Table of Contents

Purpose .................................................................................................................. 4
Parties Subject To the SOPs and SOs ................................................................. 4
Regulated Materials ............................................................................................ 4
Quality Assurance Plan ....................................................................................... 6
  Application Planning Quality Assurance Plan (APQAP) .................................. 6
  Business Rules Document ............................................................................... 8
  Use Case Document ....................................................................................... 8
  In-Progress QA Activities ................................................................................ 8
  Progress Reports ............................................................................................. 9
  Final Reports .................................................................................................. 9
  Class Versioning ............................................................................................. 9
Quality Assurance Plan (Scripts and SQL) ......................................................... 10
  Scripts and SQL Planning Quality Assurance Plan (SS-PQAP) ................. 10
Object Oriented Language Software Coding Standard ..................................... 12
  Packages ........................................................................................................ 12
  Programmatic Classes .................................................................................... 12
  Naming Conventions ..................................................................................... 13
  Typographical Conventions .......................................................................... 13
  Use of Get/Set Methods ............................................................................... 14
  Method Writing .............................................................................................. 14
  Writing Constructors ...................................................................................... 14
  Miscellaneous Notes ..................................................................................... 14
Procedural Language Software Coding Standard ............................................ 15
  Scripts and Application Files ......................................................................... 15
  Naming Conventions ..................................................................................... 15
  Typographical Conventions .......................................................................... 16
  Method/Subroutine/Function Writing ........................................................... 16
  Miscellaneous Notes ..................................................................................... 16
SQL Coding Standard ....................................................................................... 17
  SQL Script Files ............................................................................................ 17
  Naming Conventions ..................................................................................... 17
  Typographical Conventions .......................................................................... 17
  Miscellaneous Notes ..................................................................................... 18
SAS Coding Standard ....................................................................................... 19
  SAS Files ........................................................................................................ 19
  Naming Conventions ..................................................................................... 19
  Typographical Conventions .......................................................................... 20
  Miscellaneous Notes ..................................................................................... 20
Testing and Validation ..................................................................................... 21
Peer Review ......................................................................................................... 23
  Code Review Package ................................................................................... 23
  Code Review Elements .................................................................................. 23
Database Development ..................................................................................... 24
  Database Schema Testing ............................................................................ 24
Purpose

This document will provide the Standard Operating Procedures (SOPs) and Standing Orders (SOs) that govern all documents, code, documentation, and other miscellanea produced by the Zacharewski Bioinformatics Group. This is a living reference document, and will be updated periodically.

By formalizing the SOPs and SOs we begin the process of formulating our Overall Quality Assurance Plan (OQAP), a new requirement for EPA funded projects. Although this document outlines many of the aspects of the OQAP, it is no substitute for a formal OQAP which is to follow.

The SOPs and SOs within this document provide project members with the formalized structures and procedures to complete a project. The goal is to produce high quality standardized, readable code for use within the laboratory, and disclosure to the University. The planning documents dictated here provide the project team an opportunity to critically analyze their project plan prior to execution. This also gives other members of the Bioinformatics Group an idea of the other projects that are being undertaken, and provides a formal mechanism for them to provide input into projects. Many of the processes dictated within this document already occur, but through a non-formal mechanism. This serves to formalize the process, and satisfy requirements from the University’s Office of Intellectual Property and our funding agencies.

Parties Subject To the SOPs and SOs

All parties that work for the Zacharewski Laboratory, directly or indirectly, that provide some form of computational assistance to the members of the laboratory are required to adhere to the guidelines found within this document. This includes individuals who generate code solely for the purposes of their own projects.

Regulated Materials

Regulated code, documents, documentation, and other miscellanea include all of the following:

1) Code from Java, C++, C, Visual Basic, C#, Perl, SQL, SAS and R
2) Database schematics and designs
3) Database architectural and tuning changes
4) Preliminary documents
   a. System and detailed designs
   b. Testing schemes
   c. Project descriptions
   d. Project organization and responsibilities
   e. Validation and verification plans
   f. Change control and configuration management plans
   g. Audit and review documents
   h. User support documents
5) In-progress documents
   a. QA reports
   b. Updated preliminary documents
c. Progress Reports
6) Completion and Post-completion documents
   a. System documentation
   b. User documentation
7) Any other code as specified by the PI
Quality Assurance Plan

Application Planning Quality Assurance Plan (APQAP)

Prior to the execution of any coding of a regulated nature, the software developers assigned to
the project must produce a detailed preliminary document for approval by the lead
bioinformatician. The sections of this document will be:

1) Section 1: Approval by Project Participants
   a. The project lead is responsible for obtaining the signatures of all project
      personnel on the signature page.
   b. This will document all of the participant’s agreement to the project objectives,
      the testing, validation and evaluation methods, as well as their agreement to
      perform their assigned tasks according to the project plan

