Milestone 7:Tiger Vikings.
Quality Report: TimBits.

This report will be reviewing the quality and methodology of another group's project and code. More specifically, the writers of this report, The Tiger Vikings, will be critiquing the work done by team Timbits. Additionally, we will attempt to provide reasonable methods by which they might have improved their code, were they to proceed with development. Finally, there will be an analysis of the design patterns utilized by their team.

The code written by team Timbits, was well structured and sound as a whole, was plagued by several small issues that reduced the usability of their software. One such example would be the lack of cross-browser compatibility. When running their software, an analyst may find themselves unable to approve or deny software requests, provided they aren't using the correct browser for the project. In particular, the software will run without issue, but only so long as they are using google chrome, rather than mozilla firefox, internet explorer, or any other browsing methods. This can create issues for users who prefer other browsers, or are unable to use chrome for any reason.

Another minor inconvenience can be found in the lack of error notification provided by their software. Team Timbits has acknowledged this as being an issue, as they intended to have this feature implemented for their MVP release, but were unable to reach that goal, due to time constraints. The ATDD provided by Timbits clearly shows their intent, but inability to reach those intentions; however, it's worth noting that this is less of an implementation error than it is a scope error, as we are certain that this feature would have made it in, had they the time to include it.

Design Patterns:
The Model-View-Controller design pattern heavily influenced the design of this project. The Model-View-Controller design pattern consists of three parts: the Model, which contains all of the data of the program and the logic of the program; the View, which presents the user interface; and the Controller, which controls responding to the users actions. We believe that the code provided by Timbits appropriately reflects this design pattern. One benefit that was mentioned in their milestone 6 is the idea of "decoupling the front-end and the back-end which allowed [them] to work on [the] project simultaneously." This is reflected in clear demarcations of responsibility and structure in the code. Once again, a possible deviation from this design pattern is the lack of any sort of logic at all in the Model portion of the project. Other benefits of the MVC design pattern are that it helps to prevent repetition in code and it makes the code easier to modify for any future releases as each portion is relatively separated from one another. If the project had continued into MVP 2 and beyond, this would have been a great asset.

Code Smells Discovered.
Long Method:

This happens when a method contains too many lines of code usually beyond ten lines. An example of lengthy code can be found within openRequest.php as there are many lines (particularly between lines 22 and 58) where the code could have benefited from extraction into smaller methods for simplicity's sake. This would eliminate a great portion of the page's dirt, and allow the comments to be handled by method names, rather than documentation.

Documentation(comments)
Documentation plays a vital role when it comes to development. One should always have comments that explains the logic of an entire program or the different code segments in the program. In the case of TimBits there is little or no documentation/comments that is been put out to explain different logics or methods used in this development, hence, leading to lack of comprehension of certain code segments. For someone who does not have the good knowledge about the programming language used, there were no comments explaining code fragments that are clearly not obvious to the reader and also explaining why certain things were being implemented in a particular way.
Suggestions:
- Reconstruction of the code structure will be a good idea if there is a need to make the code obvious to the reader.
- Having complex expressions that are not understandable, it would be a good idea to breakdown the expression into smaller expressions.

Data Class:
We found that the class Request in request.php was a data class. A data class is a class that only exists as a container for data that is passed to other classes. That is, it does not contain any methods that work on the data in the class itself. The request class contains all of the pertinent data about a request such as the username of the requester, the email of the requester, etc. We found no true consensus over whether this is an actual code smell or simply another way of designing a project. On the one hand, data does not exist in a vacuum and computers operate on data in order to do something useful. Continuing this line of thought, if the data is ever modified or used, it will necessarily be done outside of the class. This may potentially unintentionally obfuscate the code. Imagine the case of someone new trying to understand the code, looking at the Request class, and then having to navigate through the entire project to find how the data is operated on. Furthermore, removing data classes reduces the coupling present between classes. Low coupling seeks to reduce classes' dependence on each other, which a data class necessarily prevents by having its data being used exclusively outside of itself. On the other hand, we found the project was designed using a MVC design pattern. Model-View-Controller emphasizes the use of models containing the data and business logic and Controllers that handle user interactions. The requests have no internal logic outside of being manipulated by the users, approvers, and analysts. In the MVC system, the user would interact with the controller, and the controller manipulates the model which contains the data.
Suggestion:
- If we were to refactor the code, we would place the logic of any alterations to the Model within the model itself, rather than within the Controller, as it is now.

Long Parameter List

This occurs when we have more than three or four parameters for a method. A long list of parameters might happen when several algorithms are being used in the same method. This may also results as a results of minimize coupling between objects to make them more independent of each other. For example, if the code for creating a particular object needed for a method was moved to the method for calling the method but the created objects are passed to the method as parameters. The original class no longer knows about the relationships between objects, hence reducing dependency. However, if more of these objects are created each will require its own parameters and a longer parameter list will be created.
For Timbits' project, they had some long parameter in their code. For example:

```
$newRequest = new request($reqID, $uID, $approverID, $analystID, $appID, $userName,
$userEmail, $num, $department,$appName, $permission, $reason, $approverName,
$analystApproved, $approverApproved, $permissionGranted, $isOpen, $date);
```

This can be found at /Controllers/openRequest.php from line 62 to 65.
Suggestion:
- A suggestion on how this can be avoided is by grouping them via *Introduce Parameter Object*. After analyzing the code, an argument can be made for this choice. These parameters are fields for their database but can still be presented better by encapsulating the parameters into an object and passing in this object. Example: $newRequest = new request($requestContents); where $requestContents holds all the parameters.

In addition, for security purposes, the line of code: $user = unserialize($_SESSION['user']); from Controllers/openRequest.php at line 20 is not advisable. According to php documentation, "Do not pass untrusted user input to unserialize() regardless of the options value of *allowed_classes*. Unserialization can result in malicious user may be able to exploit this code." In this case, the $_SESSION['user'] is contains gotten from user input.

In conclusion the Timbit's project code was well done. MVP1 of their project reflects the goals stated in the ATDD. However several of the ATDDs related to error notification were not completed. In the end, the code as designed enables other programmers to continue easily where they left off. The presence of a few code smells leave their code as written in a less than perfect state, as they have plenty of room to improve.

References:
Long Method, https://refactoring.guru/smells/long-method
Documentation, https://refactoring.guru/smells/comments
Long parameter list, https://refactoring.guru/smells/long-parameter-list
Data class, https://refactoring.guru/smells/data-class