Introduction
The best projects, regardless of discipline, are well thought out and planned prior to execution. This document lays out a planning strategy to be used by the group when planning projects in Java and C++. This framework is a hybrid of our current and the eXtreme Programming (XP; www.extremeprogramming.org) techniques.

The XP methodology has some ideas that are beneficial to an educational environment. Principally, XP promotes knowledge sharing by moving people around from task to task. This has pro’s and cons. By moving people around and having more than one person centered on a task, you are ensuring that there is knowledge transfer, and there is not just one seat of knowledge for a code block – there is knowledge redundancy. However, it hampers exploitation of a specialist in the team. A database specialist may be forced to work on GUI design to expand their knowledge. Although this is not necessarily a bad thing for all involved, it can result in a net decrease in productivity. With those issues in mind, however, I feel it is necessary to experiment with new ideas, and to expose the students within our group to new methods of coding.

This methodology will only be used with larger projects for the short term, to gauge its success within our environment. If it proves valuable, it may become the de facto standard within the group.

Why Change Our Current Methods?
Due to the close working relationship we maintain with our user-base, it is possible for us to communicate needs from the users to the developers in real-time. The ability to incorporate user comments and suggestions early in the development process is a great advantage for us, and leads to more useful products in the end. However, this comes at the cost of developer frustration from constantly evolving user ideas and requirements. As addressed in the Mythical Man-Month, there needs to be a balance between requirements set by the software developer and the end-users. In the past we had tried to
meet this balance by developing down one path, and changing mid-stream. This is not only inefficient, but quite frustrating for all parties involved. The XP framework, designed for environments with constantly changing user-requirements, may be one method by which we can overcome many of our development frustrations, while maximizing our efficiency.

One major drawback to the XP method is that it does not require much documentation throughout the process, instead relying upon the developer’s memory. To combat this, developers will be required to document certain parts of the project’s development, such as the use-cases, tasks, and task deadlines, to keep better track of where the project needs to go.

A second drawback to the XP methodology is that it does not allow for a strict deadline to be set. It should be noted that strict deadlines rarely work in software development environments, however, in an attempt to keep things within reasonable time frames, deadlines will be created on a per-task basis. An estimated deadline for the project completion will also be set, and will be allowed to move within reason, based on additional requirements. As additional requirements are formed, they will be documented, as will the reason for the change in deadline.

**Software Cycle**

Our previous software cycle (SC) has been altered to reflect our implementation of XP. As it stands, the new SC is a combination of our old methods with selected parts of the XP framework.

1. Project idea developed by someone within the lab or within the team
2. Project idea is brought to coordinator for initial planning, analysis of priority, assignment of assets, etc…
3. Release Planning:
   a. Assets meet with coordinator and a representative of the users to discuss needs
   b. Use-case diagrams are generated based on user stories
   c. Use-cases are designed into tasks and assigned realistic number of programming days to complete
   d. Tasks are ranked according to priority and assigned to assets for the first iteration cycle (1-3 weeks, to be decided at the planning meeting)
   e. Task scheduling added to the Microsoft Project file
   f. Tactics to accomplish tasks discussed
      i. Objects for each task are assigned to a card
      ii. The cards are arranged on the table to model the situation
      iii. Iterate through different arrangements until the best one is found
      iv. Go with it!
   g. Methods for testing are discussed, including test scenarios suggested by users that can be simulated on our side
   h. Assets assigned to design UML for that portion
4. Iteration Meeting:
a. Progress on tasks discussed  
b. Bugs from prior releases that have been noted need to be fixed and prioritized  
c. Any tasks that failed testing and have not been fixed prior to the meeting will be continued at this point  
i. If a person/team working on a task has not been able to rectify the testing issue, a change in assignment or addition of another asset may be required at this point  
d. New tasks added according to rank for next iteration  
e. Tactics to accomplish new tasks based on past experience and prior portions of the development  
i. Use the objects already designed whenever possible  
f. Assign deadlines for tasks from this meeting  

5. Code development  
a. Refactor code  
b. Communicate needs to coordinator so that asset (re)assignment can be accomplished as needed  

6. Code is reviewed by Bioinformatics Group  
7. Assets make changes and revisions as requested by the group to the code, and resend code for review and testing  
8. Following code approval by the group, the code is beta tested by a user outside of the group  
9. Comments from beta test are incorporated by the assets into the code  
10. Code is retested by beta tester  
11. Following approval from beta tester, code is made available to the lab  
12. User comments are assessed for possible inclusion in future releases of the software
UML Models and Other Supporting Documentation

Class interaction diagrams (UML Models) and use cases are required of all object oriented projects. UML Models are developed as a result of discussion from the Iteration Meeting. Other supporting documentation may also be submitted to the coordinator and the group for evaluation of the project plan. These include:

1. Sequence or Flow Diagrams
2. Documentation/Source from outside APIs and libraries to be used in execution of the project
3. Documentation/Source from similar projects ongoing within the group

Development Order

Although developers are generally unconstrained in how they develop their project, the one unending requirement is that an application will always be model-driven, not GUI-driven. Developers must develop the GUI last – if they require a GUI, a simple one should be made to avoid putting business logic into the View. The Model must be completed first, then development of the View and Controllers can take place. There are very few exceptions to this rule.

Pair Coding

At times it may be necessary to experiment with the notion of pair coding. This is a technique where two people sit at the same monitor, share the same keyboard and mouse, and code the same thing together. This has worked in other environments, and has lead to an overall time savings. This option may be explored as necessary.
**Code Testing**

All production-level code is required to be tested often during the development process. The lab standard for code testing will be use of the XUnit.org testing environments (i.e., for Java that is JUnit). Tests should be written before the code is written (i.e., development planning stages). Code testing is something that should occur during the development process, not at the end of the project.

There will be some code that is just not testable using JUnit. Under these circumstances alternative means of testing should be completed, such as debugger values, use of test-values within the database, or use of test tables.

**Code Refactoring**

A major part of the XP movement is code refactoring. That is, revisiting code from time-to-time to ensure it is as simple and efficient as possible. The more code refactoring is done prior to code review, the less code downtime there will be. This translates into time savings and getting the product out to the users faster.

**Code Review**

All code is subject to review for quality control and assurance. No code may leave the lab without having first been reviewed. Code should be reviewed within 1 week of completion, unless other arrangements have been made. Code review also serves as an educational experience for coders within the lab, as others can learn from techniques developed by others.

**Reference Material**

The lab provides reference material for the use of group members. It is expected that lab members use these materials when possible, especially the Java Tuning book! This book has many useful tricks and techniques for efficiency coding, and is an excellent software engineering guide.