Seismic-radar toolbox GUI

Critical Design Review Document

Dr. Chris Gerbi

Team Penobscot
Adam Farrington, Alex Thacker, Jens Hansen, Nathan Gazey

December 19, 2019
Critical Design Review Document, Version 1.0.0

Table of Contents

Introduction

Executive Summary

The Seismic Radar Toolbox, SeidarT, is a modeling software used in glaciology research. Using graphical representations of a glacier, users can approximate the effects of both seismic and radar propagation through the approximated glacier. This approximation allows users to more efficiently study phenomena such as glacial creep and the alignment of c-axes. The study of these concepts will heavily impact our understanding of glacial structures in the wake of climate change. Glaciers that are less structurally sound may have a tendency to melt as temperatures rise increasing sea levels worldwide. Thus it is the goal of the application's creators that it be further developed and distributed such that glaciology researchers have access to this powerful modeling tool.

As SeidarT currently stands, it requires intimate knowledge of a command-line interface, as well as a unique text file formatting. Thus to use the application, one must first spend a large amount of time familiarizing with both the .prj text format as well as SeidarT's commands before any reasonable modeling can be completed. With the goal of removing this barrier to entry, Dr. Gerbi contracted us, Team Penobscot, to work on SeidarT as a part of our computer science capstone project. The goal of this collaboration is to restructure SeidarT so that it is a standalone application, requiring no additional installations, which operates via a GUI.

The timeline of our capstone project extends two semesters. The first semester is to be allocated for planning, documenting, and understanding the project at hand. During this timeframe, a software requirements specification, system design document, and user interface design document have been produced to describe the project. These documents can be found in appendices D, E, and F respectively. The system requirement specification details what functionalities SeidarT must have at the completion of this project. The system design document details the architecture of the software produced. The culmination of these three documents is detailed here in the critical design review document as a means of describing our processes to any interested in the design or use of SeidarT. For those intending to utilize SeidarT (now or in the future) this document should describe the specific manners in which to navigate the new UI. For those interested in altering or extending the GUI in the future, this document will serve as a primer to the structure and design of our codebase.

As it currently stands, our team has begun prototyping a graphical user interface. The current status is a rudimentary graphical design along with a portion of event handling. The GUI developed has hooked into the SeidarT backend and processes image inputs. Moving forward, we hope to finalize the parsing of user input as well as a continual refinement of the visual components of the graphical interface.

Background

In nature, ice crystals form in hexagonal sheets. As more ice forms, these sheets begin to layer on top of each other. Due to this structure, each ice crystal has a vector norm which is orthogonal to each of the hexagonal sheets. This normal vector is known as the “c-axis” of the crystal. In a realistic setting, when many ice crystals are pressed together they form large glacial structures, but maintain many unique c-axes. These natural structures are called anisotropic glaciers. As a glacier moves, structural variance occurs, known as creep. Creep is caused by glacial slippage and leads to the alignment of c-axes within the glacier. As the c-axes align, the structure of the glacier becomes isotropic and the strength of the glacier becomes very weak.

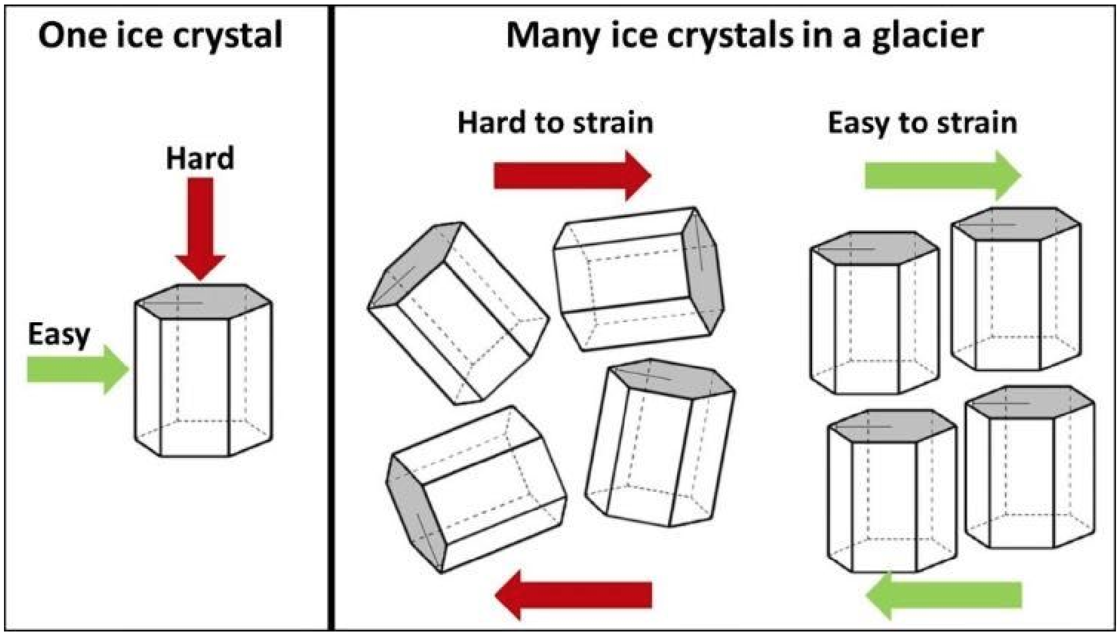


Figure 1. A graphical representation of anisotropic vs. isotropic glaciers

Using anisotropic models when performing glacier calculations leads to imprecisions, as true glaciers have a tendency towards isotropic. Hence a large scale goal of glaciologists is to better understand the seemingly random nature of c-axes in a large glacier. If our understanding could be improved, models would be able to better predict the motion and stability of glaciers, among other quantities. In order to achieve this understanding, glaciologists must study seismic and radar propagation through glaciers to better understand isotropic formations. Executing these experiments requires a large amount of time and effort, as researchers must travel to a glacial field.