2) Section 2: Project Description
   a. This section will have the following subsections
      i. Subsection A: Overview of the project
      ii. Subsection B: Intended uses
      iii. Subsection C: Planning documents
          1. Business rules
          2. Use-case scenarios
          3. UML Diagrams
          4. Other documents as necessary
      iv. Subsection C: Quality objectives
      v. Subsection D: Schedules and milestones
          1. The schedule will include code walk-throughs by the
             Bioinformatics Group
      vi. Subsection E: Hardware and operating systems that the end-product
          will operate under

3) Section 3: Project Organization and Responsibilities
   a. This section lists all of the project personnel, their responsibilities on the
      project
      i. Describe any training that will be required for project personnel
      ii. A Traceability Matrix must be produced that ties together each
          member of the project to their respective project end item (PEI)
   b. This section must also list the QA personnel who will be involved in testing,
      verification, and code review

4) Section 4: Functional Requirements
   a. This section must list all of the functions that the software must
      perform/address
   b. Each function, or PEI, must be assigned to a member of the project in Section
      3 using a Traceability Matrix
   c. If this software is used to generate data, this section should also briefly outline
      any quality monitoring it performs to ensure the data being produced are
      accurate

5) Section 5: System Design
   a. This section provides the detailed inner workings of all of the algorithms
b. An executive summary of the section must be provided
c. Justifications should be given for the use of controversial algorithms
d. Use-case, data-flow and other similar diagrams may be helpful at this stage

6) Section 6: Implementation
   a. Cite this document in this section as providing the standard operating procedures for the project
   b. If other SOPs are adopted in addition to those listed here, they must be detailed in this section
   c. Any internal checks that are performed during development should also be described in this section

7) Section 7: Testing
   a. Describe the complete testing scheme in this section
   b. The section should be broken down by each test, and include a discussion of the purpose of the test, what a positive and negative result would look like, and how the test is performed
   c. If an alternative test could be employed instead of your test, and you do not perform all of the alternative tests, give a justification for why they were not performed

8) Section 8: Data Validation and Verification
   a. Describe in detail the complete method for validating the data being produced by the algorithm, including the source of the validation data
   b. Describe the method for verifying the methods

9) Section 9: Change Control and Configuration Management
   a. Describe the procedures used for controlling and documenting all significant changes made to the code

10) Section 10: Audits and Reviews
    a. Describe all of the quality assessment methods that are planned to monitor:
       i. Performance
       ii. Data quality
    b. Describe the code review process
    c. Describe the alpha and beta testing strategies that will be employed

11) Section 11: System Documentation
    a. Describe how user documentation will be disseminated
    b. Describe the review process

12) Section 12: Handoff and Transition
    a. Describe the handover and transition process
       i. Soft handoff: transition from one team to another where both have overlapping time schedules to permit training of the second team by the first
       ii. Hard handoff: transition from one team to another where the teams do not overlap in schedule, making the training opportunity impossible

This preliminary document will first exist as a proposal where it will be discussed at a meeting of the entire Bioinformatics Group. The project team will formally present the document at this meeting, and discussion will ensue, and recommendations will be made. Once the recommendations have been incorporated into the document, the project members will
reintroduce the proposal to the Bioinformatics Group. This process will continue until final approval of the plan has been granted.

Throughout project progression, the team is responsible for ensuring this document is updated as necessary. The APQAP document must be amended as stipulated within this document. It is not expected that this document will accurately reflect the final product at first inception; however, any changes in the project execution plan must be documented.

**Business Rules Document**

The Business Rules Document is a planning document described in Section 2, Subsection C of the above APQAP. This document must list all of the rules governing the project requirements. These rules form the foundation or the basis of all the requirements. Each business rule must be issued a unique identifier starting with a BR prefix.

<table>
<thead>
<tr>
<th>Identifier:</th>
<th>BR1</th>
<th>Rule: Only a Data Manager shall create a new user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier:</td>
<td>BR2</td>
<td>Rule: A User shall create data records within the database</td>
</tr>
<tr>
<td>Identifier:</td>
<td>BR3</td>
<td>Rule: A User shall not create other users</td>
</tr>
</tbody>
</table>

**Use Case Document**

The Use Case Document is a planning document described in Section 2, Subsection C of the above APQAP. This document shall consist of a cover page, an introductory section, a section describing the actors, a section describing the use cases which also includes the sequence diagrams, and any miscellaneous other sections as required. The template in Appendix E must be followed for construction of the Use Case Document.

**In-Progress QA Activities**

Once a project plan has been finalized and approved, the project team will begin work on said project. Throughout the development of the application code, it will be documented with in-line documentation and any language specific documentation tags. Any supplemental documentation to facilitate understanding of the code must also be updated throughout development. It is appropriate to make sure all code is documented on a daily basis. All code must be logged into CVS on a nightly basis.

The project team is responsible for ensuring all of the planned QA activities occur on schedule. They are also responsible for ensuring their code conform to all applicable standards.

