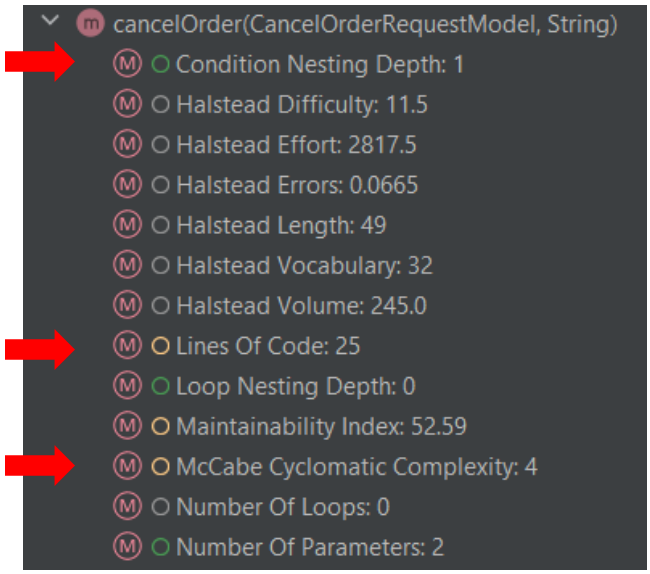


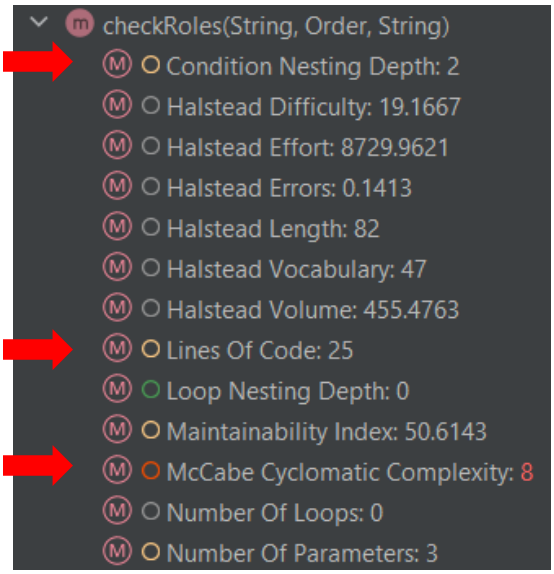
The metrics tree for the cancelOrder method in ***OrderController*** shows that the cyclomatic complexity, the nesting depth and the lines of code are big problems.

One look at the method itself shows why this is the case. There are too many null checks or role checks in this method, so it has to be split up. There is even one else-case that could be removed without breaking the method.

To fix our problem, the method needed to be split up. The most logical split would be to keep the main logic in this method, but to move all the role checks to a new method. This new method is called **checkRoles()** and takes care of almost all the if-statements, so this greatly reduces our complexity and lines of code. To make sure this new method does not face the same problems, minor refactoring has been done to reduce the lines of code. Finally to remove one more if-statement, the **removeCoupon()** method in the ***RestService*** now takes care of the case where coupons are null. Now this no longer has to be done by the **cancelOrder()** method.

The new metrics for cancelOrder and checkRoles look like this:





For reference, here is how the method looked before refactoring:

```java
@PostMapping("/cancel")
public ResponseEntity<String> cancelOrder(@RequestBody CancelOrderRequestModel request,
                                          @RequestHeader(name = "Authorization") String token) {
    // get credentials
    String userId = authManager.getNetId();
    String role = authManager.getRole();
    token = token.substring(7);
    // get information needed to validate request
    int orderId = request.getOrderId();
    Order order = orderMap.get(orderId);
    if (order == null) {
        return ResponseEntity.badRequest().body("Order " + orderId + " was not found.");
    }
    String storeId = order.getStoreId();
    LocalDateTime now = LocalDateTime.now();
    // for each role check if their actions are valid
    if (role == null) {
        return ResponseEntity.badRequest().body("No role was found for this user.");
    }
    if (role.equals("customer")) {
        // you can only delete orders that were made by you
        if (!userId.equals(order.getUserId())) {
            return ResponseEntity.badRequest().body("Order " + orderId
                    + " does not belong to user " + userId + ".");
        }
        // only allow in-time cancellations
        if (!checkTimeIsValid(now, order.getFinishTime())) {
            return ResponseEntity.badRequest().body("Order "
                    + orderId + " is too close to its finish time to be cancelled.");
        }
    } else if (role.equals("store")) {
        // stores can only cancel orders made to their store
        if (!storeId.equals(userId)) {
            return ResponseEntity.badRequest().body("Order " + orderId
                    + " does not belong to store " + userId + ".");
        }
        if (order.getCoupon() != null) {
            RemoveCouponRequestModel requestModel = new RemoveCouponRequestModel(order.getCoupon());
            ResponseEntity<String> resp = restService.removeCoupon(requestModel, token);
            if (!resp.getStatusCode().equals(HttpStatus.OK)) {
                return ResponseEntity.badRequest().body(resp.getBody());
            }
        }
    } else if (role.equals("manager")) {
        // TO-DO:
        // send notification to user whose order got cancelled
        //sendNotification(userId, "Order " + orderId + " got cancelled.")
    } else {
        // if the role is not one of the above three, the cancellation will not be accepted
        return ResponseEntity.badRequest().body("Role " + role + " does not exist.");
    }

    // if all checks are passed, the order can safely be removed
    this.orderQueue.remove();
    this.orderMap.remove(orderId);

    return ResponseEntity.ok("Success!");
```

And this is the result of our refactoring operations:

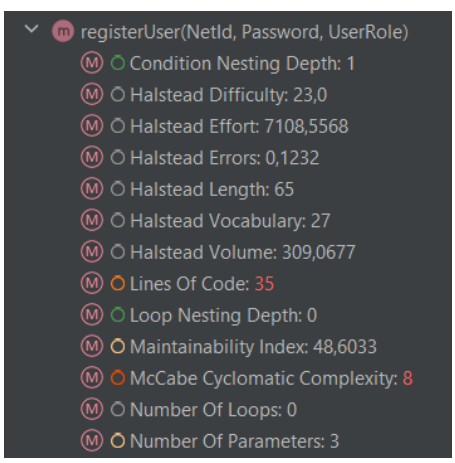
```java
@PostMapping("/cancel")
public ResponseEntity<String> cancelOrder(@RequestBody CancelOrderRequestModel request,
                                          @RequestHeader(name = "Authorization") String token) {
    // get credentials
    String role = authManager.getRole();
    token = token.substring(7);

    // get information needed to validate request
    int orderId = request.getOrderId();
    Order order = orderMap.get(orderId);

    if (order == null)
        return ResponseEntity.badRequest().body("Order " + orderId + " was not found.");
    if (role == null)
        return ResponseEntity.badRequest().body("No role was found for this user.");
    String roleCheckResult = checkRoles(role, order, token);
    if (roleCheckResult != null)
        return ResponseEntity.badRequest().body(roleCheckResult);

    // if all checks are passed, the order can safely be removed
    this.orderMap.remove(orderId);
    return ResponseEntity.ok("Success!");
}
```