In order to mitigate these costs, SeidarT was developed. Drs. Steven Bernsen, Christopher Gerbi, Seth Campbell, Senthil Vel, and Knut Christianson, in conjunction with their graduate and undergraduate students, collaborated to produce a terminal-based seismic and radar propagation modeling software. The models that SeidarT generates are used in order to give researchers approximations on which regions they should perform their experiments. This allows for more tests to be performed per expedition and increases the quality of each data set.

The above discussion offers one specific usage of SeidarT's models, but this is far from its only use. The application can be used for but is not limited to, "investigating, estimating, and imaging englacial ice structure, sub-glacial boundary conditions on the sub-regional scale." In general, it is a powerful asset that can be used for targeted experimentation in glacial regions. Through its modeling abilities, glaciology researchers will be able to obtain better estimates for glacial structures, stability, and motion. These estimates will be used in order to better predict the impacts of global warming on glacial fields, which will have direct societal impacts as sea levels rise.

Purpose

At this point in time, SeidarT is not used as widely as such a potent tool should be. This is in large part due to its relative difficult usage through the command line. Thus, in order to distribute SeidarT for wider use within glaciology research, Dr. Gerbi partnered with our team, Team Penobscot, in order to develop a GUI. This partnership is facilitated through the 2019-2020 University of Maine computer science capstone course curriculum. The goal of this collaboration is for Dr. Gerbi, and his associates, to receive a fully functional graphical user interface so that they might distribute SeidarT, and for Team Penobscot to learn the fundamental concepts of software engineering in a real-world environment.

The remainder of this document serves to describe the specific software engineering processes that are being employed in the development of the SeidarT GUI. This includes the software's requirements, design ideology, testing strategies, and relative dates of deliverable submissions.

Requirements

Functional and Nonfunctional requirements are the clear elicitation of expectations for the software. These requirements were created between Team Penobscot and Dr. Gerbi at the creation of the SRS. As the product has been further clarified, these requirements have been further refined to better describe the precise needs of Dr. Gerbi. Listed below are the abridged functional requirements, the tests used to inspect those functional requirements, and the abridged nonfunctional requirements. While the functional and nonfunctional requirements are only listed briefly here, they are expanded upon further in Appendix D - System Requirements Specification.

Functional Requirements

- FR2: The software shall import models drawn by the user.
- FR3: The software shall receive all necessary inputs from the user through a GUI.

Function Requirement Tests

- Test Case 2.1
  - Correct input is entered into the GUI.
  - The input is saved in a way to be read by the Back-End.
- Test Case 3.1
  - Correct input is supplied to the Back-End.
  - The input is used to generate a graphic.
- Test Case 3.2
  - Incorrect input is supplied to the Back-End
  - An error message is generated to inform the user of the error.

Nonfunctional Requirements

- NFR0: The program shall work without an internet connection.
- NFR3: 90% of users shall be able to generate a graphic with less than an hour of training.
- NFR4: The software should be platform-independent.
- NFR5: The software should be package-independent.

Design

As in any software development process, the overarching structure a developer chooses has a large impact on the ease of design and use for any project. This section will detail the design choices we have made and why we made those choices.

Design Constraints

Our contribution to SeidarT is primarily a graphical user interface. This imposes strict constraints on the input-output structure of our GUI. Firstly, when the user is entering information, they must first select the mode in which the backend will process the given input. This selection will dictate which input fields are available to the user as well as the formatting of the resultant command.

Following user input, the information given by the user must be preprocessed into a precise format that matches what the SeidarT backend accepts.

Further, due to the development of SeidarT being partially completed in Python, Dr. Gerbi has expressed his desire that the GUI developed also be written in Python. Finally, at the completion of this project, the GUI, as well as SeidarT, should be operable via an executable with no further installations required.

Design Approach

The first decision that was chosen was which design pattern we would choose for the application. As the GUI is primarily a pipeline that connects a user to the backend, a reasonable choice was the “Model-view-controller” pattern. In this pattern, there are often 3 primary components: One which processes information, one which displays information, and one which handles information. In this respect, our GUI is the view, the pipeline we develop is the model, and the controller is the existing SeidarT backend.