Throughout project execution, it is expected that amendments to the original plan will be necessary. The project team is expected to propose their amendments to the original APQAP, and present the amended document to the lead bioinformatician. The lead may either approve or disapprove of the proposed amendments. In some instances, the amendments should be presented to the Bioinformatics Group as a whole for further discussion prior to implementation. The lead bioinformatician will convene a meeting of the Group when necessary.
**Progress Reports**
The project team is responsible for ensuring progress reports are filed promptly, if/when required. Progress reports should detail the progress made to date, and include a section detailing what progress has been made since the last report. If any quality assurance issues have been identified during the reporting period, the report must detail what was found, and how it was rectified. All progress reports must also include a section discussing the testing and validation methods used. See the section on Testing and Validation for more information.

**Final Reports**
All projects will have a Final or Close-Out report to summarize the project development process. These reports will contain details of any major quality issues encountered during development, how they were rectified, and comments on how to improve overall project management for the future. All reports will include a Testing and Validation Matrix (TVM; see the section on Testing and Validation). These reports will also include a copy of the final user manual. The report will be signed by all project participants, and will be vetted for quality assurance purposes. Final approval of the Final Report does not mean that the project will not be re-opened for maintenance or refactoring at a later time.

**Class Versioning**
All classes will contain a version number. Once a class has been completed, that version is 1.0. Once a class undergoes minor revision, the revision must be noted at the place of revision, and a notation made in the header. Version numbers are such that the first number is a major version change, while the second number (the one following the period) represents a minor version change. It is at the authors’ discretion as to when to increment the major or minor version number. Class versioning should not be confused with application versioning.
Quality Assurance Plan (Scripts and SQL)

Scripts and SQL Planning Quality Assurance Plan (SS-PQAP)

Scripts are defined as small applications that do not include a GUI and perform a very simple procedure. By their very nature, scripts are not meant to take a long time to code. Although they are relatively simple, a script will often have very significant impacts on research results and outcomes. For this reason, script development is still regulated.

All code other than scripts is required to use the APQAP for quality assurance purposes. The trigger mechanism for defining a script as an application for these purposes is the complexity of the problem being addressed. The complexity is defined as the sum of the point values associated with each of these activities:

1) Reading file = 0.5 pts
2) Writing file = 0.5 pts
3) Parse input file = 0.5 pts
4) Calculation based on parsed data = 0.5 pts
5) Database call = 0.5 pts
6) Database insertion = 2.0 pts

If the complexity value is greater than 2.0 pts, the code will be governed by the APQAP.

Most SQL scripts that are generated do not require SS-PQAP development. The trigger for SQL is whether or not the query will have a significant impact on a research process. The following activities are automatic triggers for the SS-PQAP requirement:

1) Any query used for quality assurance purposes
2) Any query with research implications including:
   a. Any query that generates data to test or otherwise confirm or reject a hypothesis
   b. Any query that will generate data that will impact a hypothesis or cause generation of a hypothesis
   c. Any query that will impact decision making processes

Prior to the execution of any script coding of a regulated nature, the software developers assigned to the project must produce a detailed preliminary document for approval by the lead bioinformatician. The sections of this document will be:

1) Section 1: Project Description
   i. The goal of the script must be described
   ii. The user pool must be defined
   iii. The method by which users will interact with the script must be defined

2) Section 2: Functional Requirements
   a. This section must list all of the functions that the software must perform/address
b. If this software is used to generate data, this section should also briefly outline any quality monitoring it performs to ensure the data being produced are accurate.

3) Section 3: Implementation
   a. Cite this document in this section as providing the standard operating procedures for the project.
   b. If other SOPs are adopted in addition to those listed here, they must be detailed in this section.
   c. Any internal checks that are performed during development should also be described in this section.

4) Section 4: Testing
   a. Describe the complete testing scheme in this section.
   b. The section should be broken down by each test, and include a discussion of the purpose of the test, what a positive and negative result would look like, and how the test is performed.
   c. If an alternative test could be employed instead of your test, and you do not perform all of the alternative tests, give a justification for why they were not performed.

5) Section 5: Data Validation and Verification
   a. Describe in detail the complete method for validating the data being produced by the algorithm, including the source of the validation data.
   b. Describe the method for verifying the methods.

6) Section 6: System Documentation
   a. Describe how user documentation will be disseminated.
   b. Describe the review process.

7) Section 7: Handoff and Transition
   a. Describe the handover and transition process.
      i. Soft handoff: transition from one team to another where both have overlapping time schedules to permit training of the second team by the first.
      ii. Hard handoff: transition from one team to another where the teams do not overlap in schedule, making the training opportunity impossible.

This preliminary document will first exist as a proposal where it will be discussed at a meeting of the entire Bioinformatics Group. The project team will formally present the document at this meeting, and discussion will ensue, and recommendations will be made. Once the recommendations have been incorporated into the document, the project members will reintroduce the proposal to the Bioinformatics Group. This process will continue until final approval of the plan has been granted.