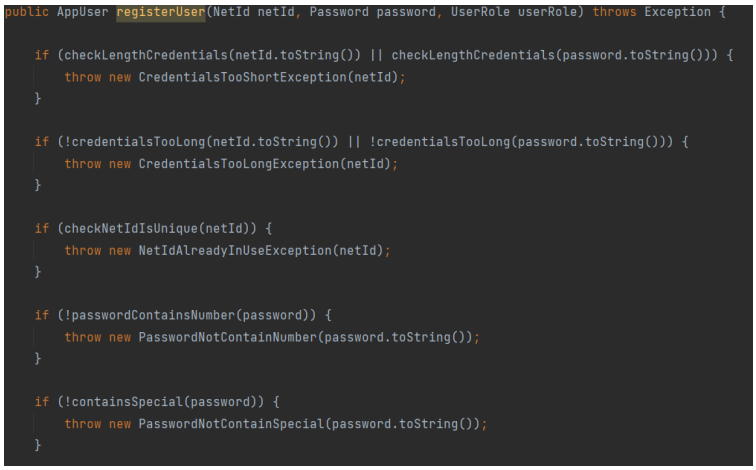
```java
private String checkRoles(String role, Order order, String token) {
    String userId = authManager.getNetId();
    String storeId = order.getStoreId();
    LocalDateTime now = LocalDateTime.now();
    if (role == null) {
        return "No role was found for this user.";
    }
    if (role.equals("customer")) {
        if (!userId.equals(order.getUserId())) {
            return "Order " + order.getOrderId() + " does not belong to user " + userId + ".";
        }
        // only allow in-time cancellations
        if (Duration.between(now, order.getFinishTime()).toMinutes() < 30) {
            return "Order " + order.getOrderId() + " is too close to its finish time to be cancelled.";
        }
    } else if (role.equals("store")) {
        // stores can only cancel orders made to their store
        if (!storeId.equals(userId)) {
            return "Order " + order.getOrderId() + " does not belong to store " + userId + ".";
        }
        if (order.getCoupon() != null) {
            RemoveCouponRequestModel requestModel = new RemoveCouponRequestModel(order.getCoupon());
            ResponseEntity<String> resp = restService.removeCoupon(requestModel, token);
            if (!resp.getStatusCode().equals(HttpStatus.OK)) return resp.getBody();
        }
    } else if (!role.equals("manager")) {
        // if the role is not one of the above three, the cancellation will not be accepted
        return "Role " + role + " does not exist.";
    }
    return null;
}
```

Method refactoring #4: registerUser

(lab-template/authentication-microservice/src/main/java/nl/tudelft/sem/template/authentication/domain/user/services/RegistrationService.java)

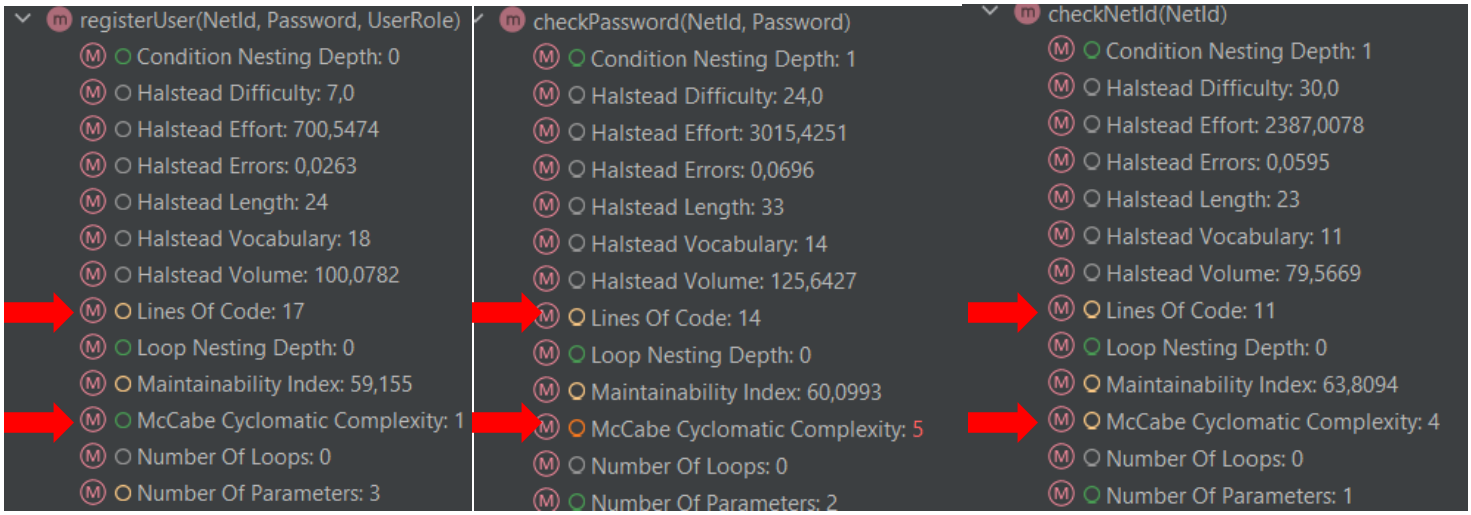
The metrics tree for the registerUser method in the ***RegistrationService*** class from the user microservice shows that both the Lines of Code and the Cyclomatic Complexity are bad.





```
public AppUser registerUser(NetId netId, Password password, UserRole userRole) throws Exception {

    if (checkLengthCredentials(netId.toString()) || checkLengthCredentials(password.toString())) {
        throw new CredentialsTooShortException(netId);
    }

    if (!credentialsTooLong(netId.toString()) || !credentialsTooLong(password.toString())) {
        throw new CredentialsTooLongException(netId);
    }

    if (checkNetIdIsUnique(netId)) {
        throw new NetIdAlreadyInUseException(netId);
    }

    if (!passwordContainsNumber(password)) {
        throw new PasswordNotContainNumber(password.toString());
    }

    if (!containsSpecial(password)) {
        throw new PasswordNotContainSpecial(password.toString());
    }
```

Upon further inspection of the method itself, the culprit appears to be the many checks for the password and username. See the picture on the right for reference. To reduce both the LOC and the CC we can extract these checks to their own methods: **checkNetId()** and **checkPassword()**. These methods now respectively take care of checking whether the username is allowed and whether the password is allowed.



As can be seen, the LOC and CC for registerUser are a lot better now and neither are the problems propagated to the newly created methods. The method extraction refactorization was successful.