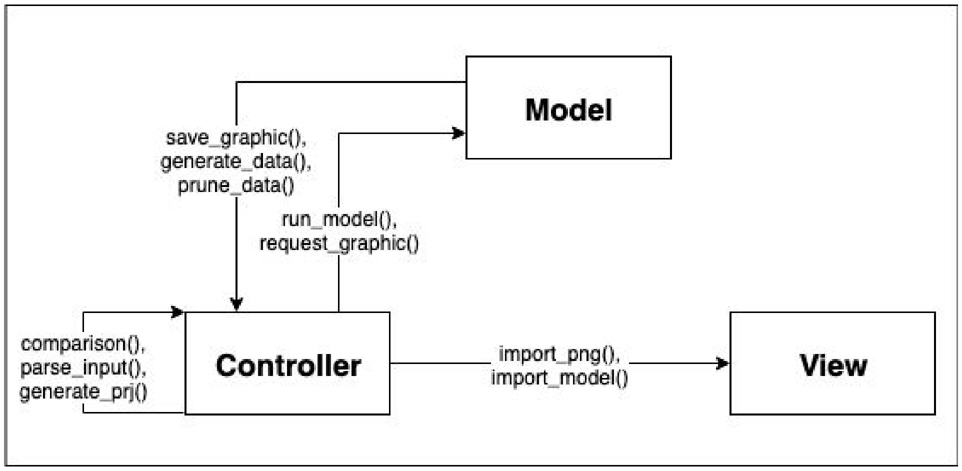
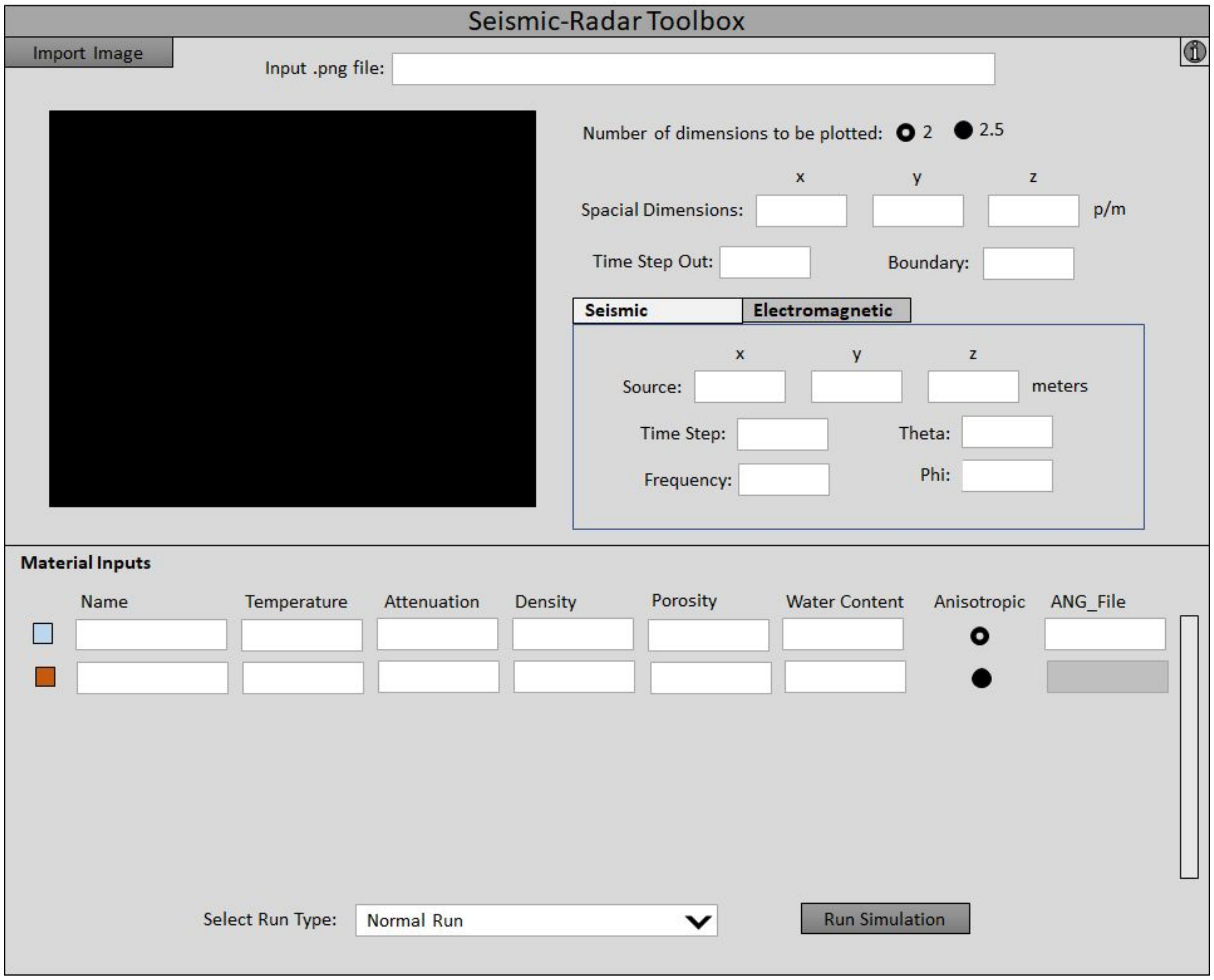


Figure 2. a graphical representation of our design pattern

This model was chosen as the GUI itself naturally serves as the view. All processing we include with the backend for format information and send commands will act as the controller or pipeline. And finally, the SeidarT backend will continue to exist and process all the information we pass to it.