Throughout project progression, the team is responsible for ensuring this document is updated as necessary. The SS-PQAP document must be amended as stipulated within this document. It is not expected that this document will accurately reflect the final product at first inception; however, any changes in the project execution plan must be documented.
Object Oriented Language Software Coding Standard

The following software coding standard will be followed for all regulated object oriented languages (excludes all R versions and Perl version 5 and lower). The standard specifies 1) what constitutes a package, 2) the existence of a header on all code, 3) the existence of a body on all code, 4) the acceptable typographical format used for control loops and brace placement, and 5) object and primitive names.

Packages

All software written within Bioinformatics Group that is of an object oriented nature will use packages. Packages will encapsulate classes of a similar functionality in projects, OR they will encapsulate whole functioning units only if they are a part of a library. All packages must have a index.html file included within them to specify their function.

Programmatic Classes

Each class will reside within its own file, unless it is a Java Swing enabled class, where an inner class would be appropriate (e.g., ActionListener). Each class will have a header with the following structure:

```plaintext
/**********************************************
/* Class Name: xyz        */
/* Date: MM/DD/YYYY        */
/* Author: Firstname Lastname */
/* Versions:               */
/* Version 1.0  Lyle D. Burgoon MM/DD/YYYY */
/* Version 1.1  John R. Doe MM/DD/YYYY      */
/* Purpose: xyz            */
/* Algorithm: xyz          */
/* Notes: miscellaneous information */
/* Debug: information for debug purposes  */
/* Copyright YYYY Michigan State University Board of */
/* Trustees                   */
/* All rights reserved       */
.fullNameSpace separator
ảy

```

Immediately following the header will be the package statement. Immediately following the package statement will be the import statements.

All code will be documented by both in-line documentation. In-line documentation is required inside methods and constructors, and should occur on the same line as the code. Try to align all of the in-line documentation within a block area so that they start at the same place.
**Naming Conventions**

When you name anything in your code, make sure the name has an obvious meaning. Naming your variables, classes, methods, etc, things that only are of interest to you is of little use to others who are reading your code, and will make your code less readable. The fastest way to generate a QA infraction is to make your code unreadable by others. Furthermore, do not give your variables the same name as those within its superclass.

**Classes:** All class names will begin with a capital letter. If the class name is a compound name made up of more than one word, such as TrafficDirector, all words in the name will be capitalized.

**Methods:** All method names will begin with a lowercase letter. If the method name is a compound name made up of more than one word, such as getLocusLinkIdentifier(), the first word will begin with a lowercase letter, while each subsequent word will begin with a capitalized letter.

**Objects:** Object names will begin with a lowercase letter to denote the type of object that it is. The following table dictates the controlled vocabulary codes for Java objects. For user generated objects, the object name will begin with the first letters of each word in the class name, in lowercase, as a prefix directly attached to the object name, where the first and subsequent words begin with capital letters. For example, if I created a new TrafficDirector object, it would be called tdObjectName. The only exception to the prefix rule is the String class, which follows the rules for primitives.

<table>
<thead>
<tr>
<th>Java Object</th>
<th>Prefix Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayList</td>
<td>al</td>
</tr>
<tr>
<td>HashMap</td>
<td>hm</td>
</tr>
<tr>
<td>Object</td>
<td>o</td>
</tr>
<tr>
<td>Integer</td>
<td>i</td>
</tr>
<tr>
<td>Double</td>
<td>d</td>
</tr>
<tr>
<td>Vector</td>
<td>v</td>
</tr>
</tbody>
</table>

**Primitives:** Primitives are not objects, and are not subject to their naming conventions. All primitives are named such that the first word is lowercase, and all subsequent words in the name are capitalized at the first letter. Take, for example, “intCounter”.

**Typographical Conventions**

Standard typographical conventions apply for commas (i.e., one space follows every comma). Closing braces must be aligned vertically with their associated command clause. For all control structures that take a pairing command clause (e.g., if-else, if-then-else, if-else if-else, if-elseif-else), the subsequent command clauses must occur on the line following the closing brace from the preceding command clause. For example, the following code is the only legal option:

```java
if(blah > 0){
    counter++;
```


} else{
    counter--; 
}

In for statements that only have one line of executable code, you must still use the above brace structure. The following format is illegal:

for(int i = 0; i < 10; i++) blah++;

There is no standard regarding the placement of a space between a command clause and its opening brace.

**Use of Get/Set Methods**
The use of get methods is encouraged. The use of set methods must be restricted to only those times when it is absolutely necessary. Set methods are used to change the state of the object. Under many circumstances, the objects in our applications do not require a change in state – under these circumstances, the use of set methods is inappropriate, and instead values for class member variables should be set in the constructor.

**Method Writing**
All methods, with the exception of gets and sets must be private, unless otherwise impractical. All methods should only perform one function, unless it is computationally inefficient to break apart. Furthermore, private methods should always be declared as void. In those rare instances where this is not possible, it will be acceptable to allow them to return a value. By declaring private methods as void, it prevents synchronization issues in more complex code.

