Requirements Analysis

The Disease Outbreaks Team
Abdulaziz Alhawas
JP Labadie
Jordan Marshall
Luis Valenzuela

Table of Contents

Introduction

The purpose of this document is to define the Business Requirements for the Disease Outbreaks Capstone Project sponsored by TGen North. The intended audience for this document includes all associates of TGen North, as well as the capstone professors overseeing these projects. Components of the UI solution include: problem and solution statements, functional requirements, environmental requirements, non-functional requirements, potential risks, and the group's project plan.

This draft is intended to be a fluid and flexible document that will change and evolve throughout the team's development and design process. This work will be completed in multiple sprints and releases. The document will help outline the main features and abilities that the interface will provide to its end users.

The background for these needs stems from an accessibility gap of the end users at TGen North, these scientists need a way to easily communicate with their computer cluster without having to go through the command-line. Because of the unfamiliarity of the command-line that non-programmers have, it can be difficult for these users to run a task that is fully to their specifications and liking. A more graphical user interface is a great solution that can increase functionality, while also reducing confusion when running through these command-line tasks.

The objective is to create a stand-alone application that will bridge all of the gaps that the current command-line tool generates. The application will run through a "wizard" like interface to make sure a user can send off a task easily and will include separate tabs along the interface to reduce cluttering and confusability. It will only show options that are completely necessary, while at the same time hiding more advanced and tedious options that do not need to be muddled through. This tool will basically abstract all of the command-line tool functionality into a more user friendly environment, allowing the end users ease of use and peace of mind.

Problem and Solution Statement

The sponsor for this project is Darrin Lemmer at TGen North. The strategic plan for TGen North is focused on diagnostic, analytic, forensic and epidemiologic research related to pathogens important to medicine, public health and biodefense. The research capabilities at TGen include a variety of DNA sequencing and PCR-based analyses, forensic analysis of outbreak and bio threat incidents, as well as advanced bioinformatics and computing infrastructure.

The Northern Arizona SNP Pipeline is the primary tool used by the disease outbreak division at TGen, the sponsor, to investigate fungal and

bacterial genomes for the purpose of tracking disease outbreaks. The pipeline is optimized to run on a computing cluster, which can be remotely accessed. Currently, users interact with a command-line program, which emulates a 'wizard' style interface, asking the user for the locations of various files, parameters that need to be included with the data, as well as other options to include while running the process. This command-line program then generates a formatted XML document, after which is passed on to the computing cluster. The NASP tool uses the information in the XML to run the process.

This command-line tool, while effective, can be daunting and unforgiving for a non-programmer to use. Right now there is no way to clear an entry, adjust an entered parameter, or fix a mistake without resetting the whole command-line tool. The command-line tool does not allow the input of custom options, unless the XML document is directly manipulated, this is not feasible for users unfamiliar with XML and its structure. The users seek a way to pass in their data to the computer cluster, run NASP, and receive their output without having to go through the command-line or directly into the XML document. The primary objective of this project is to create a more user friendly graphical interface in which users of the current command-line tool will have more flexibility and increased ease of use.

The main focus for the GUI is to replace the command-line tool entirely, providing a robust and modular interface which improves the user experience. This goal is well within our abilities, as the command line tool is sufficiently abstracted from the NASP job process. Because the NASP pipeline is dependent solely on the XML, the GUI will be completely independent of the other technologies used in NASP. The core requirement of the project is then to provide a user interface which can generate this XML and pass it to NASP pipeline to begin a job. Features of the project will build upon this, including providing a visual representation of job completion by interacting with the job manager, and visualizing the phylogenetic trees which result from the job once it has finished.

Potentially, this GUI could set the standard for genetic SNP's phylogeny. Since command-line tools are the current route to manage the NASP tool currently, a graphical user interface will provide massive amounts of extra usability to users that are not only uncomfortable with the command-line, but also unfamiliar with the generated output that will be passed to the cluster.

There were no other avenues to explore for this project, because the sponsor specifically wants a graphical user interface, as this seems to be the best and most practical solution.

