4/3/2017

Software Testing Documentation

Team U.I. Fit – Version 1.0

Team Members:
Charles Chatwin
Matthew Burns
Tanner Brelje
Joshua Gutman

Sponsor: Dr. Abolfazl Razi
Mentor: Dr. Abolfazl Razi

The purpose of this document is to provide a timeline and implementation plan for testing BioNetWeb.

TABLE OF CONTENTS

INTRODUCTION

The field of molecular biology is an ever advancing science that requires tedious experimentation and research. In order to generate concrete research results, many molecular biologists must place precious time and resources into large-scale experiments. These experiments are used to show the process of biochemical molecular interaction, or in layman's terms, the result of the reaction between two or more molecules. In order to aid this meticulous process, Northern Arizona University graduate Brandon Thomas, in tandem with an experienced team of graduate students and molecular biologists, created the program BioNetFit. BioNetFit is a command line tool that was created to provide a fast and easy method to simulate complex molecular bonds. These simulations allow researchers to later run the tests in a real environment in a way that gives them a degree of confidence in the expected interaction.

Built on top of BioNetGen and NFSim, BioNetFit allows researchers to simulate a single experiment many times with a range of parameters. The results of each of these simulations is compared to an experimental results file that the researchers upload. Because the combinations of different parameters scale non-linearly, various methods are used to select the best combinations of parameters, including genetic algorithms, simulated annealing, and a few others. After the simulation has been run many times (usually thousands), BioNetFit creates a final output file that shows information about the molecules involved in the reaction.

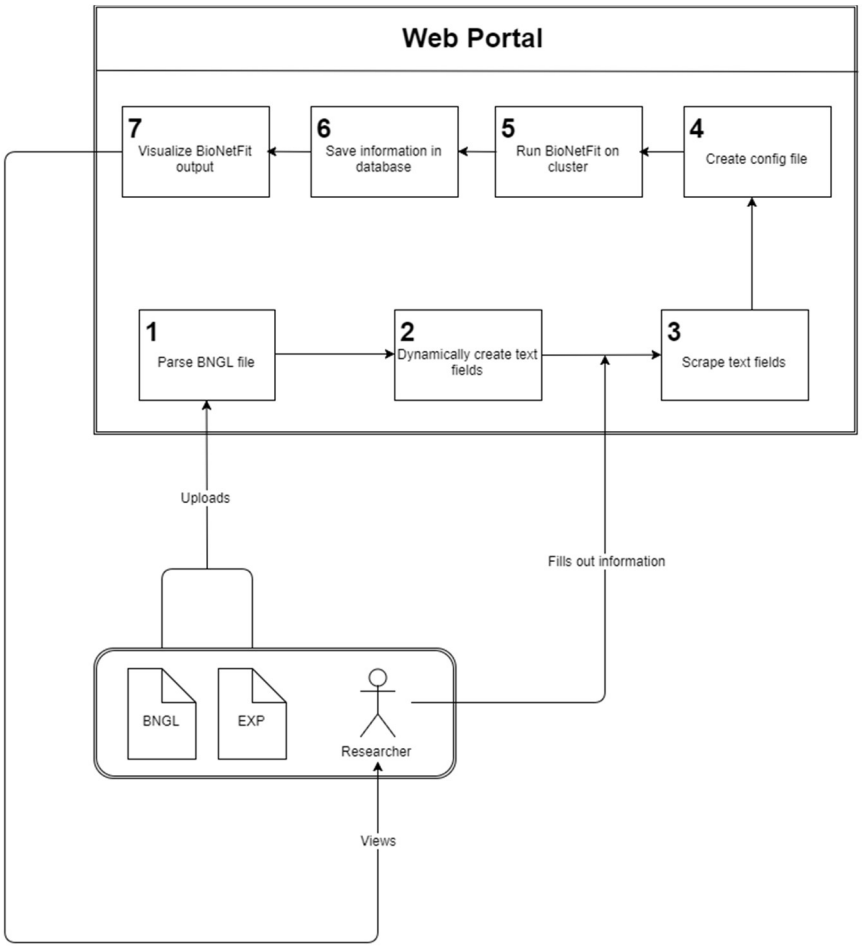
While the program is incredibly useful, its implementation is not user friendly. It exists as a unix-only, command line tool. Researchers who are not technically proficient will struggle at getting the program to run and will attempt to find other avenues when faced with having to spend time learning a new system to run their tests. Additionally, the results currently output by the program are difficult to digest, and do not lend themselves easily to analysis for data collection by researchers. Without much labeling or clear direction in the results, researchers will have to extract results from a command line output, which as previously discussed, is problematic for many scientists. Lastly, the program cannot run large scale experiments in a short amount of time. This lessens the practicality of the program as the experiment becomes more and more grand in scale. In order to make BioNetFit the stellar application it can be, these issues need to be addressed. If done correctly, the program could see widespread use in molecular biology labs all over the world.