**Writing Constructors**
Constructors must be used to set as many class member variables as possible. Always name the parameters in your constructor the same name as your class member variables. Always use the this.varname mechanism for assignment.

**Miscellaneous Notes**
Write code so that it has a clear intent, and such that the code is clear and understandable by other individuals. Avoid the unnecessary use of synchronized classes. For example, use ArrayList instead of Vector, and HashMap instead of HashTable unless synchronization is required. Set all objects to null once they will no longer be used, to speed garbage collection.
**Procedural Language Software Coding Standard**

The following software coding standard will be followed for all procedural languages (includes R and Perl versions 5 and lower). The standard specifies 1) the existence of a header on all code, 2) the existence of a body on all code, 3) the acceptable typographical format used for control loops and brace placement, and 4) variable names.

**Scripts and Application Files**

Every programmatic file will have a header with the following structure:

```plaintext
/******************************************************
/* Class (File) Name: xyz                          */
/* Date: MM/DD/YYYY                                */
/* Author: Firstname Lastname                     */
/* Versions:                                      */
/* Version 1.0  Lyle D. Burgoon MM/DD/YYYY         */
/* Version 1.1  John R. Doe MM/DD/YYYY             */
/* Purpose: xyz                                    */
/* Algorithm: xyz                                  */
/* Notes: miscellaneous information               */
/* Debug: information for debug purposes           */
/* Copyright YYYY Michigan State University Board of */
/* Trustees                                        */
/* All rights reserved                             */
/******************************************************
```

Immediately following the header will be the appropriate declarations and import/include statements, followed by the code body.

All code will be documented by in-line documentation. When available, code should also be documented using language specific tags for the generation of other standalone documentation (e.g., HTML/SGML/XML documents). In-line documentation is required inside methods, subroutines, and constructors. Try to align all of the in-line documentation within a block area so that they start at the same place.

**Naming Conventions**

All code constructs and variables must have names with an obvious meaning.

**Methods/Subroutines/Functions:** All method/subroutine/function names will begin with a lowercase letter. If the method name is a compound name made up of more than one word, such as `getLocusLinkIdentifier()`, the first word will begin with a lowercase letter, while each subsequent word will begin with a capitalized letter.
**Objects:** Object names will begin with a lowercase letter to denote the type of object that it is. For user generated objects, the object name will begin with the first letters of each word in the class name, in lowercase, as a prefix directly attached to the object name, where the first and subsequent words begin with capital letters. For example, a new TrafficDirector object, would be called tdObjectName.

**Primitives:** Primitives are not objects, and are not subject to their naming conventions. All primitives are named such that the first word is lowercase, and all subsequent words in the name are capitalized at the first letter.

**Typographical Conventions**
Standard typographical conventions apply for commas (i.e., one space follows every comma). Closing braces must be aligned vertically with their associated command clause. For all control structures that take a pairing command clause (e.g., if-else, if-then-else, if-else if-else, if-elseif-else), the subsequent command clauses must occur on the line following the closing brace from the preceding command clause. For example, the following code is the only legal option:

```java
if(blah > 0){
    counter++;
}
else{
    counter--;  
}
```

In for statements that only have one line of executable code, you must still use the above brace structure. The following format is illegal:

```java
for(int i = 0; i < 10; i++) blah++;  
```

There is no standard regarding the placement of a space between a command clause and its opening brace.

**Method/Subroutine/Function Writing**
All methods/subroutines/functions, with the exception of gets and sets must be private, unless otherwise impractical. All methods should only perform one function, unless it is computationally inefficient to break apart. Furthermore, private methods should always be declared as void. In those rare instances where this is not possible, it will be acceptable to allow them to return a value.

**Miscellaneous Notes**
Write code so that it has a clear intent, and such that the code is clear and understandable by other individuals.
SQL Coding Standard

The following software coding standard will be followed for SQL. The standard specifies the typographical conventions used. If SQL code is generated “on-the-fly” for simple purposes, it is not required that a header be generated. However, if the SQL code is part of a larger body of SQL code, or the SQL code is saved within an SQL script file (.sql typically) then a header must be present. The SQL typographical conventions must be followed in all other programmatic contexts where SQL is used (e.g., Perl, Java, SAS, R, etc…), but the SQL header does not need to be present.

SQL Script Files

Every programmatic file will have a header with the following structure:

```
/*************************************************************/
/* SQL Code Name: xyz                                     */
/* Date: MM/DD/YYYY                                         */
/* Author: Firstname Lastname                              */
/* Versions:                                               */
/* Version 1.0     Lyle D. Burgoon MM/DD/YYYY               */
/* Version 1.1     John R. Doe MM/DD/YYYY                   */
/* Purpose: xyz                                            */
/* Algorithm: xyz                                          */
/* Notes: miscellaneous information                        */
/* Debug: information for debug purposes                   */
/* Copyright YYYY Michigan State University Board of Trustees */
/*************************************************************/
```

Immediately following the header will be the code body.