Functional Requirements

The final implementation must be able to fully replace the command-line tool that is currently used to interact with NASP, while leveraging the advantages offered by a more advanced interface. To this end, it must provide an intuitive Graphical User Interface (GUI) which allows users to build a new job for the NASP tool. This GUI must also allow this job to be sent to a number of remote computing centers (currently ASPEN and MONSOON), or run locally.

To begin the process of creating a job in the new tool, users must be able to specify where output files will be written. Users must be able to provide either local or remote reference FASTA files. Users must be able to toggle the option to ignore duplicated regions found in the reference files. The user must be able to select the job management system that will be used, as well as the option to run without a job manager. Likewise, the tool must support interactions with the PBS/Torque, SLURM, and SGE job management systems. If a job management system is chosen, the system must allow users to specify a queue or partition to be used for all jobs, or use the default queue. The system should also allow users to add additional arguments to the job request.

The user should also be able to supply local or remote FASTA files from external genomes. If remote external genome FASTA's are supplied, the user should be able to specify advanced settings for the NUCmer tool.

The user should be able to supply local or remote read files if they wish. The user should be able to select which alignment tools they would like to use, including BWA, Novoalign, and SNAP. Alternatively, users should be able to provide pre-aligned files such as BAM. Depending on the choice of alignment tools and files specified, users should be able to choose only related additional options.

If the BWA aligner was selected, users should be able to select options such as running the BWA samp/se tool, and the BWA mem tool. Furthermore, they should be allowed to define any advanced settings related to the BWA tool and optional tools.

Users should also be allowed to run the Bowtie2 tool. Users should likewise be able to set advanced settings for the Bowtie2 tool.

If users chose to include Novalign as an aligner step, they should be allowed to enable and define extra Novalign settings. For instance, users should be able to specify an alternate Novalign version from the default. Users should be able to supply additional arguments. Users should also be able to Novalign's runtime settings, such as defining queue and partition, maximum memory allotment, maximum CPU allotment, and maximum runtime allotment.

If users choose to run SNAP, they should be allowed to specify local or remote prealigned SAM or BAM files to be included. Users must also select one or more SNP caller functions, GATK, SolSNP, VarScan, and SAMtools. Users must also be allowed to provide local or remote VCF files if they are available.

If users elect to include the GATK SNP caller function, they should be allowed to define advanced options for the tool. If users elect to define advanced options for GATK, they should be able to: choose an alternate version of GATK, define the queue or partition for GATK to run on, define the maximum memory allocation for GATK, define the maximum CPUs allotted to GATK, define a maximum runtime to GATK, and define additional arguments for the GATK runtime.

Users should also be allowed to define how NASP will filter results based on coverage. Users should be able to select a minimum coverage threshold, including zero (no coverage filtering). In addition, Users should be able to filter based on the proportion of reads that match the call made by the SNP caller. Users should be to enable this filtering, define the minimum acceptable proportion.

Users should be able to define advanced parameters for the MatrixGenerator processing step. These should include: define an alternative MatrixGenerator version, pass additional arguments to the MatrixGenerator, define the queue or partition for the MatrixGenerator to run on, define the maximum memory to be used in processing, define the maximum CPU count to be used in processing, and define the maximum run time of the job in hours.

Finally, the user should be able to decide if the generated matrix should include all reference positions, or if the generated matrix should mask low-quality calls. Based on the user's settings, as defined above, the tool should then create an XML document which conforms to the schema defined by TGen North. This XML is the interface used by the NASP tool, and completely defines the job and tasks desired by the user.

The tool must also be able to connect to the remote computing centers and start the job, using the XML generated and providing it to the NASP pipeline through the job managers selected. Currently, the tool must support interactions with the job managers PBS/Torque, SLURM, and SGE. The tool must also provide an interface for the user to visually track the progress of started jobs via said job managers, and to retrieve the files generated by the job upon completion. Finally, the tool should provide the user with visualizations of the generated matrix, including diff-like side-by-side views of SNPs, and a phylogenetic tree. Additional features may expand upon these visualizations, and could allow users to draw upon and annotate trees, collect individual sample information, and additional filtering.

The tool should also be 'modular' where possible, but especially in terms of the visualizations, providing users a straight-forward method for expanding or changing functionality should the need arise after the project is completed.

