Our team used GitHub for managing the project source and a combination of text messages, Facebook messages, and face-to-face talking for communication.

Github was perfect for managing the source. We used pull requests for nearly every commit, as it was extremely useful in checking each other's code to verify that the code we were committing was correct. Even though most of the time the code did not need many (if any) modifications, the fact that there was a checking system in place ensured better code in the long run and peace of mind for each of us that at any time, master was in the best state it could be in at that point in time.

There was one interesting issue that arose with pull requests; after a pull request was submitted, either of the two other group members could review it. However, if Ian reviewed and approved the request, he would merge it without commenting, but Elliot and Cassius would say "LGTM" (looks good to me) and leave it to the requester to merge the changes. This discrepancy lead to some confusion. For instance, if the changes were merged without any comment, it was assumed that they were approved, but it would have been nice to have some concrete evidence of that. But if there was an approval comment left without merging, then the tree was not as up-to-date as it could have been, because if the changes were approved, there is no reason why the changes should not have been made available for everybody to use immediately. We eventually solved this inconsistency by agreeing that the reviewer, if he approves of the changes, should leave a comment saying so, and then merge the changes himself. This way provides the benefits of both of the previous solutions.

When each of us were working on our code, we had our own branches. Using this methodology, we could each change as much as we needed to, or experiment as much as we wanted to, without worrying about breaking the master branch for any of the other coders. Each of our features were in isolated units; if somebody had changes he wanted to commit to master, he could commit those changes to his branch and the person reviewing those changes would only have to see those specific changes, not the changes that everybody else was working on as well. Then, when the code was merged, everybody knew that the only features and code that was being merged was equal to one unit of development.

Other Github features, such as the issues tracker, allowed us to manage bugs that needed to be fixed, and we could see who took ownership of the bug and was responsible for fixing it. While bug trackers can often be used for tracking anything and everything that needs to be done, we only used the issue tracker for bugs. For feature enhancements or additions, we mentioned those in our message threads. Messages are by no means a scalable solution for this problem, but we found it easiest for this instance.

For communication, our team used a combination of text messages, Facebook messages, and in-person talking. We met weekly on Fridays (with the exception of Thanksgiving weekend) to plan out the project and discuss progress, and otherwise talked over text and Facebook messages when not together. There were also a few times after class when we would stay around to discuss the project and assign tasks to each person. For the digital communication, there was generally no reason for using one over the other, with the exception of one night when Elliot was working on the project in the basement of the library where there was very poor cell reception, in which case Facebook Messenger was the sole communication choice. Facebook Messenger was generally the nicer option because it displays who has read which messages and that somebody is typing a response, but both forms of communication were used with relatively the same frequency.