Created in order to tackle these challenges, Team U.I. Fit, composed of Charles Chatwin, Matthew Burns, Tanner Brelje and Josh Gutman, will develop software solutions in order to turn BioNetFit into the powerful program it can be. Led by client and mentor Dr. Abolfazl Razi, our team will aim to:

- Create an attractive and simple web portal for BioNetFit.
- Visualize the results of BioNetFit into easily digestible data for researchers.
- Implement parallelization in order to run large experiments on a computer cluster.

Firstly, our team will develop a simple and effective Web 2.0 Graphical User Interface that will house the BioNetFit software. This web page will be easily accessible and easily usable by any researcher who wishes to run tests using the BioNetFit software. In the web portal, the user can either create their own BNGL file from

scratch, or upload one that was either previously downloaded, or was saved remotely on the website itself. From here, the user can run the file either locally on their own system or, in the case of a large file, run the file on the computing cluster Monsoon, available on the Northern Arizona University campus. From there, the program will output a configuration file that will be visualized to the user using graphs and charts. Finally, the user can either save and visualize the outputs on the website, or download them to their own local machine. The BNGL files can then be tweaked if wanted, and the experiment can be run again.



In this document, our team will outline the testing that will take place in order to ensure that the BioNetFit software solution is working to its full potential. In order to label these tests, we will split them into 3 categories, Unit testing, Integration testing, and Usability testing. Each of these categories will delve into the specifics of the software in its current implementation of the software solution, with each of the specific tests assuring the usability of a certain aspect of the software. The testing and protocols will be laid out in each section, with information pertaining to how the test will be implemented, and what the expected result of each step of the testing is. The final result should be a comprehensive guide to understanding how we as a team are planning to ensure that the BioNetFit software solution is one that will be resistant to bugs and possible failures that may be unforeseen.

UNIT TESTING

The first section of testing for the BioNetWeb software solution is dedicated to the unit testing of the software. The purpose of a unit test is to delve into the individual aspects of the software, and ensure that each of the most pivotal pieces of the software work correctly as intended. In order to dissect and create test for each of these pieces, we must first divide the software solution up based on its main components. From there, we will dive deeper into each of these components, looking at what sections of code specifically fuel each of them. Once we identify these code segments, the final step will be designing and implementing a testing strategy. We will delve into what packages and techniques will be used for the test, as well as a timeframe or strategy for how the implementation will take place. With that in mind, let us first identify the main components of the software solution.

In its current state, we can break the solution up into 4 main sections:

- Web Portal
- Database
- Visualizations
- Web Services

1. The Web Portal section deals with the interface that is seen by the user. These are the building blocks that display the website in its finished state to the user. In order to ensure that the U.I. works to its full potential, we must ensure that factors such as login, web navigations, etc. work correctly and catch expected errors.

2. The Database section deals with the MongoDB that is implemented on the server that has been provided by NAU web services. In order to ensure that the database is a stable addition to the software solution, factors such as data insertion, retrieval, and storage work without issue.