Environmental Requirements

The NASP pipeline runs in a computing cluster where users interact with it using a command-line program. The command-line program asks the user series of questions to locate various files and options to include when running the process. After gathering all the parameters it needs, the program generates a formatted XML document which is passed to the computing cluster in order to run the process.

Our GUI will replace the command-line tool entirely, providing a robust and modular interface which improves the user experience. Because the NASP pipeline is dependent solely on the XML, our GUI is completely independent of the other technologies used in NASP. The core requirement of the project is then to provide a user interface which can generate this XML and pass it to NASP pipeline to begin a job. Features of the project will build upon this, including providing a visual representation of job completion by interacting with the job manager, and visualizing the phylogenetic trees which result from the job once it has finished.

Since our GUI is independent of any technology that NASP use, we have complete freedom to choose any programming language we'd like. We decided to use Java with JavaFX as the GUI framework. We also intend to use D3.js for visualizations as an additional feature.

These technologies fulfill our requirements and will seamlessly integrate with our particular environment. Obviously, a Java-based application will run on any major operating system. Furthermore, since the NASP tool is abstracted behind its XML input, no difficulties will be encountered in interfacing with it. It is well within the capabilities of Java to generate XML and interact with a remote machine and its process managers. Serving the visualizations generated by any D3.js scripts is likewise well-handled by our chosen technologies. JavaFX is designed for both local GUI implementation and serving rich web-content, and thus can serve the content generated by D3.js.

Non-Functional Requirements

The GUI shall have a simple design that should not require users to undergo additional training. The GUI design needs to look similar to TGen's existing UI programs, and should follow their design patterns. Results must be processed and displayed to the GUI to allow users to visualize data in an organized way.

Additional tools should be easily integrated into the solution. The GUI needs to be implemented in a manner which produces reliable and correct XML documents. The GUI shall also provide a visual representation of NASP job completion status. Finally, the design shall be documented using TGen’s design templates.

Potential Risks

This project will make use of five technologies that serve specific purposes in the creation of the interface and the interaction with the command line tool. We will be using Java, JavaFX, and D3.js. There is a risk involved with using each of these tools.

We could very well have issues producing the XML file using Java. This is one of the most important aspects of the project, since TGen's tool relies on this file to run. Another risk associated with Java would be correctness. The Java code that we write might be able to produce the file, but the file format or file data may not be correct. This would lead to incorrect results from the tool and would therefore make anything that those results are used for also incorrect. The output must be correct to ensure that it does not have a negative effect on other processes that rely on it. The final risk associated with Java is the communication with the NASP tool which lives on a separate computer. Our program has to be able to send the file it produces to NASP which then runs based on the contents of the XML. This communication is also a vital part of the tool. Producing a correct XML file is only half of the work, sending that file to NASP is the other just as important part. The likelihood of these problems occurring however are slim. We are confident that we can create a program that reads an interface's contents, creates a correct XML given that information, and communicates with a server that NASP lives on.

D3.js is the tool that we are using to create a visualization of the results produced by the NASP tool. The script that we write should be able to create an accurate visualization in the form of a tree. The biggest risks associated with this tool would be the accuracy of the visualization and being able to have it loosely coupled with the main program. The visualization will be done within the tool but is an aspect that should be easy to modify without affecting the rest of the tool. Depending on how the programming goes this may or may not be an issue but loose coupling is something that we want to accomplish.

Using JavaFX as a tool to create the interface is something that we consider to be relatively low risk. The interface will simply present the user with some questions and options which the Java code will then translate into XML. We are confident that we can create an interface that will be able to accomplish this. There is a chance that we may not include everything that we need in the interface but after having multiple iterations of the interface the chances of that occurring are not likely.

Project Plan

UI Design and XML Generation

In the following weeks, after having a better understanding and grasp on what exactly the sponsor wants, we will begin creating some initial interface designs based on some of the tools that TGen is currently using. We want the tool feel familiar to the users in terms of their interactions with it. While we make iterations of the interface we want to also be working on beginning of the writing the code that will generate the XML given any input by the user. Since the changes in the interfaces design for the most part will not have an impact on how the code interacts with the interface we can work on both simultaneously.