The refactored registerUser method:

```java
public AppUser registerUser(NetId netId, Password password, UserRole userRole) throws Exception {
    checkNetId(netId);
    checkPassword(netId, password);

    HashedPassword hashedPassword = passwordHashingService.hash(password);
    List<Long> allergies = new ArrayList<>();
    allergies.add(0L);

    AppUser user = new AppUser(netId, hashedPassword, userRole, allergies);
    userRepository.save(user);
    return user;
}
```

The extracted methods:

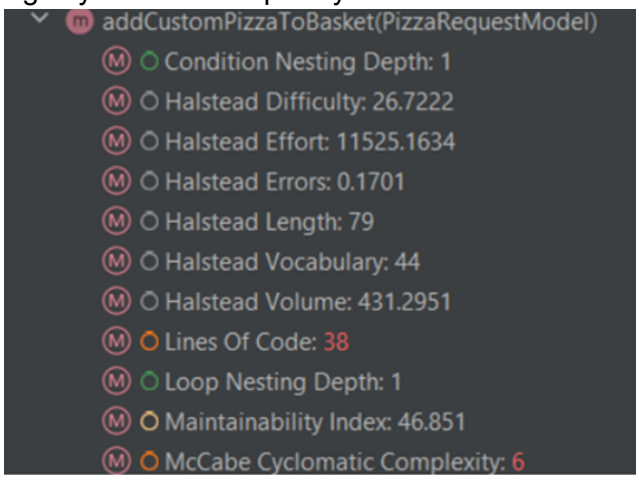
```java
private void checkNetId(NetId netId) throws Exception {
    if (checkNetIdIsUnique(netId)) {
        throw new NetIdAlreadyInUseException(netId);
    }
    if (credentialsTooShort(netId.toString())) {
        throw new CredentialsTooShortException(netId);
    }
    if (credentialsTooLong(netId.toString())) {
        throw new CredentialsTooLongException(netId);
    }
}
```

```java
private void checkPassword(NetId netId, Password password) throws Exception {
    if (!passwordContainsNumber(password)) {
        throw new PasswordNotContainNumber(password.toString());
    }
    if (!containsSpecial(password)) {
        throw new PasswordNotContainSpecial(password.toString());
    }
    if (credentialsTooShort(password.toString())) {
        throw new CredentialsTooShortException(netId);
    }
    if (credentialsTooLong(password.toString())) {
        throw new CredentialsTooLongException(netId);
    }
}
```

Method refactoring #5: addCustomPizzaToBasket

(lab-template/basket-microservice/src/main/java/nl/tudelft/sem/template/basket/controllers/basket/BasketPizzaController.java)

Running metricsTree in this methods gives us 2 particular bad code smells: lines of code and high cyclomatic complexity:



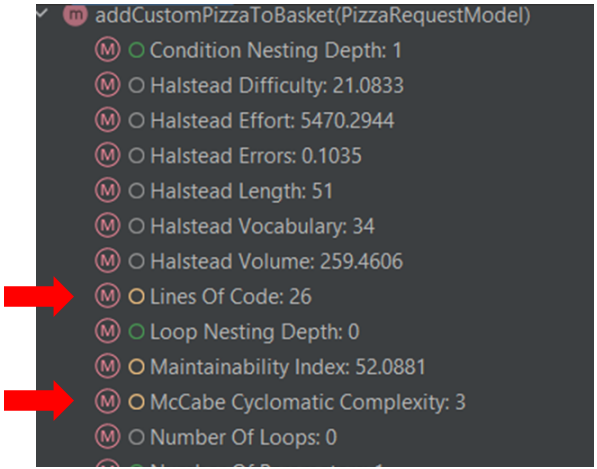
Originally this is what the method looked like:

```
@PostMapping("/addPizza/custom")
public ResponseEntity<String> addCustomPizzaToBasket(@RequestBody PizzaRequestModel pizzaReqModel) {
    String customerId = authManager.getNetId();
    System.out.println(customerId);
    if (basketService.getBasket(customerId) == null) {
        basketService.createBasket(customerId);
    }
    if (pizzaReqModel.getIngredients().isEmpty()) {
        return ResponseEntity.badRequest().body("Please provide at least " + "one ingredient.");
    }
    pizzaBuilder.setName(pizzaReqModel.getName());
    List<Ingredient> ingredients = new ArrayList<>();
    // Check if all ingredients are in the database.
    for (String ingredient : pizzaReqModel.getIngredients()) {
        if (ingredientService.getByName(ingredient) == null) {
            return ResponseEntity.badRequest().body("We do not have "
                    + ingredient + " as an ingredient on our inventory");
        } else ingredients.add(ingredientService.getByName(ingredient));
    }
    pizzaBuilder.setIngredients(ingredients);
    Pizza pizza = pizzaBuilder.build();
    if (pizza == null) {
        return ResponseEntity.badRequest().body("It is not possible to create this pizza");
    } else {
        basketService.addPizzaToBasket(customerId, pizza);
        return ResponseEntity.ok("Pizza " + pizza.getName()
                + " is added to the basket. Current basket is seen below: \n"
                + basketService.getBasket(customerId).toString());
    }
}
```

We can see that this method is indeed pretty complex, since it incorporates a lot of logic in order to check the validity of the request, like checking if the ingredients are present in the repository for example.

After some refactoring we managed to lower both the lines of code and the CC of this method to a significantly better value.

The metric below is the end result:



```java
@PostMapping("/addPizza/custom")
public ResponseEntity<String> addCustomPizzaToBasket(@RequestBody PizzaRequestModel pizzaReqModel) {
    String customerId = authManager.getNetId();
    checkBasketExistence(customerId);
    if (pizzaReqModel.getIngredients().isEmpty()) {
        return ResponseEntity.badRequest().body("Please provide at least " + "one ingredient.");
    }
    pizzaBuilder.setName(pizzaReqModel.getName());
    List<Ingredient> ingredients = new ArrayList<>();
    // Check if all ingredients are in the database.
    if (checkIngredientValidity(pizzaReqModel.getIngredients(), ingredients) != null)
        return checkIngredientValidity(pizzaReqModel.getIngredients(), ingredients);
    pizzaBuilder.setIngredients(ingredients);
    Pizza pizza = pizzaBuilder.build();
    basketService.addPizzaToBasket(customerId, pizza);
    return ResponseEntity.ok("Pizza " + pizza.getName() + " is added to the basket. Current basket is seen below:\n"
            + basketService.getBasket(customerId).toString());
}
```

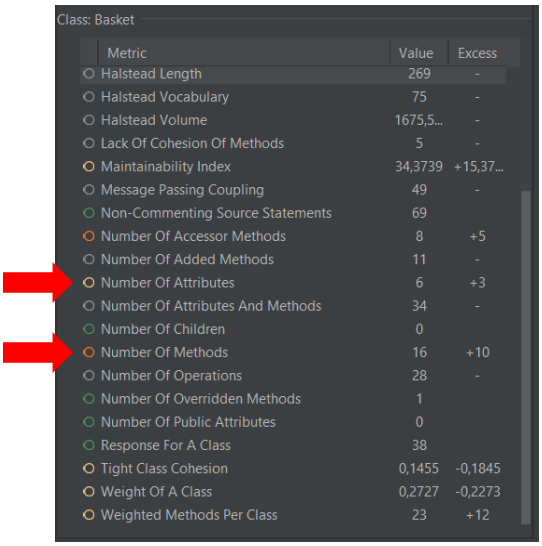
The main type of refactoring operation done here was the *'Extract Method'* technique. One of the if-conditions to check if the user had already created a basket, and the for-loop to check the existence of all the ingredients provided in the request were extracted to separate methods. Also note that we noticed that the null check for the pizza builder was redundant, since we already do the necessary checks to the integrity of the pizza beforehand.

```java
public void checkBasketExistence(String customerId) {
    if (basketService.getBasket(customerId) == null) {
        basketService.createBasket(customerId);
    }
}

public ResponseEntity<String> checkIngredientValidity(List<String> reqIngredients, List<Ingredient> ingredients) {
    for (String ingredient : reqIngredients) {
        if (ingredientService.getByName(ingredient) == null)
            return ResponseEntity.badRequest().body("We do not have "
                + ingredient + " as an ingredient on our inventory");
        else ingredients.add(ingredientService.getByName(ingredient));
    }
    return null;
}
```

These are methods extracted from **addCustomPizzaToBasket()**.

