My Academic Plan

- **Step 1. Understanding the Problem ✓**
  - Determining the detailed structural similarities and differences between source code fragments is a complex task
  - It can be applied to solve several source code analysis problems
  - As a specific application, our focus is on the study of where logging is used in the source code
  - logging is a pervasive practice and has various applications in software development and maintenance
  - It is a challenging task for developers to understand how to use logging calls in the source code
  - In this research, we would like to understand where developers log in practice, in a detailed way
- **Step 2. Proposing a Solution to the Problem ✓**
  - Develop an automated approach to detect the detailed structural similarities and differences of Java methods containing logging calls
  - Our solution would:
    - classify logged Java methods into groups using a measure of similarity such that entities in each group has maximum similarity with each other and minimum similarity to other ones
    - construct a structural generalization of each group that represent the detailed structural similarities and differences of all logged Java methods in the group
- **Step 3. Gathering Background Information ✓**
  - Abstract Syntax Tree (AST), which is the basic structure we will use for describing software source code
  - First-order anti-unification, which is a technique used to construct generalizations
  - Higher-order anti-unification modulo theories (HOAUMT), which is used to address the limitations of first-order anti-unification
  - The Jigsaw framework, an existing tool for a subset of HOAUMT, which we extend to determine potential correspondences between ASTs of logged Java methods
  - Agglomerative hierarchical clustering, which is a clustering algorithm used for classifying logged Java methods into groups using a measure of similarity
- **Step 4. Methodology ✛**[12 DAYS TO COMPLETE]
- **Step 4.1 Constructing an anti-unifier (structural generalization) from two given logged Java methods with a special attention to logging calls ✓**

1. Developing an Algorithm that:
   * maps the source code of two logged Java methods to AST structures via the Eclipse JDT framework ✓
   * creates an extension of AST structures, called AUAST, to allow the application of higher-order anti-unification modulo theories ✓
   * determines potential candidate structural correspondences between AUAST nodes using the Jigsaw framework ✓
   * applies some constraints to prevent the anti-unification of logging calls with anythisng else ✓
   * uses a greedy selection algorithm to approximate the best anti-unifier to our problem by determining the best correspondence for each node to handle the problem of having multiple potential anti-unifiers
   * Develops a measure of similarity between the two AUASTs ✓
2. Implementing our approach as an Eclipse plug-in ✓
3. Evaluating our approach and tool by conducting an experiment on 10 sample logged Java methods, as a test set ✓

- **Step 4.2 Anti-unifying a set of AUASTs of logged Java methods ✛**[2 DAYS]

  1. Developing a modified version of a hierarchical agglomerative clustering algorithm suited to our application ✓
  2. Implementing our approach as an Eclipse plug-in ✓
  3. Evaluating our approach and tool by conducting an experiment on the test set ✛[2 DAYS TO COMPLETE]

- **Step 4.3. Characterizing logging practices ✗**[10 DAYS]

  - Applying our tool on the source code of three open-source software systems that make use of logging ✗
  - Extracting all logged Java methods from these systems to construct the structural generalizations ✗
  - Representing the results and contributions ✗

- **Step 5. Writing the thesis ✛**[22 DAYS]

  - Writing chapters of thesis, including:
    * CHAPTER 1: Introduces the problem and why it matters, outlines the idea of our proposed solution, and some key related work to argue that the problem has not been solved and the idea of solution is novel ✛[3 DAYS TO COMPLETE]
    * CHAPTER 2: motivates the problem of understanding where to use logging calls in the source code through an example ✛[2 DAYS TO COMPLETE]
    * CHAPTER 3: provides the background information ✓

∗ Chapter 4: describes our proposed approach and its implementation as an Eclipse plug-in ✓
∗ Chapter 5: presents an empirical study conducted to evaluate our approach and its application to characterize logging usage ✢[5 days to complete]
∗ Chapter 6: discusses the results and findings of my work, threats to its validity, and the remaining issues ✢[3 days to complete]
∗ Chapter 7: describes related work to our research problem and how it does not adequately address the problem ✓
∗ Chapter 8: concludes the dissertation, presents the contributions of our study, and future work ✢[2 days to complete]
∗ Corrections and Revisions ✢[7 days to complete]