Once this design pattern was chosen, we needed to select a GUI development kit. As two of the primary requirements of this project are such that SeidarT will be cross-platform and the GUI will be written in Python, Kivy was a natural choice. Kivy is a python package that allows for easy compilation, consistent cross-platform styling, and fluid design. Using this toolkit, we have structured our code to be very modular. This was done so that the GUI is fully extensible if the backend ever extends its functionality.

The main goal of our software is to make a GUI that simplifies data entry for the user. To achieve this goal, the GUI is structured in a linear manner. The GUI is designed with the workflow from top to bottom, with clear separated sections to guide the user through its usage. Structuring the design this way will aid in user accessibility, a primary goal of this project. The specific design details are further discussed in Appendix F.



Select Run Type: Normal Run

Figure 3. Preliminary SeidarT GUI

As seen in Figure 3 above, the GUI is well-organized and has no distracting features. The user will go through the prompts on the GUI from top to bottom. Starting by first importing a png file and ending at running the simulation. There is a help button on the top right of the window which allows the user to access the user manual for the software or activate a walk-through tutorial. Both of these windows can be viewed in the UIDD (Appendix F).

Testing Strategy

Current Testing Procedures

During the development of SeidarT's GUI, we will be using TravisCI for continuous integration support. This service will handle both unit testing and integration testing as development continues. Unit tests will be written to manage all functional aspects of our GUI (component generation, input validation, etc..). Integration testing will be focused on two aspects: unit integration and GUI-SeidarT integration. As our software will overlay on top of an existing application, great care will be used during development so that it integrates well with SeidarT's existing backend. Unit integration will ensure that code generated for the GUI will format data consistently and produce expected results when running in sequence. GUI-SeidarT integration will ensure that the command generated by the user via the GUI will produce the expectant figures and project files via the SeidarT backend.

During development, iterative accessibility testing will be done via beta testing to ensure a pleasant UI experience. These beta tests will be performed on each major prototype release. The subject group of these tests will consist of Dr. Gerbi as well as an assortment of people who have a range of experience with SeidarT currently. We are using testers of different experiance levels to ensure the GUI not only facilitates usage by experienced users but also for those who have limited or no experience operating SeidarT currently.

Deliverables

Below is a table of all planned deliverables to be delivered throughout this project. Delivery dates are based on the due dates set by Professor Yoo. Dates for deliverables scheduled to be delivered in the spring semester will be set by the end of the first week of February 2020. To see team contribution on each deliverable please refer to Appendix C of those documents. If any Delivery Dates change this document will be amended with the new date and reason for the change.

Deliverable	Description	Delivery Date
System Requirements Specification	Documentation that describes the software requirements requested by Dr. Gerbi.	Oct 24, 2019
System Design Document	Documentation of the design of the requested software systems.	Nov 12, 2019
User Interface Design Document	Documentation and designs of the GUI for the requested software.	Nov 26, 2019
Critical Design Review	A presentation from our team to the capstone class and product owner describing the software and future plans.	Dec 10, 2019
Critical Design Review Document	This document is the full design of the software to be developed.	Dec 19, 2019
Prototype 1 Release	The first version of the software able to do single-shot runs.	Jan 31, 2020
Prototype 1 Testing	The first round of testing of the software.	Feb 7, 2020
Prototype 2 Release	A second release to refine the UI quality and extend the “run” functionality	Feb 21, 2020
Prototype 2 Testing	The first round of significant beta testing.	Feb 28, 2020
Prototype 3 Release	A further release including the help page and better UI	Mar 6, 2020
Prototype 3 Testing	Another round of UI acceptance testing	Mar 13, 2020
Prototype 4 Release	A penultimate release for further UI refinement	Apr 3, 2020
Prototype 4 Testing	Last round of accessibility testing	Apr 10, 2020
Final Project Report	The final report from our team reviewing the completed project.	Apr 30, 2020
Source Code	Python Files (.py)	Apr 30, 2020
Finished Executable	Standalone Executable (.exe, .APP, and Linux binary)	Apr 30, 2020
Code Inspection Report	A preliminary report of unit testing and the first prototype	TBD
Administrator Manual	Documentation for the product owner in how to maintain and upkeep the software.	TBD
User Manual	Documentation of what the software does and how to use it for the end-user.	TBD

Table 1 - Deliverable breakdown by date

Conclusions

At current, a preliminary graphical interface has been implemented. This interface has hooked into the backend for image processing and .prj file generation. Once an input image has been processed, the UI generates the requisite fields for user input. Further input is not processed as of yet. Our immediate goal is to implement the processing of all user input and allow users to perform *single-shot* runs. The projected completion of this goal is January 31st, 2020. This implementation is being designed in a modular format which will allow for an extension to other run models with relative ease. A secondary goal is to reconfigure SeidarT such that it operates via a single executable. We hope to have this completed by February 21, 2020. Easy execution will facilitate beta testing and is thus a very high priority.

Upon completion of these goals will mark 1.0.0 of the SeidarT GUI.
As described above, following this first release we hope to have iterative releases and beta testing for the remainder of the project. Our hope is to establish a group of beta testers who are willing to participate in the testing for the duration of the Spring 2020 semester.