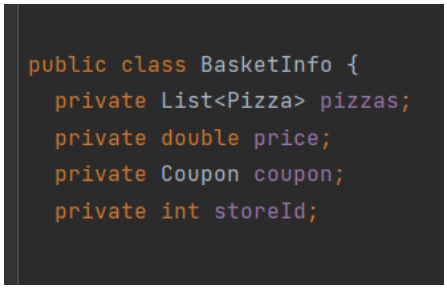
Class refactoring #1: Basket

(lab-template/commons/src/main/java/commons/Basket.java)



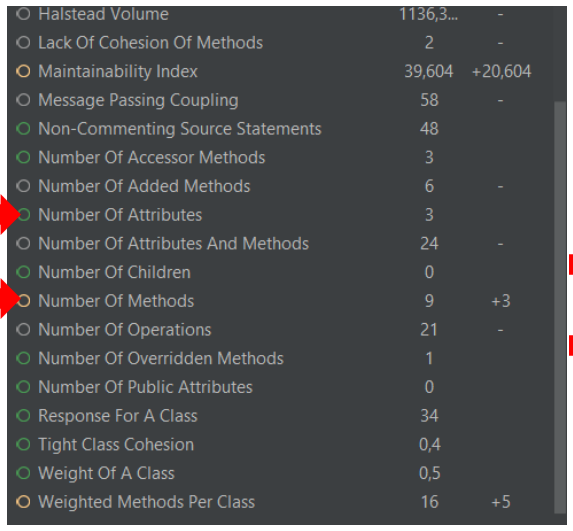
The problem with the basket class was that it had way too many methods and attributes in it.

We fix this by extracting another class from the basket - namely the **BasketInfo** class. This class contains the *price*, *pizza list, coupon and storeId* attributes which were previously in the Basket class.

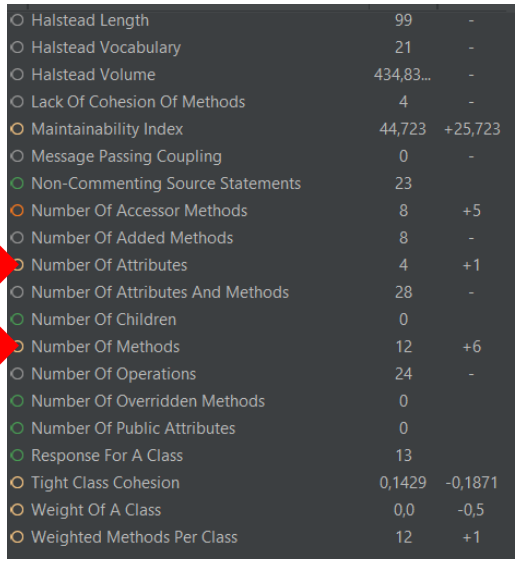


This change allowed us to move multiple get and set methods into the BasketInfo class, which greatly reduced the number of methods from basket. All the previous calls that were directly to the Basket class now reference the BasketInfo parameter in the Basket class to get the attributes such as pizza, price, etc.

Basket class:



BasketInfo class:



Class refactoring #2: OrderController

(lab-template\order-microservice\src\main\java\nl\tudelft\sem\template\order\controllers\OrderController.java)

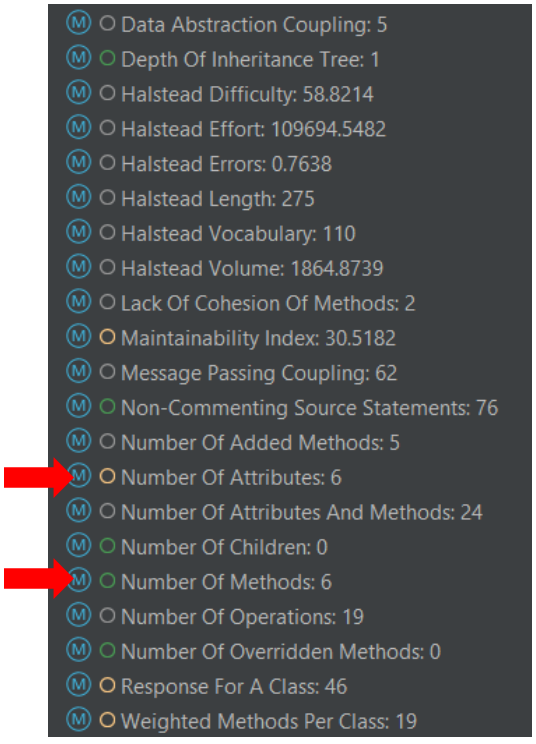
Using the Metrics Tree we identified two code smells in the **OrderController** class:

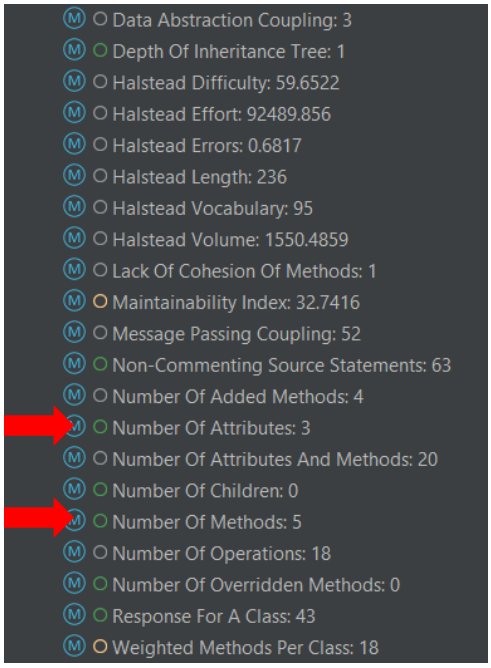
1. Unnecessary methods
2. Too many attributes

To fix the first, we removed the **checkTimeIsValid()** method as it was used only once. To fix the second code smell, I removed the *idCounter* and the *orderQueue*. I realized that the former could be replaced by making the *orderMap* an ordered map. This way, the element with the highest id can be requested in constant time. A new order id will now just be that highest id plus 1.

The other attribute I removed was the *orderQueue*. This was implemented to let the stores find the order that is closest to its delivery time. After all, this attribute was never used for this purpose and it could therefore be removed together with the lines adding items to this queue.

The new state of the metrics is as follows:





Class refactoring #3: RepoController

(lab-template\order-microservice\src\main\java\nl\tudelft\sem\template\basket\controllers\RepoController.java)

The ***RepoController*** was responsible for any functionality over three entity repositories:

- ***IngredientRepository***
- ***PizzaRepository***
- ***CouponRepository***

This caused several issues. The first issue comes with the size of the class, as storing the methods for 3 different repos meant having 10 methods and 240 lines of code. The second problem was that the constructor of the controller had 5 parameters (repo services + *RestService* + *AuthManager*) which was a red error.