All code will be documented by in-line documentation. Comments in SQL92 are denoted by either an opening /* and a closing */ or a double dash (for in-line documentation). Try to align all of the in-line documentation within a block area so that they start at the same place.

Naming Conventions

Aliases used within a piece of code must make sense and be documented. Use of single letter aliases is discouraged.

Typographical Conventions

Standard typographical conventions apply for commas (i.e., one space follows every comma). Parentheses, when used, do not need to line up, and the closing parenthesis must be “attached” to or immediately following the final word in that clause. Major clauses must occur at the same level as their cousin clauses. All clauses that occur within another clause (e.g., `JOIN` which only
occurs within a FROM) must be placed on the following line, and must be indented. All clause
names and their predicates must be capitalized (e.g., SELECT, FROM, JOIN, WHERE, GROUP
BY, ON, etc...).

Properly formatted SQL code would look like the following:

```
SELECT name, locus_link
FROM genes3
    JOIN gene_names3 ON genes3.gene_id = gene_names3.gene_id
WHERE unigene_cluster_organism = 'Hs'
ORDER BY name;
```

**Miscellaneous Notes**
Write code so that it has a clear intent, and such that the code is clear and understandable by
other individuals.
SAS Coding Standard

The following software coding standard will be followed for all SAS code. The standard specifies the typographical conventions used. If SAS code is generated “on-the-fly” for simple purposes, it is not required that a header be generated. However, if the SAS code is part of a larger body of SAS code, or the SAS code is saved within a SAS file (.sas) then a header must be present.

SAS Files

Every programmatic file will have a header with the following structure:

```sas
/******************************************************************************
/* SAS Code Name: xyz */
/* Date: MM/DD/YYYY */
/* Author: Firstname Lastname */
/* Versions: */
/* Version 1.0 Lyle D. Burgoon MM/DD/YYYY */
/* Version 1.1 John R. Doe MM/DD/YYYY */
/* Purpose: xyz */
/* Algorithm: xyz */
/* Notes: miscellaneous information */
/* Debug: information for debug purposes */
/* Copyright YYYY Michigan State University Board of Trustees */
/* All rights reserved */
/******************************************************************************
```

Immediately following the header will be the code body.

All code will be documented by in-line and outside of block documentation. Comments in SAS are denoted by either an opening /* and a closing */. Try to align all of the in-line documentation within a block area so that they start at the same place.

Naming Conventions

Variables in SAS need to follow the SAS naming conventions for variables. Names must have clear and definable meanings. SAS typically deals with tables and matrices (if using IML). All tables and matrices must have clear and meaningful names, and all columns within these structures must also have clear and meaningful names. Documentation of a table or matrix must include the name and purpose of the columns in the table.

All SAS macros must have a header of the following structure:
Typographical Conventions

Standard typographical conventions apply for commas (i.e., one space follows every comma). A SAS procedure is written such that the procedure call is on the first line, and followed by its required first-line parameters. Parameters that are set following the first semicolon in the first-line parameters call are indented once and occur on a new line, with each parameter existing on its own line. The run command associated with one or many SAS procedures occurs at the same level as the procedure(s) that it calls – thus not being indented.

For example, the following code would be used to define and sort a table:

/*Define a table named tempTable
id: id-value
oldValue: old value
newValue: new value*/

data tempTable;
  set oldTable;
  if oldValue < 0.5 then newValue = -1;
  else newValue = 1;
run;

/*Sort tempTable by the id*/
proc sort data = tempTable;
  by id;
run;

Miscellaneous Notes

Write code so that it has a clear intent, and such that the code is clear and understandable by other individuals.
Testing and Validation

All software that is written within and for this lab will be tested and validated. This testing and validation process will be documented. As part of the Final Report, a Testing and Validation Matrix (TVM; Appendix B) will be created that lists all of the tests and validation processes and whether or not the test succeeded or failed. In those instances where a binary success/failure scheme is impractical (i.e., complexity tests), the matrix will report the numeric or textual value, and whether this value is normal, high, or low.

All testing strategies are the same, but different tests have different purposes. Once a standing code unit is created, such as a class, it must be tested to ensure it works properly and was designed properly. As more code is added, these previous tests must be run and the code must still pass all previous tests. It is inappropriate to assume that a class will work properly if no code changes were made to it. Two different classes of tests must be run regularly on manufactured code: 1) Functionality Tests and 2) Complexity tests.

Functionality tests are unit tests that determine if a class is functioning properly. They test if the results from the class are as expected. JUnit is the unit testing framework used by this laboratory.

Functional classes may also be overly complex. Complexity tests quantify the degree of complexity inherent within the code. Generally, more complex code is less understandable, and more difficult to maintain. Software developers must run these tests to determine if their code is overly complex. The development team must justify their assertions for using otherwise complex code as assessed by a complexity test. For more information on complexity testing please see: http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html and http://hissa.ncsl.nist.gov/HHRFdata/Artifacts/ITLdoc/235/mccabe.html. Laboratory software writers will calculate cyclomatic complexity metrics for their classes.

