The first major refactor I did was create a helper method for the addWord method, which allowed an easier understanding of the logic being handled inside of the major if statement of addWord. I have noticed that the method was a bit longer than necessary and that the logic for adding branches could easily be moved into a helper method, and so I created a private helper method called traverseAndAdd that handles this logic. The helper method is self-descriptive and easy to understand, which allows addWord to feel less complicated.

Before:

```
void Trie::addWord(string word)
{
    if (word.size() != 0)
    {
       char first = word[0];
       string newString = word.substr(1);
       /// If this branch already exists, we do not want to replace
       /// it with a new one, and so we skip it.
       if (branching.count(first))
       {
           branching[first].addWord(newString);
       }
       else
       {
           Trie temp;
           temp.addWord(newString);
           /// Set this branch in the current Trie to the new branch we
           /// created.
           branching[first] = temp;
       }
    }
    else
    {
       isAWord = true;
    }
}
```

After:

```
void Trie::addWord(string word)
{
    if (word.size() != 0)
    {
        char first = word[0];
        string newString = word.substr(1);
        traverseAndAdd(first, newString);
    }
    else
    {
        isAWord = true;
    }
}
```

The second major refactoring that I did was make a helper method for allWordsStartingWithPrefix called getAllWordsOfPrefix. When I refactored my code to work with a map instead of an array of Nodes, this caused my code in allWordsStartingWithPrefix to become more complicated, since initially my code relied on setting a temporary Trie to “this”, which I can’t do in the refactor with the code now using the actual object of Trie rather than a pointer to it. To compensate for the complication, I moved all of the logic responsible for tree traversal and adding words to a vector into a private helper method, which allowed me to make the code easier to read and understand.

Before:

```
vector<string> Trie::allWordsStartingWithPrefix(string prefix)
{
    vector<string> allWords;
    /// Checks if the prefix is valid. If not, returns an empty vector.
    if (!checkIfValid(prefix))
   {
      return allWords;
   }
   /// If the prefix is a word, it should be added to the vector.
   if (isWord(prefix))
   {
      allWords.push_back(prefix);
   }
   /// Traverses down to the last character of the prefix and calls
   /// the checkAllBranches method on the node of the last character
   /// once it is reached.
   if (branching.count(prefix[0]))
   {
        Trie temp = branching[prefix[0]];
        if (prefix.size() == 1)
        {
            temp.checkAllBranches(allWords, prefix);
        }
        else
        {
            for (int i = 1; i < prefix.size(); i++)
            {
                char current = prefix[i];
                if (i == prefix.size() - 1 && temp.branching.count(current))
                {
                    temp.branching[current].checkAllBranches(allWords, prefix);
                }
                temp = temp.branching[current];
            }
        }
    }
    else
    {
        checkAllBranches(allWords, prefix);
```

```
    }
    return allWords;
}
```

After:

```
vector<string> Trie::allWordsStartingWithPrefix(string prefix)
{
   vector<string> allWords;
   /// Checks if the prefix is valid. If not, returns an empty vector.
   if (!checkIfValid(prefix))
   {
      return allWords;
   }
   /// If the prefix is a word, it should be added to the vector.
   if (isWord(prefix))
   {
      allWords.push_back(prefix);
   }
   /// If the prefix isn't empty, traverses down to the last character
   /// of the prefix and calls the checkAllBranches method on the node
   /// of the last character once it is reached.
   if (branching.count(prefix[0]))
   {
      getAllWordsOfPrefix(allWords, prefix);
   }
   /// If the prefix is empty, adds all existing words in the tree to the
   /// vector.
   else
   {
      checkAllBranches(allWords, prefix);
   }
   return allWords;
}
```