Communication with NASP

Once we have an interface that correctly generates the XML file we want to begin working on communicating with the servers that NASP lives on. Establishing that communication will then allow us to be able to actually send jobs to the server in the form of the XML file we generated. At this point we expect to still be tweaking the interface design as well. We might have one person working on that and three people working on getting communication working. We could have someone start working on the visualization module but we want to have the core functionality of the tool done before we move on to other aspects of the project.

Checking on Job Progress

With communication with the servers functioning this is the next module that we want to get working. TGen would like to be able to check on the completion of any given job. This will also be part of the tool. Given a job ID we plan on being able to ask the server for a status update on any given job that is currently being handled.

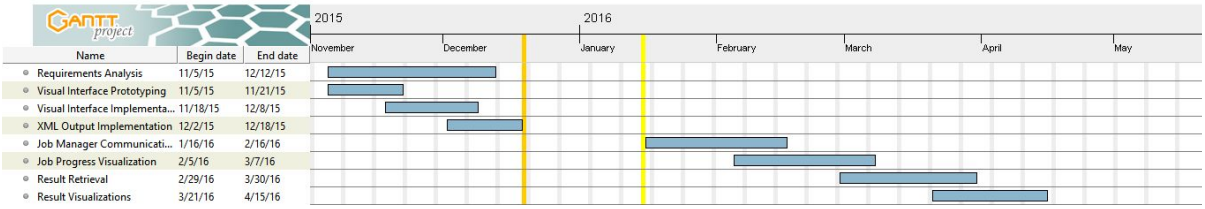
Result Visualization

After NASP completes a job, the user will be able to download the file. Using this file the tool will be able to create a visualization in the form of a tree. The goal is to have the visualization be part of the tool. This means that the tool itself should be able to generate that visualization and display it.

Timeline

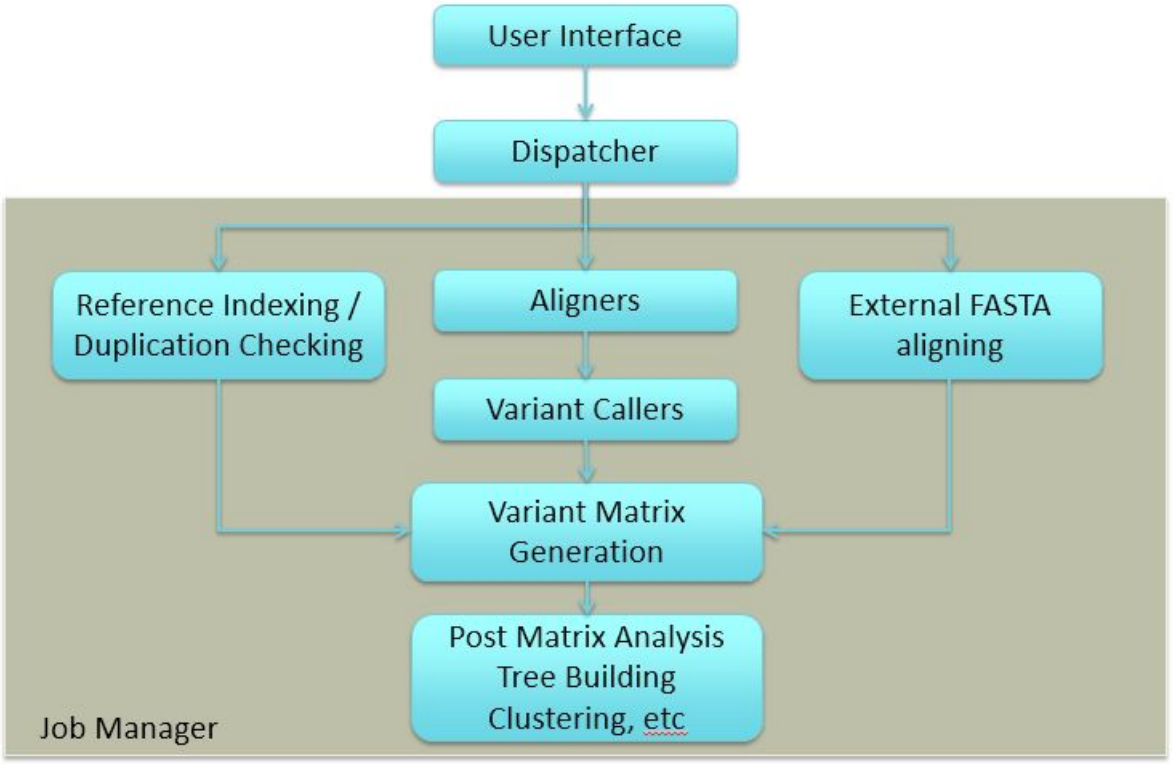
The following is a rough timeline of the milestones that we currently. We expect to adjust the timeline as we progress through the project. Specific tasks that need to be accomplished as part of every milestone will also be added.