Validation is not the same as testing. Unit testing ensures each class or code unit works properly, and tests each class individually. Testing can also occur across classes to ensure proper communication and data passing between classes. Validation, however, ensures that the answers generated by a class and by an application are correct. That is, validation ensures the algorithm yields the correct answer in reality, while testing only ensures the computation generates what was intended. Thus, it is possible for all code to pass a test, but for it to yield invalid answers.

When creating testing and validation schemes, the project team may need to identify data that already exist or create toy data where the solution set is known. Because the data used to test and validate the system are crucial to the success of the system, all datasets that are going to be used need approval prior to use.

Throughout the development cycle, the project team will be required to generate progress reports at regular intervals. As part of these reports, the project team is expected to discuss any major issues revealed during testing, the test methods, the validation methods, and the data used for testing and validation.
The testing methods noted above may not be practical for database design issues. Please see the section on Database Development for more information.
Peer Review

All regulated material that is developed within the lab must be peer reviewed prior to beta testing. This peer review process is crucial to the overall QAP, and serves to identify code that is not as efficient as possible, problems with the algorithm, and cosmetic problems with the front-end of the application.

Code Review Package

In order to properly code review a piece of code, the project team must disseminate to all members involved in the review a code review package. The package will consist of:

1) Source code
2) The application in the same executable format that will be used by end-users
3) The Application Planning Quality Assurance Plan
4) Any external documentation (e.g., javadoc, text documents)
5) Sample input files
6) The user documentation

Incomplete code review packages will not be reviewed, and the project team will be asked to create a proper code review package.

Code Review Elements

During a code review, all of the participants are asked to analyze the entire package. Reviewers need to be especially watchful for:

1) The application addresses all of the intended uses
2) The application encompasses all of the use-case scenarios appropriately
3) The system design is appropriate for the intended purposes
4) The code follow all relevant coding standards
5) Whether or not all of a classes functionality is being tested
6) The validation method is appropriate for the purposes and data that will be generated by end-users
7) Documentation is appropriate and understandable
8) The code is understandable
9) The user documentation is understandable

The project team must address all of the peer review comments. The lead bioinformatician may take responsibility for further review following initial peer review to speed the process if necessary. If and when this occurs, a notation must be made within the final report.
Database Development

All major database development efforts (i.e., development of subsystems or whole databases) require adherence to the standards within this document. All minor database development activities (e.g., development of a single table, creation of synonyms, indexes, fields, etc) do not require the substantial QAP efforts described in this document. Minor database development activities do, however, require the prior approval of the lead bioinformatician and database administrator.

Major database development projects must adhere to the above standards; however, as development may occur by using the management software directly, and may not require the production of SQL code, schemas become the primary regulated material.

Database Schema Testing

Once a database schema is generated, the design team must ensure the documentation for the schema is up-to-date. The schema must then be built into the database and populated with appropriate data. Testing of the schema and the database design occurs through SQL queries and is primarily driven by use-case scenarios, and data modeling requirements.
Coding Best Practices
When it is possible, all coding projects should have at least two individuals working on it. Each project will be assigned a project manager who is ultimately responsible for the execution of the project. The other members of the project team will serve to assist the project manager in ensuring the project is completed according to the QAP.

When a project team only has two members, the project manager is primarily responsible for getting the project done. The second member should be used as a sounding board, and a preliminary check on tests, but they should not be expected to contribute code. Exceptions to this are: 1) when the project team has been instructed to use a peer-coding strategy, where the team shares the same keyboard, mouse, and monitor, and work on code in a collaborative manner, and 2) when directed by the lead bioinformatician.

Comments on a project, especially use-case scenarios should be gathered from the end-users as soon as possible. To stimulate lab discussion, the project team must have a basic prototype (hand-drawn prototypes will suffice). These discussions should be part of the general laboratory meetings. The project team must note that the general laboratory group may not have experience with the project and its goals. In these situations it becomes necessary for the project team to direct the general lab group on sources of information prior to discussion. In some circumstances we may need to consult with outside groups that have the expertise we’re looking for to generate the use-cases. Contacting outside groups must not be undertaken without advise and consent from the lead bioinformatician. This course of action is suggested as the general lab group has in the past not been able to provide feedback on abstract computational toxicology projects without a relatively mature prototype to guide the discussion.

On larger projects of great significance, the project team is encouraged to create a prototype and demonstrate it in some fashion at a general laboratory meeting. This prototype may serve to spark further discussion, and should not be considered a completed project. It is imperative to note, however, that no changes can be made to any of the planning documents without first discussing it with the lead bioinformatician. Furthermore, existing projects may not be changed without first writing a proposal and a QAP. In instances where time is a significant factor, an expedited proposal may be written, with permission from the lead bioinformatician. In cases where an existing GUI is being updated or a new application is being developed based on old GUIs, rapid prototyping should occur to ensure the end-users are as satisfied as possible prior to any extended investment.

