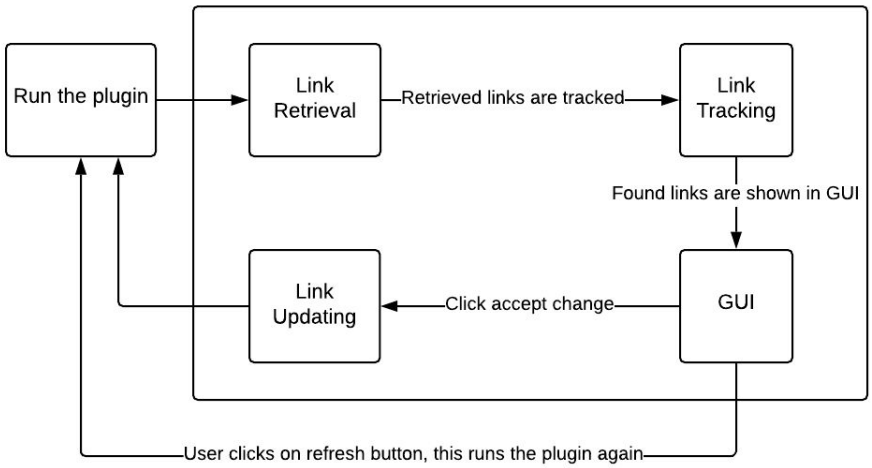
Higher level description of the implementation of the plugin

The plugin tackles the problem of links from markdown files that have not been maintained over time, by checking which links have gone out of date and in the case of them going out of date, suggesting new actions to the user for each link. In case of changed links, the actions would be either to delete the link or update it with a newer link, which is suggested by the plugin.

The plugin makes use of the git history of the project to a great extent for tracking links that correspond to the project open in the IDE.

Below is attached a ***diagram*** that describes the main logic of the plugin.



The first task that the plugin executes is retrieving all the links that appear in the markdown files in the currently open project. After they are retrieved, they are categorized into specific link object depending on their link path formats.

From that moment, each link object makes use of a change tracker service class implementation, that defines the environmental-dependent logic of retrieving changes of elements referenced by links. Each link object knows which method needs to be called from the change tracker service (and also knows with which parameters to call these methods).

Relative links and web-links that correspond to the currently open project will be tracked using the git history of the project. Otherwise, the plugin will resort to the use of the platform specific APIs on which the code is hosted.

Links that correspond to the open project are tracked in the following ways:

1. **Links to files**

Files are being tracked in the following way:

- First, the plugin checks whether this file is untracked -> if so, return immediately
- Then, the plugin will retrieve all the changes that have affected this **file** (I'm saying file and not full path), and then it will check whether the path exists in any of these changes. If the path exists, then start traversing through the changes starting from that path until the last change from that path. The last change will represent the final change that has affected that path, and it can be either a deletion, move.
- Then, using this new found path (in case there is one), we want to check whether there are any working tree changes that might have affected this new path. If there exist working tree changes, use that change instead. Otherwise, use the change found from going through the Git history
- While performing the traversal, we're storing every commit and path at this commit of the file we are tracking. Using these, we can then track line(s) (by performing diffs between the version of the file at the before commit and the before path and the version of the file at the after commit and after path, but more details will be presented at point #3.)

2. **Links to directories**

Using data gathered from version control systems, the plugin can easily find the changes that have been applied to a file over time. More specifically, the plugin would be interested of the set of changes that have happened between the version of the project at the time the link was created or the time the link was last updated and the current version of the project. Using this data, we can determine the current status of the file that was originally referenced by the link upon creation. The file can come up as having the same path, but having been modified, having another path and having been renamed or having been deleted. Finally, we can create the new link based on the current status of the file.

Due to the fact that Git works by tracking individual files and not directories as a whole throughout the history of a project, there is no way the plugin can retrieve data from Git in a such direct-fashion as for files.
Therefore, for directories, the plugin will proceed in the following manner:

- It will first get from the VCS the directory contents (that is, all the files and sub-directories in that directory) at the moment the link to the directory was created/last updated.

- It will then track, one at a time, each file inside of the directory from the moment the link was created/last updated to the current time.
- If there is at least one file that remained in the current directory, then do nothing, the directory still exists. However, if all of the files seem to have been moved, we need to check if there is a majority of the files that appear to be moved altogether in another directory. If so, then it can be said that the directory has moved to this new directory.

3. Links to line(s)

The plugin will first try to fetch the “start” commit of the line containing the link in the markdown file (e.g. the commit at which that line was created) and try to get the contents of the line(s) in the file at that commit. It will then fetch the history of the file in which the line(s) resides and it will perform a git diff commit-by-commit (taking each consecutive pair of links). It will also do a git diff with the working version of the file, in case that file appears to be modified in the working tree.
Using the git diff outputs, it will collect the deleted lines and added lines, along with the context lines for each of these lines. After that, these lines will be passed to an implementation of the LHDiff algorithm, which will map a deleted lines(s) to an added line(s) with pretty good accuracy. The LHDiff implementation will then output the status of that /those line(s) (e.g. deleted / moved) along with the new mapped lines (which contain the new contents and new line numbers). From these, the plugin is able to generate new path(s).

After all the changes have been gathered, the plugin will show to the user (in the GUI) the links in a tool window, categorized into 3 classes:

- **Changed links:** those links that reference an element that has been deleted / had it’s location moved and require action. The user will have to option updating such links from the UI. In the case of a move change, the user will be able to update the link to a new link having a new link path / commit SHA (for web-links that are permalinks).

- **Unchanged links:** those links that reference an element that has not been changed over time (aka it still exists and it points to the same element as when the link was initially created). No option to update the links is given to the user.

- **Invalid links:** these links were never valid or there was some error that occurred while trying to get their changes. No option to update them is given to the user. An error message is shown to the user for these links.