Glossary

- CDR
    - Critical Design Review
- CDRD
    - Critical Design Review Document
- FR
    - Functional Requirement
- GUI
    - Graphical User Interface
- NFR
    - Non-functional Requirement
- SDD
    - System Design Document
- SeidarT
    - Seismic Radar Toolbox
- SRS
    - Systems Requirement Specification
- UIDD
    - User Interface Design Document

Appendix A - Agreement Between Customer and Contractor

By signing below, the team members understand and agree to complete the functional and non-functional requirements listed and the delivery of the deliverables listed above. By signing below, the customer understands and agrees that the completion of the functional and non-functional requirements listed and the delivering of the deliverables listed above is equivalent to the completion of the software project as a whole.

Should one or both parties desire a change to this document, a new draft of this document will be formed. This new draft will be written through collaboration between both parties, to elicit shortcomings of previous versions of this document and new functional and/or non-functional requirements. This new version will be signed and dated, replacing the previous version of this document only when both parties have signed and dated the document. Should signature dates occur on different days, the new version of the document will replace the previous version on the latest date specified below.

Team Member #1:
Name (Printed): Adam Farrington Date: Dec 18, 2019
Signature: Adam Farrington

Team Member #2:
Name (Printed): Alexander Thacker Date: Dec 18, 2019
Signature: Alexander Thacker

Team Member #3:
Name (Printed): Jens Hansen Date: Dec 18, 2019
Signature: Jens Hansen

Team Member #4:
Name (Printed): Nathan Gazey Date: Dec 18, 2019
Signature: Nathan Gazey

The Customer:
Name (Printed): ______________ Date: ________
Comments:______________________________
____

Signature: ______________________________

Appendix B – Team Review Sign-off

By signing below all team members agree they have reviewed this document. Signing below, team members agree to all content in this document aside from any comments in the space provided below. Signing below, team members agree that the format used in this document is agreeable aside from any comments in the space provided below. Singers acknowledge that the comment area below is not a place to voice major points of contention, only minor points of disagreement in this document.

Team Member #1:
Name (Printed): Adam Farrington Date: Dec 18, 2019
Comments:

Signature: Adam Farrington

Team Member #2:
Name (Printed): Alexander Thacker Date: Dec 18, 2019
Comments:

Signature: Alexander Thacker

Team Member #3:
Name (Printed): Jens Hansen Date: Dec 18, 2019
Comments:

Signature: Jens Hansen

Team Member #4:
Name (Printed): Nathan Gazey Date: Dec 18, 2019
Comments:

Signature: Nathan Gazey

Appendix C - Team Contributions

- Adam - 20%
  - Requirements Section
  - Review and acceptance of all sections
- Alex - 20%
  - Part of Design Approach
  - Updated GUI Diagram
  - Deliverables
  - General document editing, formatting, revising
- Jens - 40%
  - Executive Summary
  - Background
  - Contributed on Requirements Sections
  - Design Constraints and Approach
  - Testing Strategy
  - Organized and expanded deliverables
  - Conclusion
  - General document editing, formatting, revising
- Nathan - 20%
  - Reviewed entire document
  - Formatted and imported all prior documents
  - Testing Strategy
  - Assisted with Deliverables
  - Architectural design contributions

Appendix D - System Requirements Specification

1. Introduction

1.1 Purpose of This Document

This is the formal agreement between Dr. Gerbi and Team Penobscot that describes in detail the product to be delivered. Dr. Gerbi has requested assistance in designing and building the front end of a program that he has been working on with his colleagues. Included in this document are the description, scope, and agreed-upon requirements of the desired software. This document also describes the timeline on which Team Penobscot will produce the desired software. This timeline will describe all documents that will be generated and software artifacts that will be produced.

