**Detailed features:**

1. **Predict merge conflict with parent branch and send a notification to the user who created the local commit.**
   1. Whenever a user creates a local commit it will be gathered by our software and sent to our middleware which will compare this local commit with the current parent branch. If a merge request is possible then no warning is shown, on the other hand whenever a merge conflict is predicted by trying to merge the local commit with the remote branch then a notification is issued to the developer to alert that there are changes in the parent branch with the current status which might produce a merge conflict.

The user case: a person creates a local commit, this is fetched by our client software which communicates the patch details to our middleware and in the middleware we try to merge with the parent branch. If the merge is successful no notification is issued, if there is a merge conflict then the middleware issues a notification for the client to raise awareness about the current status of the software. As a follow up we could highlight the areas were a possible merge conflict could exist and notify this to the author of the local commit.

1. **Predict merge conflicts against pull requests done by the teammates.**
   1. Every time one of the users makes a local commit on their repository, a patch will be sent to the middleware containing all the changes. The idea is to use this code and predict merge conflicts with the code being pushed by the other teammates. This is don given it might occur that no merge conflicts are found on the master branch, however, there might be a merge conflict after somebody else changed the state of develop. It is necessary to implement an endpoint on the middleware which is connected to the repository. With this endpoint, information regarding the pull requests can be leveraged and compared against the user’s local commits.   
      The user case is: a person creates a local commit, sends it to the middleware and the middleware queries the repository for pending pull requests (status = open) and tries to merge the patch on that pull request code. If no conflicts are found, no alert will be displayed to the user. If an conflict is found a notification will be displayed to the user urging him to be alert on the specific pull request. Given that if the pull request goes first into the master branch, a merge conflict will occur when the user tries to push their code.
   2. Edge cases: It might happen that one of the PRs has a merge conflict with the master branch. At that point, the specific PR will not be considered for the merge conflict predictions.  
      If a conflict is found with one of the PRs, all PRs combined will also throw a conflict.  
      If there is no conflict with each PR, the possibility of a merge conflict with all the PRs combined still exists. The order of testing is: try each PR with the local commit, then combine every PR with the master branch and then test again against the local commit.
2. **Graph the local commits which are still not pushed to git:** 
   1. For this feature, we want to implement a graph displaying information which is currently not accessible using git or one of its multiple plugins. The novelty of the graph is to consider the whole team of people working on one repository and keep a constant track of their local commits. These local commits can be displayed in a tree showing how far ahead they are form the master HEAD. In the same graph, information about each teammate will be displayed. All of their local commits will be shown, and people will be able to see how far ahead their teammates are from the master branch. The father away a user is, the higher the probability a merge conflict will occur given there is much more changes being done to the code repository than just by one commit.   
      After the graph is done, a function to click on the points of the graph displaying the local commits will be implemented. Once a person clicks on the local commit, information will be displayed as to which are the files being changed, when was the commit made, the commit message, etc.   
      The user case is: a person can open a popup display screen where they will be presented with the graph of the teammates local commits. This person can also see their local commits.   
        
      To gather all the information we require from the commits, it is necessary to implement a listener for when a local commit is done. This can be achieved either by creating a alert on the system every time git is invoked or by creating a polling service on the client’s machine. The approach of polling will be done with Python and installed on every user’s local environment.   
        
      Additionally, it is necessary to leverage information about the project. For example, the project git’s url, which is the master branch which will always have the latest code (master or develop) and the name. Once the client install the plugin or service, they will be asked to provide the git’s url and the directory path where the git project can be found. By using those two pieces of information, every other part can be leveraged using git’s API or local information (patches).   
        
      A list of users will be kept for every project, giving us the possibility of deciding who is on each team and who should see the shared information.

Mockup features 1 & 2:

