Jeremy Cross
Iden Ellia
Abdullah Alasmari

ENSE 470 Refactoring Report (Team Covfefe)

GitHub Organization

When viewing team Covfefe's github page, there are folders labeled for each of the milestones containing the appropriate set of slides for that milestone as well as any other information required for that milestone. Other information like the user story map and Lo-Fi prototypes are also available individually under the corresponding milestone however the ATDD templates were absent and only accessible within the milestone slides themselves. The software folder below the milestone folders contains all the software for their application. There is also a readme file included that provides the link for their application page. The documation for each of the software files is not very detailed and just provides a short sentence explaining the update. Overall the organization of the page is good and it isn't really difficult to find the desired file or presentation.

Using the Application

According to the the user story map and ATDD templates the application starts off with a login page were either the user, approver, or analyst can login. The user is first able to request a software that they want approval for. The approver is then able to look at the software requests for approval and decide whether to approve or deny the software. Once the software is approved the analyst then confirms with the approver. The approved software is then moved from the pending requests section to completed requests. The analyst then enters the user into the software list database allowing the user to use the software. The application definitely follows this design however there is some functionality that doesn't work as intended. The application is also missing a logout button for the user, approver and analyst login types.

User's Perspective

Once logged in as the user, the first screen that appears allows the user to enter the desired software. When we entered the names into the search box, possible software names were listed below based on possible existing software in the database. This worked as described in the test templates. When we entered a valid name of a software that existed, the submit software button was pressed and the input was accepted into the database as shown from the tests. However when we entered an invalid name of software and pressed the submit button the name was also accepted. There was no prompt to prevent invalid software from being submitted. There is also no indication of what software(s) the user has requested.

Approvers Perspective

When logged in as the approver, the first screen displayed shows a list of pending software requests. When we clicked on a specific software request for more information, a request box appears on screen allowing us to either approve or deny the software. If the approve or deny button is pressed a confirmation box appears checking if that is really the desired action. This functionality worked as intended and once the software that we approved was confirmed it was then sent to the analyst.

Analysts Perspective

From the analyst's perspective, there are 3 different tabs displayed at the top, pending requests, completed requests, and manual entry. The first screen that appears is the pending request screen. Here we were able to see the software that we approved from the approver login and follow up by confirming the correct approver from the correct name, phone number and email. This worked as intended but once the approver has been confirmed the software is suppose to have moved over to the completed requests tab. The software that we confirmed did not end up going there. The functionality for the email notification to the approver was pushed to the next release so there was no notification. Theoretically if the request was confirmed and completed we would then be able to enter the name of the user into the software list database under the manual entry tab. We tried adding the username and software name for the manual entry anyway and the information appeared to be added to the database but there was no notification popup saying it had been added. Again the functionality to email the user of there software confirmation was pushed to the next release.

Code Implementation

Code Smells

1. User input for variables can prove to be an insecure environment allowing users to login with any credentials using user defined sql expressions, or even dump the database.

```
$conn = new mysqli("localhost", "thoma26s", "008096", "thoma26s");
if($conn->connect_error){
    die("Connection failed: " . $conn->connect_error);
}
$username = $_POST["user"];
$password = $_POST["pswd"];

$sql = "SELECT tier FROM ense470users WHERE username='$username' AND password='$password'";
$result = $conn->query($sql);
```

(Try signing in with any username, and password that evaluates to “' OR '1'='1”)

2. Not using different files for the MYSQL session, this enables a lot of code reuse. (This is apparent in all of the files, as well as using the same code more than once in the same file)

```
if(isset($_POST['search'])){
        $response="<ul><li>No data found!</ul></li>";
        $conn = new mysqli("localhost","thoma26s","008096","thoma26s");
        $q = $conn->real_escape_string($_POST['q']);
        $sql = $conn->query("SELECT softwarename FROM ense470software WHERE softwarename LIKE
        if ($sql->num_rows>0){
                $response = "<ul>";

                while($data= $sql->fetch_array())
                        $response .= "<li>".$data['softwarename']."</li>";

                $response .= "</ul>";
        }
        exit($response);
}
```

3. Uses AJAX in the code but fails to use it when requesting software.

4. There are only client-side checks for requesting software.

```
$softname = $_POST["softwarename"];
$comment = $_POST["usercomment"];
$username = $_SESSION["currUsername"];
$sql = "INSERT INTO ense470request (owner,software,comments) VALUES ('$username','$softname','$comment')";
//$cCode = $_SESSION["currentCode"];
//$query = "SELECT temp FROM tablename WHERE input ORDER BY ID DESC";
//$replyResult = $conn->query($query);

if ($conn->query($sql)===TRUE) {
    header("Location: usersplash.php");
}
else{
    echo "Error: " . $sql . "<br>" . $conn->error;
   }

   $conn->close();

?>
```

Design Techniques

1. This code did not use the OOP paradigm, therefore most of the patterns were not explicit. (No relationships between objects because there are no objects).
2. The strategy pattern was used for determining which type the logged in user was (analyst, user, approver) in order to execute the correct algorithm.
3. The facade pattern was used implicitly to complete a request from the user's perspective. The user had control to communicate with the analyst & approver, but the user did not have control to approve the request themselves. There is an apparent abstraction layer between the approver, analyst, and the user.
4. The observer pattern was used to notify analysts and approvers of software that needed to be handled.
5. The composite pattern was not used because there weren't any composite objects.

Refactoring Techniques

I would start off by separating code that i will intend to reuse, as there are alot of methods that implement the same thing in the same file as described above (mysql session, sessions). I would then choose to stick with the OOP paradigm in order to maintain a loosely coupled design using PHP classes. The observer pattern would be used to notify users of pending and active requests, either by email or a notification system on the website. I would also use the factory pattern to instantiate objects for different types of authentication (User, analyst, approver). All authentication objects will inherit from the same abstract parent, allowing an abstract method to be chosen at runtime (Strategy pattern). I would also add server-side validation in order to reduce spam and not allow users to make invalid requests.