Class: RepoController

Metric	Metrics Set	Description	Value	Regular R...
CHVL	Halstead ...	Halstead Volume	2033.628	
CHD	Halstead ...	Halstead Difficulty	55.7143	
CHL	Halstead ...	Halstead Length	296	
CHEF	Halstead ...	Halstead Effort	113302.129	
CHVC	Halstead ...	Halstead Vocabulary	117	
CHER	Halstead ...	Halstead Errors	0.7805	
WMC	Chidambe...	Weighted Methods Per Class	27	[0..12)
DIT	Chidambe...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidambe...	Response For A Class	54	[0..45)
LCOM	Chidambe...	Lack Of Cohesion Of Methods	1	
NOC	Chidambe...	Number Of Children	0	[0..2)
NOA	Lorenz-Ki...	Number Of Attributes	5	[0..4)
NOO	Lorenz-Ki...	Number Of Operations	23	
NOOM	Lorenz-Ki...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-Ki...	Number Of Added Methods	9	
SIZE2	Li-Henry ...	Number Of Attributes And Methods	27	
NOM	Li-Henry ...	Number Of Methods	10	[0..7)
MPC	Li-Henry ...	Message Passing Coupling	76	
DAC	Li-Henry ...	Data Abstraction Coupling	5	
NCSS	Chr. Clem...	Non-Commenting Source Statements	80	[0..1000)
CMI	Maintaina...	Maintainability Index	27.3025	[0.0..19.0]

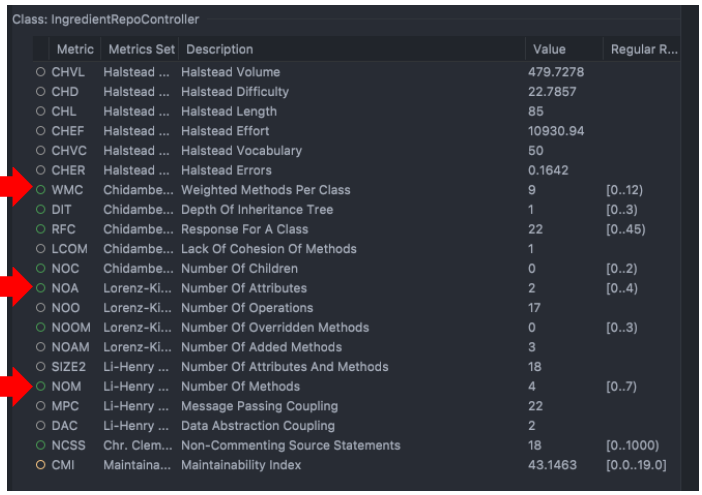
In order to fix this, the ***RepoController*** class is divided into the following three controllers:

- ***IngredientRepoController***
- ***PizzaRepoController***
- ***CouponRepoController***

After the division, the LOC of the controllers is 91-110-114, and the number of constructor parameters is 2-4-2. This positive impact results in a clearly divided code that is easier to work on.

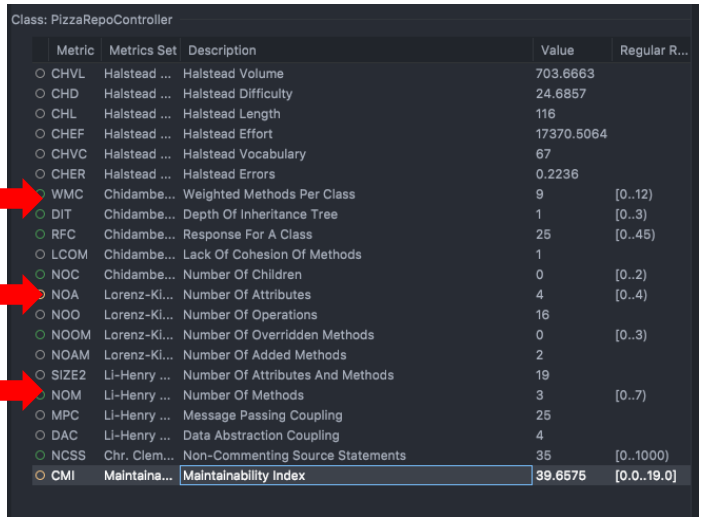
The path to the endpoints were previously **/api/repo/{entity}/…** for all methods and therefore, this didn't create any problems, as now, for example, the ingredient controller's address is **/api/repo/ingredients/** . Here is how the ***IngredientRepoController*** looks:

```java
/**
 * Ingredient DB controller is responsible for any incoming requests regarding the Ingredient DB. Includes:
 * - GETting all ingredients from the repo - ADDing a new ingredient into the repo - GETting all ingredients
 * in a list display for allergy selection
 */
@SuppressWarnings("PMD")
@RestController
@RequestMapping("/api/repo/ingredients")
public class IngredientRepoController {

    private final IngredientService ingredientService;
    private final transient AuthenticationManager authManager;

    /**
     * Constructor for the ingredient repo controller.
     * Params: ingredientService – IngredientService instance
     *         authManager – AuthenticationManager instance
     */
    @Autowired
    public IngredientRepoController(IngredientService ingredientService, AuthenticationManager authManager) {
        this.ingredientService = ingredientService;
        this.authManager = authManager;
    }
```



Class: IngredientRepoController

Metric	Metrics Set	Description	Value	Regular R...
CHVL	Halstead ...	Halstead Volume	479.7278	
CHD	Halstead ...	Halstead Difficulty	22.7857	
CHL	Halstead ...	Halstead Length	85	
CHEF	Halstead ...	Halstead Effort	10930.94	
CHVC	Halstead ...	Halstead Vocabulary	50	
CHER	Halstead ...	Halstead Errors	0.1642	
WMC	Chidambe...	Weighted Methods Per Class	9	[0..12)
DIT	Chidambe...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidambe...	Response For A Class	22	[0..45)
LCOM	Chidambe...	Lack Of Cohesion Of Methods	1	
NOC	Chidambe...	Number Of Children	0	[0..2)
NOA	Lorenz-Ki...	Number Of Attributes	2	[0..4)
NOO	Lorenz-Ki...	Number Of Operations	17	
NOOM	Lorenz-Ki...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-Ki...	Number Of Added Methods	3	
SIZE2	Li-Henry ...	Number Of Attributes And Methods	18	
NOM	Li-Henry ...	Number Of Methods	4	[0..7)
MPC	Li-Henry ...	Message Passing Coupling	22	
DAC	Li-Henry ...	Data Abstraction Coupling	2	
NCSS	Chr. Clem...	Non-Commenting Source Statements	18	[0..1000)
CMI	Maintaina...	Maintainability Index	43.1463	[0.0..19.0]