3. The Visualizations sections regards the graphs and visuals that are generated by the solution in order to help users to make more sense of the results that are generated by BioNetFit. Factors that require testing in this area are accurate graph generation, dealing with errors in results, etc.

4. The Web Services section includes both the server that is going to host the website and database, as well as the provided virtual environment that has been provided for the use of the website. While most of the interactions between our own solution and the provided services are mostly out of our control, we can be sure to plan in the events of server shutdowns and restarts, as well as testing to ensure the site cannot be accessed outside of the NAU VPN.

WEB PORTAL

The web portal of the website is the most important part of the entire implementation, as it provides a bridge between the user and the BioNetFit software. Because of this, it is paramount that the functionality of the website be near perfect. The implementation must account for not only what is expected of the website, but must also handle possible errors that can happen within the code. With that in mind, we will be implementing tests of the following segments in order to ensure that the code will work as expected:

- File upload accepts/processes only the correct file formats.
- Parameter generation creates only acceptable parameters.
- Login functionality allows for seamless account access.
- Website does not hit dead zones/ pages that do not exist.

Note that this is not a complete and comprehensive list of all tests that may be implemented in this process, however these factors are some of the most important within the website, and will be given priority when generating the unit tests. For each of these tests, we will outline the issue that we want to eliminate, how we plan to approach the test, any libraries that may exist in order to assist the process, and the expected result of each of the tests. This process will also be used in the generation of tests for the other three sections of unit tests that will be outlined within the paper.

Issue	**Approach**	**Implementation**	**Result**
File upload must only take in files of the expected types. Uploading incorrect files may cause the service to crash or run into errors.	Generate a test that intentionally tries to upload correct and incorrect file types.	The ability to process and understand whether a file is of the correct type is included within the assertions provided by HTML and python.	The web portal will accurately identify whether a file is of the correct type, and will either correctly process the file, or handle the incorrect file and return an appropriate error message.
The parameters generated within the GUI must not be able to include values that are not accepted by BioNetFit.	Generate a test that tries to push the boundaries of each parameter.	For each of the possible parameter, a range checker will implemented in Python, which will check the given parameter value against the possible range of values.	The web portal will accurately identify whether a parameter is or is not allowed within a config file, and will ensure that files with incorrect parameters will not be used.
Login functionality must allow for seamless login and account access.	Generate a test that tries to login with real and fake accounts.	Django is able to register whether a login attempt is valid. We must ensure that python processes the registration correctly.	The web portal will allow for the login of real accounts, while rejecting fake ones/ providing error messages.

The website itself must not allow any deadzones under its URL, meaning non-existing HTML pages must transfer to accurate, existing pages.	Generate a test that attempts to visit HTML pages that do not currently exist under the current implementation.	Django has options that allow programmers to reroute incorrect page requests to HTML pages that exist specifically to be a landing page for incorrect requests.	The web portal will accurately differentiate between correct and incorrect HTML requests, and will reroute the request based on the unit test and Django.

DATABASE

The database of the website is done in MongoDB. MongoDB is what is considered a NoSQL database. This means that the database does not have a specific format or structure that it must adhere to. While this allows for a lot of freedom when it comes to the implementation of the database itself, we must ensure that the space is being used effectively. In order to ensure this, we will implement two unit tests that will ensure that the database not only has some form of structure, but also does not accept files that would be too large, causing an inordinate amount of data to be used by the server. Therefore, the tests will explore the following factors:

- Users and files attached to users will be inserted with a specific format.
- The database will not allow files deemed too large by the system.

Note that this is not a complete and comprehensive list of all tests that may be implemented in this process, however these factors are some of the most important within the website, and will be given priority when generating the unit tests.