1.2 References

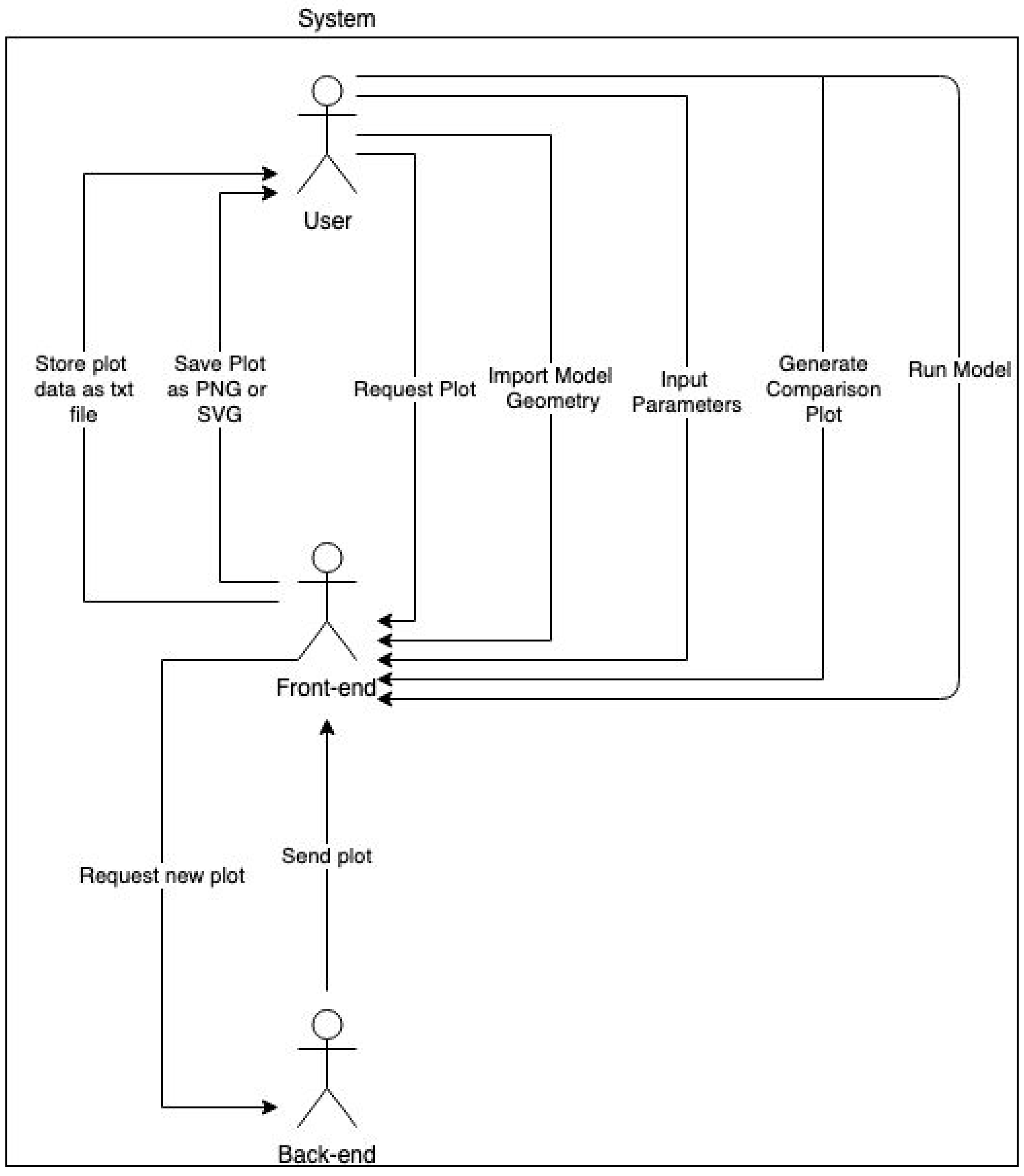
1. A slideshow describing the purpose of the backend software.
2. The software generates cross-platform executables out of python code.

1.3 Purpose of the Product

Dr. Christopher Gerbi is a professor for the School of Earth and Climate Sciences at the University of Maine. Dr. Gerbi, along with Dr. Seth Campbell, Ann Hell, and Steven Bernsen, has been developing backend software that is able to predict radar and seismic signals from simulated survey sites. This software is nearing completion and currently is operated through the use of command-line inputs. This is not an ideal interface for the software. Thus, Dr. Gerbi has partnered with the 2019 computer science capstone course at the University of Maine to create a Graphic User Interface (GUI) for new software. This GUI should be able to simplify the use of the program and allow for widespread distribution to the academic community.

1.4 Product Scope

The system being produced will act as a direct link from the existing backend software and the user. The diagram below shows the primary connections that will exist between each of the actors (being the user, backend and frontend).



2. Functional Requirements

This section of the document outlines all functional requirements derived from the user stories generated jointly by the customer (Dr. Gerbi) and the development team (Team Penobscot). Each use case will map to one or more test cases that will mark the completion of that requirement.

2.1 Use Cases

Number	1	
Name	Import Model Geometry	
Summary	The User imports a model geometry (.png) into the GUI.	
Priority	5	
Preconditions	A plot is stored locally as a png.	
Postconditions		
Primary Actor	User	
Secondary Actors	Front-End	
Trigger	The user clicks the “Import Model Geometry” button.	
Main Scenario	**Step**	**Action**
	1	Front-End prompts the user to enter a model geometry image.
	2	Front-end imports the model geometry for usage.
Open Issues	N/A	

Number	2	
Name	Import Image	
Summary	The user imports a png file which will be used to construct a prj file.	
Priority	5	
Preconditions		
Postconditions	An image is stored locally as a png or svg, and an incomplete plot has been created.	
Primary Actor	User	
Secondary Actors	Front-End	
Trigger	A new dat/csv output is requested.	
Main Scenario	**Step**	
	1	Front-End prompts the user to input a png or svg.