Class: PizzaRepoController

Metric	Metrics Set	Description	Value	Regular R...
CHVL	Halstead ...	Halstead Volume	703.6663	
CHD	Halstead ...	Halstead Difficulty	24.6857	
CHL	Halstead ...	Halstead Length	116	
CHEF	Halstead ...	Halstead Effort	17370.5064	
CHVC	Halstead ...	Halstead Vocabulary	67	
CHER	Halstead ...	Halstead Errors	0.2236	
WMC	Chidambe...	Weighted Methods Per Class	9	[0..12)
DIT	Chidambe...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidambe...	Response For A Class	25	[0..45)
LCOM	Chidambe...	Lack Of Cohesion Of Methods	1	
NOC	Chidambe...	Number Of Children	0	[0..2)
NOA	Lorenz-Ki...	Number Of Attributes	4	[0..4)
NOO	Lorenz-Ki...	Number Of Operations	16	
NOOM	Lorenz-Ki...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-Ki...	Number Of Added Methods	2	
SIZE2	Li-Henry ...	Number Of Attributes And Methods	19	
NOM	Li-Henry ...	Number Of Methods	3	[0..7)
MPC	Li-Henry ...	Message Passing Coupling	25	
DAC	Li-Henry ...	Data Abstraction Coupling	4	
NCSS	Chr. Clem...	Non-Commenting Source Statements	35	[0..1000)
CMI	Maintaina...	Maintainability Index	39.6575	[0.0..19.0]

Class: CouponRepoController

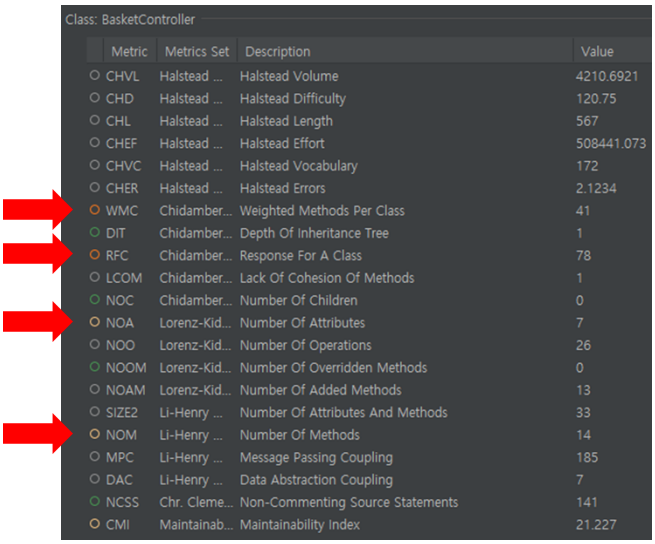
Metric	Metrics Set	Description	Value	Regular R...
CHVL	Halstead ...	Halstead Volume	695.4536	
CHD	Halstead ...	Halstead Difficulty	28.62	
CHL	Halstead ...	Halstead Length	122	
CHEF	Halstead ...	Halstead Effort	19903.88...	
CHVC	Halstead ...	Halstead Vocabulary	52	
CHER	Halstead ...	Halstead Errors	0.2448	
WMC	Chidambe...	Weighted Methods Per Class	11	[0..12)
DIT	Chidambe...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidambe...	Response For A Class	23	[0..45)
LCOM	Chidambe...	Lack Of Cohesion Of Methods	1	
NOC	Chidambe...	Number Of Children	0	[0..2)
NOA	Lorenz-Ki...	Number Of Attributes	2	[0..4)
NOO	Lorenz-Ki...	Number Of Operations	18	
NOOM	Lorenz-Ki...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-Ki...	Number Of Added Methods	4	
SIZE2	Li-Henry ...	Number Of Attributes And Methods	19	
NOM	Li-Henry ...	Number Of Methods	5	[0..7)
MPC	Li-Henry ...	Message Passing Coupling	29	
DAC	Li-Henry ...	Data Abstraction Coupling	2	
NCSS	Chr. Clem...	Non-Commenting Source Statements	30	[0..1000)
CMI	Maintaina...	Maintainability Index	38.7499	[0.0..19.0]

After the division, the metrics of the new controllers are seen on the left. The ***RepoController*** had 5 yellow warnings. Now the highest is 2 for the Pizza controller, which is due to the fact that ***ingredientService*** *(since pizza consists of ingredients)* and ***RestService*** *(for communication with other microservices)* are required.

Even on the common yellow error for maintainability index (CMI)—where a higher value is desired—the lowest of the 3 controllers is 38.75: **still more than 11 points above the previous value.**

Class refactoring #4: BasketController

(lab-template\order-microservice\src\main\java\nl\tudelft\sem\template\basket\controllers\BasketController.java)



Another class that we refactored was BasketController. Just like other controllers, it served many functionalities. The responsibilities of this class were:

- Managing pizzas in the basket
- Managing the coupon for the basket
- Managing the basket itself

This led to the class having too many methods and lines, which essentially made this class a “blob” class. Also, there were too many attributes for this class as it required all the service classes that provided the necessary functionalities. There were six attributes in total:

- PizzaService
- IngredientService
- PizzaBuilder
- BasketService
- CouponService
- AuthenticationManager

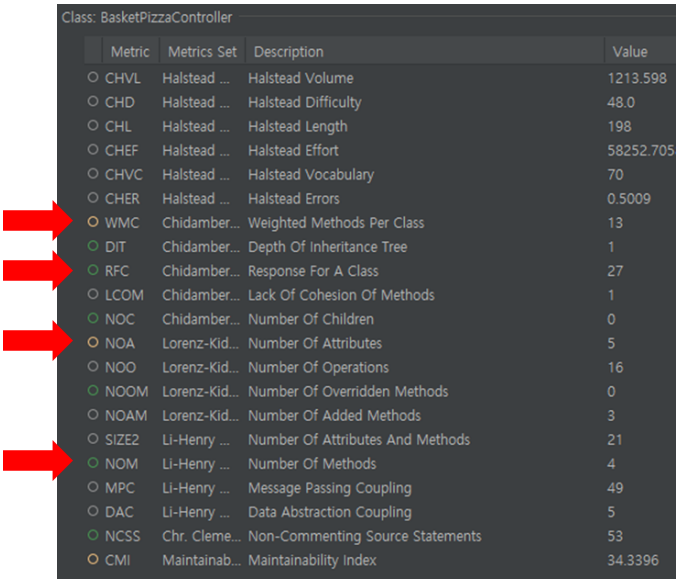
This also means that the constructor of this class had six parameters, which made MetricsTree give a red error. Upon looking at these attributes, we noticed that half of the attributes were related to pizzas.

Moreover, because there were too many methods that were used frequently, WMC and RFC were too high, giving the red errors. Therefore, we also decided to refactor this class by using class extraction like we did for RepoController.

As we classified that this class was serving three purposes, we divided this class into three separate classes: BasketPizzaController, BasketCouponController, and a new BasketController. The three classes were moved to a single directory “basket” under “controller” directory for conciseness.

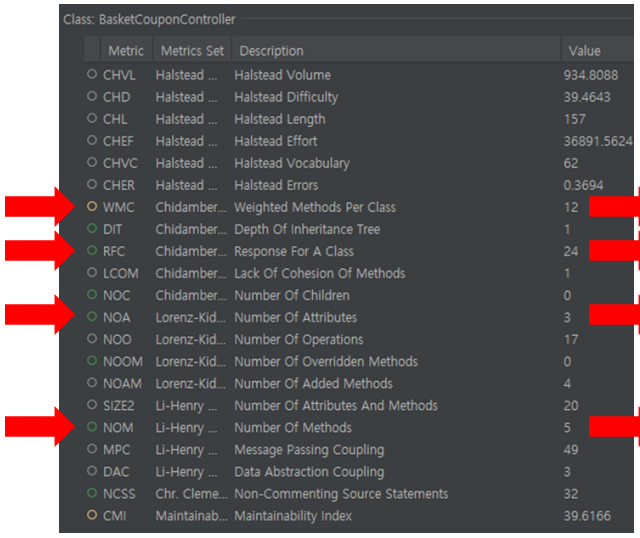
Before class extraction, LOC of the class was 344; after the class extraction, the LOC’s of the classes were 132, 97, and 165, giving 131 as average. Also, the average number of attributes were reduced from 6 to 3, and the average number of methods also decreased

from 14 to 5. This also resulted in lower WMC and RFC. The improved metrics are shown below.