Issue	**Approach**	**Implementation**	**Result**
User and file storage must follow the same formatting in the database.	Generate a test that intentionally tries to create correct and incorrect user->file relations.	PyMongo allows for the implementation of MongoDB syntax within python files. From here, we can apply python assertions to the entry strings.	The database will accurately identify whether a user is of the correct formatting, and will handle and return error message for incorrect formatting.
Files stored within the database must not exceed a certain size. This size will be determined as further implementation is done.	Generate a test that tries to upload and insert files that take up too much space on the database.	PyMongo allows for the implementation of MongoDB syntax within python files. From here, we can apply python assertions to the files.	The database will accurately identify whether a file is of too large and size, and will reject the upload and handle the error if the size is detected.

VISUALIZATIONS

The visualizations of the BioNetFit results is arguable the second, or possibly even the most important portion of the software. The visualizations and graphs that are provided by the website must not only be readable by the people who use the software, but they must also be accurate to the data, as well as be useful to those who will be deciphering the data as a whole. Unit testing of the visualizations may prove to be tricky, as it is not likely that we will be able to feasibly employ image processing within the confines of a unit test. With that said however, there are ways that we can test the actual data that is used by the visualizations themselves. If we can test and ensure that these values are correct, than testing the actual generated images will not be required. In order to test these values, we will look into the following three issues.

- Is the best fit generation being used in the visualization?
- Has the average of the generations been calculated correctly?
- Handles and acknowledges bad data.

Note that this is not a complete and comprehensive list of all tests that may be implemented in this process, however these factors are some of the most important within the website, and will be given priority when generating the unit tests.

Issue	**Approach**	**Implementation**	**Result**
Visualization must use the correct best fit generation file.	Generate a test that provides 1 correct and multiple incorrect files.	BioNetFit does not currently have unit testing that can check a correct best fit, so it will be up to us as the developers to introduce our own solution.	The visualization will correctly select the best fit file, rejecting the remainders and notifying the test of its decision.
Visualization must accurately calculate the correct average generation.	Generate a test where the average has been manually calculated, and test the output of the program against this case.	BioNetFit does not currently have unit testing that can check a correct best fit, so it will be up to us as the developers to introduce our own solution.	The visualization will correctly generate the average of the provided files, and will compare this average to the test.
Visualizations must be displayed to the user within the HTML. The visualizations also acknowledge bad data and alert the user of such.	Generate a test that attempts to generate a visualization with both usable and unusable data.	D3 offers libraries ensure that data being used is correct, and will additionally make use of python unit testing.	The visualization will appear correctly in the case of data that is usable, and will manage and return accurate error messages for data that is deemed poor.

WEB SERVICES

In the final section of the project, we will focus on the Web Services aspect. Truth be told, as developers who will be simply using the NAU services to host the website, as well as simulate a Monsoon environment, there is little control that we have in the actual implementation of this aspect of the website. We will discuss this section more in depth in later portions of this document, but as of now there are few aspects of the servers that we can design unit tests for. As for the Monsoon environment, it is important that we do not overload the environment with too many tasks. Therefore, our main unit test for this section will address the following:

- The Monsoon virtual environment will not become overcrowded with requests.

Note that this is not a complete and comprehensive list of all tests that may be implemented in this process, however these factors are some of the most important within the website, and will be given priority when generating the unit tests.

Issue	**Approach**	**Implementation**	**Result**
The provided Monsoon virtual environment must not be bogged down with a high volume of requests.	Generate a test that attempts to overload the Monsoon virtual environment with requests.	In order to handle this, the tests will likely be done directly within the Monsoon environment. We will have to delve into options that will allow us to limit the use of the environment.	The Monsoon environment will recognize when it has been requested to do too many jobs, and will reject new jobs provided by the test.