	2	A png or svg file is received by the Front-End.
	3	An incomplete prj file is created with information acquired from the image.
Open Issues	N/A	

Number	3	
Name	Input Parameters	
Summary	The User inputs into the Front-End the parameters of the survey being requested.	
Priority	5	
Preconditions	An incomplete plot has been created.	
Postconditions	All parameters are stored by the Front-End, ready to be sent to the Back-End	
Primary Actor	User	
Secondary Actors	Front-End	
Trigger	A new dat/csv output is requested.	
Main Scenario	**Step**	**Action**
	1	Front-End prompts the user to input parameters.
	2	Parameters are saved by the Front-End to a csv file.
Open Issues	N/A	

Number	4	
Name	Run Plot	
Summary	A graphic is generated at the request of the User.	
Priority	5	
Preconditions	Parameters have been input by the User. Model data has been generated.	
Postconditions	A new graphic has been generated by the Back-End, to be displayed.	
Primary Actor	Front-end	
Secondary Actors	Back-end	
Trigger	The User clicks the “run plot” button.	
Main Scenario	**Step**	**Action**
	1	The user selects which run mode they wish to use.
	2	The plot information and run type is sent to the backend
	3	The backend produces the appropriate .dat files and graphics
Extensions	**Step**	**Branching Action**
	1a	The information given to generate the plot is invalid**:**

		An error message is displayed to the User describing the error.
	3a	Back-End performs a Single Shot Run
	3b	Back-End performs a Single Shot Run
	3c	Back-End performs a Common Offset Run
	3d	Back-End performs a Common Midpoint Run
Open Issues	N/A	

Number	5	
Name	Preview Plot metadata	
Summary	A plot has been generated and its information will be displayed for the user	
Priority	4	
Preconditions	A plot has already been created.	
Postconditions	The relevant plot information will be displayed to the user.	
Primary Actor	Front-End	
Secondary Actors	N/A	
Trigger	The user requests a plot	
Main Scenario	**Step**	**Action**
	1	The user requests a plot from local storage.
	2	The defining information about the plot is loaded into the GUI for review.
Extensions	**Step**	**Branching Action**
	1a	The desired plot does not exist in local storage**:** An error message is displayed to the User describing the error.
Open Issues	N/A	

Number	6
Name	Generate a comparison plot
Summary	The user supplies two plot files that have been generated and the differences between them.
Priority	3
Preconditions	The user has generated at least two plots.
Postconditions	The GUI will generate a new plot corresponding to the difference between the previous two
Primary Actor	User
Secondary Actors	Front-end

Trigger	The user clicks the “Compare plots” button	
Main Scenario	**Step**	**Action**
	1	The application takes the difference of the two csv files.
	2	The application stores the new difference plot as a csv.
Open Issues	N/A	

Number	7	
Name	Generate a Graphic	
Summary	A graphic is saved locally for later retrieval	
Priority	4	
Preconditions	The backend has generated a plot the User wishes to save.	
Postconditions	The graphic will be saved and stored locally to a location chosen by the User.	
Primary Actor	User	
Secondary Actors	Front-End	
Trigger	The user clicks a “Save PNG/SVG” button.	
Main Scenario	**Step**	
	1	A prompt appears prompting the user to where the plot should be saved and what format.
	2	The Plot is saved in the location specified as either a png or svg.
Open Issues	N/A	

2.2 Test Cases

Test Cases outline our expected measure of success for the use cases above.

- Test Case 1.1
  - A correct plot is imported into the GUI.
  - The plot is saved locally through the GUI.
- Test Case 2.1
  - Correct input is entered into the GUI.
  - The input is saved in a way to be read by the Back-End.
- Test Case 3.1
  - Correct input is supplied to the Back-End.
  - The input is used to generate a graphic.
- Test Case 3.2
  - Incorrect input is supplied to the Back-End
  - An error message is generated to inform the user of the error.
- Test Case 4.1
  - An existing plot is requested to be displayed.
  - The defining information about the plot is displayed through the GUI.
- Test Case 4.2
  - A plot that does not exist is requested to be displayed.
  - An error message is generated to inform the user of an error.
- Test Case 5.1
  - Two existing plots are requested to be displayed.
  - A plot difference is generated and displayed to the GUI.
- Test Case 6.1
  - A plot is requested to be saved to a local file location.
  - The plot is saved correctly, able to be loaded by the GUI.

3. Non-Functional Requirements

Non-functional requirements were derived from the user stories agreed upon between the team and the customer. They will ensure the reliability, performance, and availability of the software system.

- NFR0
    - Priority: 5
    - The program shall work without an internet connection.
    - Tests for system and acceptance testing
        - SD0.1
            - Produce a graphic while disconnected from all networks.
- NFR1
    - Priority: 5
    - Data entered by the user shall be displayed accurately within the GUI.
    - Tests for system and acceptance testing
        - SD1.1
            - Upon importing data from a plot file, the data fields should accurately represent that data.
- NFR2
    - Priority: 5
    - The source code for the project must be open source under an appropriate license.
    - Tests for system and acceptance testing
        - SD2.1
            - The repository containing all requisite source code will be accessible while not logged into GitHub.
- NFR3
    - Priority: 4
    - 90% of users shall be able to generate a graphic with less than an hour of training.
    - Tests for system and acceptance testing
        - SD3.1
            - Testing can be done on random participants who have never used the software before. They should be able to generate a graph in less than 10 minutes after no more than 50 minutes of instruction 90% of the time.
- NFR4
    - Priority: 5
    - The software should be platform-independent.