Class: BasketPizzaController

Metric	Metrics Set	Description	Value
CHVL	Halstead ...	Halstead Volume	1213.598
CHD	Halstead ...	Halstead Difficulty	48.0
CHL	Halstead ...	Halstead Length	198
CHEF	Halstead ...	Halstead Effort	58252.705
CHVC	Halstead ...	Halstead Vocabulary	70
CHER	Halstead ...	Halstead Errors	0.5009
WMC	Chidamber...	Weighted Methods Per Class	13
DIT	Chidamber...	Depth Of Inheritance Tree	1
RFC	Chidamber...	Response For A Class	27
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1
NOC	Chidamber...	Number Of Children	0
NOA	Lorenz-Kid...	Number Of Attributes	5
NOO	Lorenz-Kid...	Number Of Operations	16
NOOM	Lorenz-Kid...	Number Of Overridden Methods	0
NOAM	Lorenz-Kid...	Number Of Added Methods	3
SIZE2	Li-Henry ...	Number Of Attributes And Methods	21
NOM	Li-Henry ...	Number Of Methods	4
MPC	Li-Henry ...	Message Passing Coupling	49
DAC	Li-Henry ...	Data Abstraction Coupling	5
NCSS	Chr. Cleme...	Non-Commenting Source Statements	53
CMI	Maintainab...	Maintainability Index	34.3396



Class: BasketCouponController

Metric	Metrics Set	Description	Value
CHVL	Halstead ...	Halstead Volume	934.8088
CHD	Halstead ...	Halstead Difficulty	39.4643
CHL	Halstead ...	Halstead Length	157
CHEF	Halstead ...	Halstead Effort	36891.5624
CHVC	Halstead ...	Halstead Vocabulary	62
CHER	Halstead ...	Halstead Errors	0.3694
WMC	Chidamber...	Weighted Methods Per Class	12
DIT	Chidamber...	Depth Of Inheritance Tree	1
RFC	Chidamber...	Response For A Class	24
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1
NOC	Chidamber...	Number Of Children	0
NOA	Lorenz-Kid...	Number Of Attributes	3
NOO	Lorenz-Kid...	Number Of Operations	17
NOOM	Lorenz-Kid...	Number Of Overridden Methods	0
NOAM	Lorenz-Kid...	Number Of Added Methods	4
SIZE2	Li-Henry ...	Number Of Attributes And Methods	20
NOM	Li-Henry ...	Number Of Methods	5
MPC	Li-Henry ...	Message Passing Coupling	49
DAC	Li-Henry ...	Data Abstraction Coupling	3
NCSS	Chr. Cleme...	Non-Commenting Source Statements	32
CMI	Maintainab...	Maintainability Index	39.6166

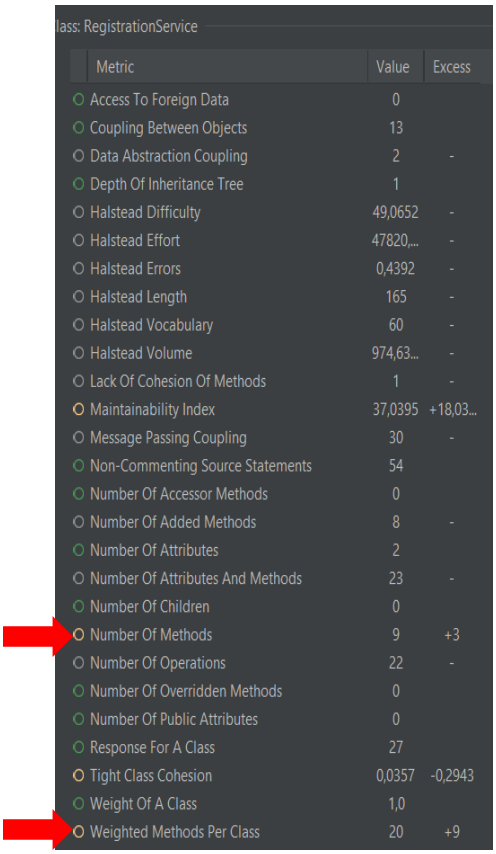
Class: BasketController

Metric	Metrics Set	Description	Value
CHVL	Halstead ...	Halstead Volume	1638.2759
CHD	Halstead ...	Halstead Difficulty	64.1702
CHL	Halstead ...	Halstead Length	244
CHEF	Halstead ...	Halstead Effort	105128.51...
CHVC	Halstead ...	Halstead Vocabulary	105
CHER	Halstead ...	Halstead Errors	0.7425
WMC	Chidamber...	Weighted Methods Per Class	18
DIT	Chidamber...	Depth Of Inheritance Tree	1
RFC	Chidamber...	Response For A Class	50
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1
NOC	Chidamber...	Number Of Children	0
NOA	Lorenz-Kid...	Number Of Attributes	3
NOO	Lorenz-Kid...	Number Of Operations	19
NOOM	Lorenz-Kid...	Number Of Overridden Methods	0
NOAM	Lorenz-Kid...	Number Of Added Methods	6
SIZE2	Li-Henry ...	Number Of Attributes And Methods	22
NOM	Li-Henry ...	Number Of Methods	7
MPC	Li-Henry ...	Message Passing Coupling	87
DAC	Li-Henry ...	Data Abstraction Coupling	3
NCSS	Chr. Cleme...	Non-Commenting Source Statements	60
CMI	Maintainab...	Maintainability Index	31.5155

Class refactoring #5: RegistrationService

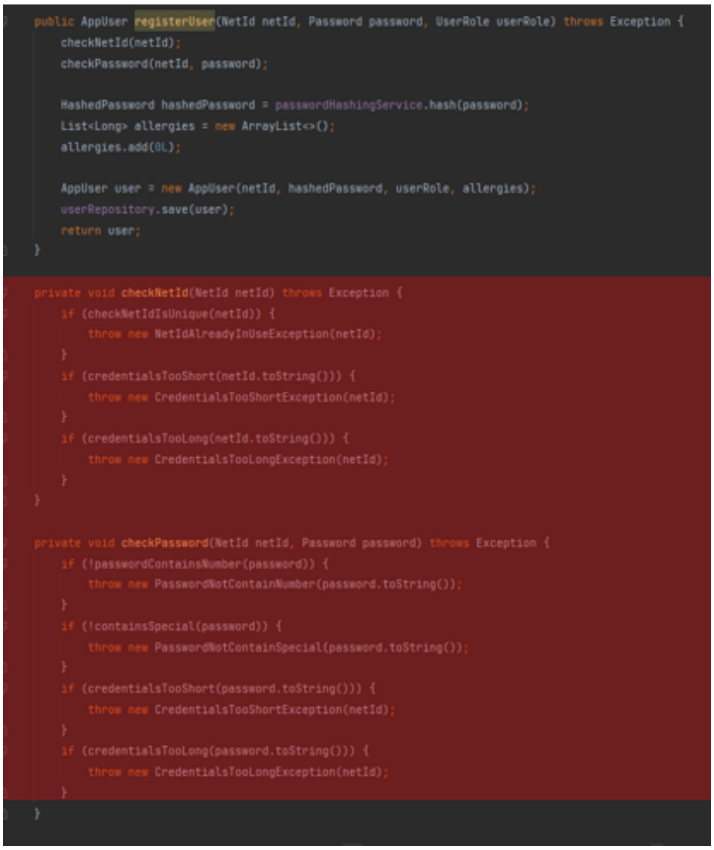
(lab-template/authentication-microservice/src/main/java/nl/tudelft/sem/template/authentication/domain/user/services/RegistrationService.java)