Interface Design and Implementation
11/20/2015
XML File Generation
12/20/2015
Communication with NASP
1/20/2016
Checking Job Progress
2/20/2016
Result Visualization
3/20/2016
2015
Nov
Dec
2016
Feb
Mar
2016
Today

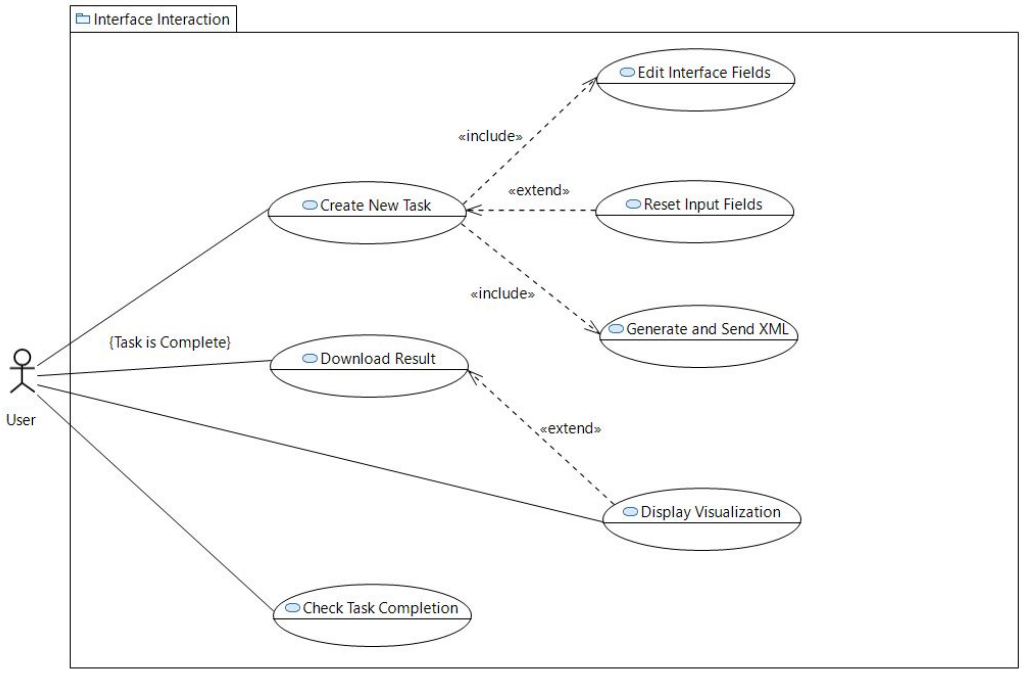

GANTT project
2015
2016
November
December
January
February
March
April
May
Name
Begin date
End date
Requirements Analysis 11/5/15 12/12/15
Visual Interface Prototyping 11/5/15 11/21/15
Visual Interface Implementa... 11/18/15 12/8/15
XML Output Implementation 12/2/15 12/18/15
Job Manager Communicati... 1/16/16 2/16/16
Job Progress Visualization 2/5/16 3/7/16
Result Retrieval 2/29/16 3/30/16
Result Visualizations 3/21/16 4/15/16

Appendix

The NASP pipeline dataflow:



GUI Tool Use Case Model:


Interface Interaction
Edit Interface Fields
«include»
Create New Task
«extend»
Reset Input Fields
«include»
Generate and Send XML
{Task is Complete}
Download Result
«extend»
User
Display Visualization
Check Task Completion