No independent coding projects may be undertaken at the request of any member of the laboratory without first getting permission from the lead bioinformatician and the PI. The exception to this is that the lead bioinformatician and the PI may ask members of the group to undertake specific projects. However, in these instances, a proposal and a QAP must still be produced.
APPENDIX A

Application Planning Quality Assurance Plan (QAP) Signature Page
Application Planning Quality Assurance Plan (QAP) Signature Page

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Project Manager</th>
</tr>
</thead>
</table>

All of the project participants, listed below, agree to perform our assigned duties as they pertain to fulfilling the project’s objectives, as well as the testing, validation, and evaluation of the methods and algorithms within the project. The project participants acknowledge that they are partially responsible for the success and failure of this project, and will make their best effort to perform their assigned tasks to ensure the success of this project, to the best of their abilities.

Sign and date here:

<table>
<thead>
<tr>
<th>Project Manager</th>
<th>Participant 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant 2</td>
<td>Participant 3</td>
</tr>
<tr>
<td>Participant 4</td>
<td>Participant 5</td>
</tr>
<tr>
<td>Participant 6</td>
<td>Participant 7</td>
</tr>
<tr>
<td>Participant 8</td>
<td>Participant 9</td>
</tr>
</tbody>
</table>

This project Application Planning Quality Assurance Plan has been accepted for review:

<table>
<thead>
<tr>
<th>Authority</th>
<th>Date</th>
</tr>
</thead>
</table>

This project Application Planning Quality Assurance Plan has been approved and may commence:

<table>
<thead>
<tr>
<th>Authority</th>
<th>Date</th>
</tr>
</thead>
</table>
APPENDIX B

Example of a Traceability Matrix
**Example of a Traceability Matrix**

<table>
<thead>
<tr>
<th>Personnel</th>
<th>PEI-1: XYZ</th>
<th>PEI-2: ABC</th>
<th>PEI-3: QIZ</th>
<th>PEI-4: NYC</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Doe</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Jane Doe</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jimmy Dean</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Note: These are generally better done in Excel than in Word
APPENDIX C

Final Report Signature Page
Final Report Signature Page

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Project Manager</th>
</tr>
</thead>
</table>

All of the project participants, listed below, have performed services that impacted the planning, development, execution, and final preparation of this project. They agree to all of the elements contained within this Final Report, and are responsible for its final content.

Sign and date here:

<table>
<thead>
<tr>
<th>Project Manager</th>
<th>Participant 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant 2</td>
<td>Participant 3</td>
</tr>
<tr>
<td>Participant 4</td>
<td>Participant 5</td>
</tr>
<tr>
<td>Participant 6</td>
<td>Participant 7</td>
</tr>
<tr>
<td>Participant 8</td>
<td>Participant 9</td>
</tr>
</tbody>
</table>

This project Final Report has been accepted for review:

<table>
<thead>
<tr>
<th>Authority</th>
<th>Date</th>
</tr>
</thead>
</table>

This project Final Report has been approved and this project is hereby closed:

<table>
<thead>
<tr>
<th>Authority</th>
<th>Date</th>
</tr>
</thead>
</table>
APPENDIX D

Testing and Validation Matrix Example
### Testing and Validation Matrix Example

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Package</th>
<th>Class</th>
<th>Pass</th>
<th>Fail</th>
<th>Level</th>
<th>High/Low/Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>getNum-1</td>
<td>gui</td>
<td>Window</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getName-1</td>
<td>dataIO</td>
<td>FileParser</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity-1</td>
<td>dataIO</td>
<td>FileParser</td>
<td></td>
<td>5</td>
<td></td>
<td>Normal</td>
</tr>
</tbody>
</table>

Note: These are generally better done in Excel than in Word
APPENDIX E

Use Case Document Template
dbZach System:
Metabolomics Interface Use Case Document

Revision History

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Reason For Changes</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Initial Draft</td>
<td></td>
</tr>
</tbody>
</table>
1. Introduction

[insert text here]
2. Actors

[insert text here]
3. Use Cases

3.1 Use Case Name

[Note: Use Case ID is of the format: INYY-DDDDDDD, where IN is project initials, YY is year, DDDDDD is numeric ID, with first one being 000001, and second one being 000002]

<table>
<thead>
<tr>
<th>Use Case ID:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name:</td>
<td></td>
</tr>
<tr>
<td>Created By:</td>
<td>Last Updated By:</td>
</tr>
<tr>
<td>Date Created:</td>
<td>Date Last Updated:</td>
</tr>
</tbody>
</table>

| Actors: |                     |
| Description: |                     |
| Trigger: |                     |
| Preconditions: |                     |
| Postconditions: |                     |
| Normal Flow: |                     |
| Alternative Flows: |         |
| Exceptions: |                     |
| Includes: |                     |
| Priority: |                     |
| Frequency of Use: |                     |
| Business Rules: |                     |
| Special Requirements: |                     |
| Assumptions: |                     |
| Notes and Issues: |                     |