The ***RegistrationService*** class currently has a high number of methods according to the metrics. On top of that the value of the weighted methods is almost double of what it should be. When looking at the class it quickly becomes clear why: A good chunk of the methods is used to validate the password and username with which the user tries to register. To reduce the number of methods, we can extract the methods for the validation to their own class.



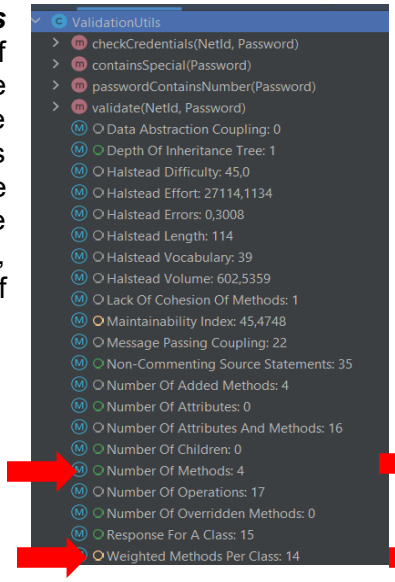
lass: RegistrationService

Metric	Value	Excess
Access To Foreign Data	0	
Coupling Between Objects	13	
Data Abstraction Coupling	2	-
Depth Of Inheritance Tree	1	
Halstead Difficulty	49,0652	-
Halstead Effort	47820,...	-
Halstead Errors	0,4392	-
Halstead Length	165	-
Halstead Vocabulary	60	-
Halstead Volume	974,63...	-
Lack Of Cohesion Of Methods	1	-
Maintainability Index	37,0395	+18,03...
Message Passing Coupling	30	-
Non-Commenting Source Statements	54	
Number Of Accessor Methods	0	
Number Of Added Methods	8	-
Number Of Attributes	2	
Number Of Attributes And Methods	23	-
Number Of Children	0	
Number Of Methods	9	+3
Number Of Operations	22	-
Number Of Overridden Methods	0	
Number Of Public Attributes	0	
Response For A Class	27	
Tight Class Cohesion	0,0357	-0,2943
Weight Of A Class	1,0	
Weighted Methods Per Class	20	+9

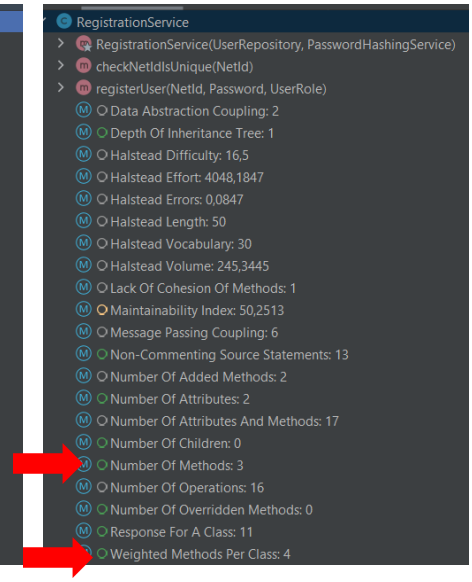


```
public AppUser registerUser(NetId netId, Password password, UserRole userRole) throws Exception {
    checkNetId(netId);
    checkPassword(netId, password);

    HashedPassword hashedPassword = passwordHashingService.hash(password);
    List<Long> allergies = new ArrayList<>();
    allergies.add(0L);

    AppUser user = new AppUser(netId, hashedPassword, userRole, allergies);
    userRepository.save(user);
    return user;
}

private void checkNetId(NetId netId) throws Exception {
    if (checkNetIdIsUnique(netId)) {
        throw new NetIdAlreadyInUseException(netId);
    }
    if (credentialsTooShort(netId.toString())) {
        throw new CredentialsTooShortException(netId);
    }
    if (credentialsTooLong(netId.toString())) {
        throw new CredentialsTooLongException(netId);
    }
}

private void checkPassword(NetId netId, Password password) throws Exception {
    if (!passwordContainsNumber(password)) {
        throw new PasswordNotContainNumber(password.toString());
    }
    if (!containsSpecial(password)) {
        throw new PasswordNotContainSpecial(password.toString());
    }
    if (credentialsTooShort(password.toString())) {
        throw new CredentialsTooShortException(netId);
    }
    if (credentialsTooLong(password.toString())) {
        throw new CredentialsTooLongException(netId);
    }
}
```

The new ***ValidationUtils*** class now takes care of validating the username and password, while the ***RegistrationService*** class focuses on registering the users. The metrics have shown nice improvement, especially the number of methods in the classes.



ValidationUtils
- checkCredentials(NetId, Password)
- containsSpecial(Password)
- passwordContainsNumber(Password)
- validate(NetId, Password)
- Data Abstraction Coupling: 0
- Depth Of Inheritance Tree: 1
- Halstead Difficulty: 45,0
- Halstead Effort: 27114,1134
- Halstead Errors: 0,3008
- Halstead Length: 114
- Halstead Vocabulary: 39
- Halstead Volume: 602,5359
- Lack Of Cohesion Of Methods: 1
- Maintainability Index: 45,4748
- Message Passing Coupling: 22
- Non-Commenting Source Statements: 35
- Number Of Added Methods: 4
- Number Of Attributes: 0
- Number Of Attributes And Methods: 16
- Number Of Children: 0
- Number Of Methods: 4
- Number Of Operations: 17
- Number Of Overridden Methods: 0
- Response For A Class: 15
- Weighted Methods Per Class: 14



RegistrationService
- RegistrationService(UserRepository, PasswordHashingService)
- checkNetIdIsUnique(NetId)
- registerUser(NetId, Password, UserRole)
- Data Abstraction Coupling: 2
- Depth Of Inheritance Tree: 1
- Halstead Difficulty: 16,5
- Halstead Effort: 4048,1847
- Halstead Errors: 0,0847
- Halstead Length: 50
- Halstead Vocabulary: 30
- Halstead Volume: 245,3445
- Lack Of Cohesion Of Methods: 1
- Maintainability Index: 50,2513
- Message Passing Coupling: 6
- Non-Commenting Source Statements: 13
- Number Of Added Methods: 2
- Number Of Attributes: 2
- Number Of Attributes And Methods: 17
- Number Of Children: 0
- Number Of Methods: 3
- Number Of Operations: 16
- Number Of Overridden Methods: 0
- Response For A Class: 11
- Weighted Methods Per Class: 4