INTEGRATION TESTING

The next two sections of testing will delve into the interactions between the components of the software solution itself, as well as the interactions between the end user of the software and the solution itself. In this section, we focus on the former. Because there are many separate parts of the solution that must work together correctly, it is paramount that the pieces mesh with little to no error. In order to digest what needs testing within the system, we will split up the testing based on the possible interactions of the software. These interactions include:

- Web Portal ⇔ Database
- Web Portal ⇔ Visualizations
- Web Portal ⇔ Web Services

For each of these interactions, we will explore the events that would require the two sections of the solution to interact, as well as the possible errors that could occur, and a detailed plan for testing in order to avoid complications when the interaction must by implemented and work seamlessly.

Web Portal ⇔ Database

Summary:

The web portal mainly accesses the database in order to load and save files based on a user that is signed in to the system. When a use uploads or runs an experiment on the website, the portal must save this information to the database in order to make it available for future use. It's important that the data is saved under the correct usernames, and the strings that hold the information from the file are not corrupted when saved to the database.

Possible Errors:

The possible errors that could arise from this interaction are as follows:

- Files saved under an incorrect username.
- Files duplicated/username duplicated in database.
- Files corrupted/inaccessible by the system.

Testing Plan:

In order to test this interaction with the system, we will be able to create a testing code within python using PyMongo. In order to test each of these possible errors and check for them, first attempt to save data using usernames that have similarities within them. The issue with MongoDB is that many times when usernames are similar, there is a level of specification that must be met. By attempting to intentionally "break" this system, we can ensure that our file storage method will not save files incorrectly, and will additionally not duplicate usernames within the system. As for testing corruption of files, this can be solved by a test that simply uploads a

file a given amount of times to similar or differing usernames, accesses the file from the database, and attempts to scan the file in order to test it against the original.

Web Portal ⇔ Visualizations

Summary:

The web portal will serve as window for the user that will present the visualizations. After a test has completed its run, and the data is retrieved by the portal, the visualizations will have to accurately present this data in a manageable form. The main issue that we may face in this situation is the visualizations are done in a JavaScript library D3, while the remainder of the implementation is done within Python.

Possible Errors:

The possible errors that could arise from this interaction are as follows:

- Syntax difference may cause an error when trying to display visualizations.
- Visualizations may not scale correctly to be displayed.

Testing Plan:

In order to test this interaction with the system, we will first ensure that the generation of the visualizations separate to that of the web portal is successful. From there, we will create a set of visualizations to be loaded into the web portal and displayed to the user. Since these require a human to confirm whether the visualizations are accurate to the created cases, we will then proceed to look over each of the generated graphs and confirm that they correct. Additionally we will delve into the data being displayed in order to assure that data is equal as well.

Web Portal ⇔ Web Services

Summary:

The web portal will have to be hosted on the web services provided by NAU. The Django framework will be hosted within the server and will run the website. This website can only be accessed by people either at NAU or on the NAU VPN. This assures that people without the correct credentials will be barred from accessing the website. Even with this in mind though, safety is without a doubt a number one priority, so we should treat it was such. Additionally, the server going down or disconnecting from virtual environment could cause lots of issues as well.

Possible Errors:

The possible errors that could arise from this interaction are as follows:

- Users without correct verification access the site.
- Outages could cause loses in data and progress.

Testing Plan:

In order to test this interaction with the system, we will have to approach both issues head on. In the case of security, there are many ways that we can test the security of the system in our own hand. The issue arise when the link to the website is made available for users. With more and more users on the system, the higher risk factor grows for the environment becoming to public. In order to test how we can prevent this, can range efforts from switching the host URL, to testing the security of the NAU VPN logins. At the end of the day though, we will likely have to keep up to date with the managers of the server in order to assure that nothing is suffering from abuse. In the event of an outage however, there are plenty of tests we can make to assure that we stay afloat. A key idea is to save the data locally until the process is done, that way if the connection is lost, the data itself is not corrupted and the experiment just has to be run an additional time.

USABILITY TESTING

In our final section of testing, we will focus on the end users, and the usability of the software as a whole. This is especially pivotal for our software solution, as the original intention for the design was to increase the overall usability and accessibility of NAU's BioNetFit software. The best way to then ensure that the software is successful in its intention, it is paramount to conduct usability testing. The main goal of usability testing is to understand how the software is used by the end users, and to collect data based on their experiences with the software. Up to this point, a majority of the user testing for the software has been done through our mentor, Dr. Razi. He has been incredibly helpful in giving as an outsiders view on the software, and has suggested many changes that have ultimately steered the project in a positive direction. Now, we must take this process further and reach out beyond Dr. Razi, in order to receive more insight into the creation and look of the website and visualizations as a whole.

In order to get the information we need in order to improve the website, we must first address the major points of who will be using the website, and what the step by step plan will be in order to interview these people. Let us first begin with the "who" of the situation. The concept of BioNetFit was to create a program that allowed biomolecular scientists run large scale experiments on their own computers. This would eliminate a large amount of time that would be spent within laboratories, and would also help cut costs for those who have to conduct the experiments with real life materials. Naturally then, the first choice for people that we would like to test our website would be biomolecular engineers who have done experiments of this caliber in a real life lab setting. At the current moment, our best options for optimal test subjects are as follows:

1. Correspondents at Los Alamos research center in Nevada. Over the course of the production of the software solution, researchers at Los Alamos have expressed interest in both assisting the production and testing the online software solution. This would be the most optimal pool of testers, as their years of experience will be incredibly beneficial to understanding how our program will help them.

2. Professors of Biomolecular science at NAU. This would be another great pool of users to interview, as professors have dedicated a large portion of their education to this specific topic. While they may not have a lot of knowledge of computer science, their knowledge of molecular combinations would be incredibly helpful in improving the software solution. Additionally, professors are likely to be more easily accessible, as they are on campus for large periods of time.

3. Students of Biomolecular science at NAU. The benefits of interviewing this group is similar to that of the professors, however they may not share the same experience. However, people within this group will likely be more available for interviewing, as the amount of students within a field will likely outweigh the professors.

4. Students of Computer Science at NAU. This group would be beneficial not for input on the biomolecular aspect, but for input on the implementation and website design as a whole. While a CS

student may not know a lot about molecules, a second opinion on the design of the website would be very helpful.

Now that we have an idea of what kinds of people we would want to test the software, we now move on to the plan of action. How will we most effectively get the information we need? What is the most optimal way of having them test the software? Below, we lay out our step by step plan:

1. Interview the subject about their experience in the field. Ask about what they enjoy and dislike about conducting real life experiments.
2. Introduce the concept of the software. Explain what it is meant to do, what it is supposed to fix, and the overall goal of the program.
3. Walk them through how the software works, ask them questions along the way to gauge the experience.
4. After the completion of the test, give them a set of questions in order to gauge how the software is performing at a whole, what people liked, what they didn't, etc.

    - Questions should centered on the experience, and how it differs from the real life experience. It is important to truly understand what makes the software solution better than the current experience that is it replacing.
    - Ex: How does the experience offered by BioNetFit compare to that of conducting a real world experiment?
    - Ex: How easy is it for you to interpret the results from real experiments, compare to the ease of interpretation from BioNetFit (both before and after the software solution).
    - Ex: How do you enjoy the stylization of the BioNetWeb portal? Is it visually appealing?

5. The test should be done with as many people as possible, and from various groups that were suggested previously. This will give us a wide range of opinions that we can use to factor into our final design.
6. Additionally, after changes are made, we should make an effort to revisit some of the people that we have already had test the software. This way, they can express their opinions of the changes to us, to let us know if we are moving in the right direction.

The last phase of the testing should focus heavily on talking to experts about the software. This is where it would be very key to have the opinion of the Los Alamos team. The software is ultimately going to be used by professionals in the field, so it the final say should belong to them. This may turn out to be tricky due to the amount of distance between NAU and Los Alamos, however we could make use of online services that would allow for communication via video. In the end gather these opinions would be helpful in not only creating a good looking product, but creating a functional and exciting piece of technology that could be used for a long time to come.