    - Tests for system and acceptance testing
        - SD4.1
            - Testing can be done by running the program on different operating systems (Windows 10, Mac OS, Linux) and multiple versions of each operating system.
- NFR5
    - Priority: 5
    - The software should be package-independent.
    - Tests for system and acceptance testing
        - SD4.1
            - Testing can be done by downloading the software onto an arbitrary computer and have no required installations for the application to operate.
- NFR6
    - Priority: 4
    - Performance should maintain visual fidelity for at minimum 95% of all user interactions.
    - Tests for system and acceptance testing
        - SD5.1
            - During regular use, the GUI should respond to all user input (clicks, button presses, and keyboard input) within 3 seconds 90% of the time.
- NFR7
    - Priority: 2
    - The software should be able to process requests for a parameter sweep of up to ten values.
    - Tests for system and acceptance testing
        - SD6.1
            - Test by entering information for 10 distinct plots and verifying that the resultant graphics are produced.
- NFR8
    - Priority: 5
    - The users should never be allowed to update the default material tensors.
    - Tests for system and acceptance testing
        - SD8.1
            - Testing can be done by running the program and manually entering the tensor data then restart the program and see if the default value has changed.
- NFR9
    - Priority: 5
    - The program will export data to the appropriate location without corruption 99% of the time.
    - Tests for system and acceptance testing

        - SD9.1
            - This can be tested by running a minimum of 20 exports to ensure more than 99% of exports succeed.
- NFR10
    - Priority: 3
    - The programs’ source code shall be organized in compliance with Git flow to ensure outside collaborators can easily access the code.
    - Tests for system and acceptance testing
        - SD10.1
            - Git repository must be validated with proper git-flow branching upon each minor release.

4. User Interface

See “User Interface Design Document for Seismic-radar toolbox GUI.”

5. Deliverables

All deliverables will be posted on Team Penobscot's Github Repository by the stated delivery date. The stated dates are set deadlines by Professor Yoo. The dates for the administrator and user manuals have not yet been announced as they are artifacts of the spring component of this course. The final project report, source code, and executable have been set to the last day of classes of the spring 2020 semester. Any deadlines listed for May of 2020 are subject to change at the discretion of Dr. Yoo. Any changes will be announced sometime in January of 2020 and will not change the dates by more than two weeks.

Deliverable	Format	Delivery Date
System Requirements Specification	PDF	October 24, 2019
System Design Documents	PDF	November 12, 2019
User Interface Design Document	PDF	November 26, 2019
Critical Design Review	Presentation	December 12, 2019
Critical Design Review Document	PDF	December 19, 2019
Administrator Manual	PDF	TBD
User Manual	PDF	TBD
Final Project Report	PDF	May 1, 2020
Source Code	Python Files (.py)	May 1, 2020
Finished Executable	Standalone Executable (.exe, .app, and Linux binary)	May 1, 2020

6. Open Issues

Issues that have been raised and do not yet have a conclusion. These issues will be addressed later in the development process.

GitHub Issue Number	**Short Description**	**Expected Resolution Date**
#1	Software Requirement Specification	October 24, 2019

Appendix E - System Design Document

1. Introduction

1.1 Purpose of This Document

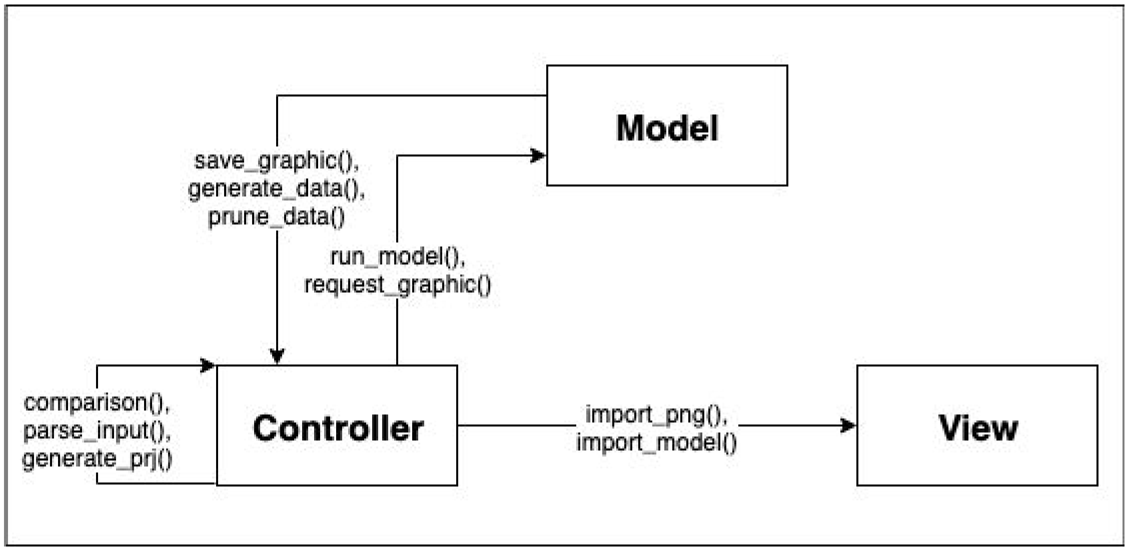
This is the formal agreement between Dr. Gerbi and Team Penobscot that describes in detail the design of the system being developed by Team Penobscot. This project is a user interface for a software system, SeidarT, currently accessed through command lines only. Included in this document are the proposed system architecture, plans for persistent data, and a summarizing table comparing system requirements which the corresponding architectural components. The remainder of this document is used to describe the plans for how this interface will be designed.

1.2 References

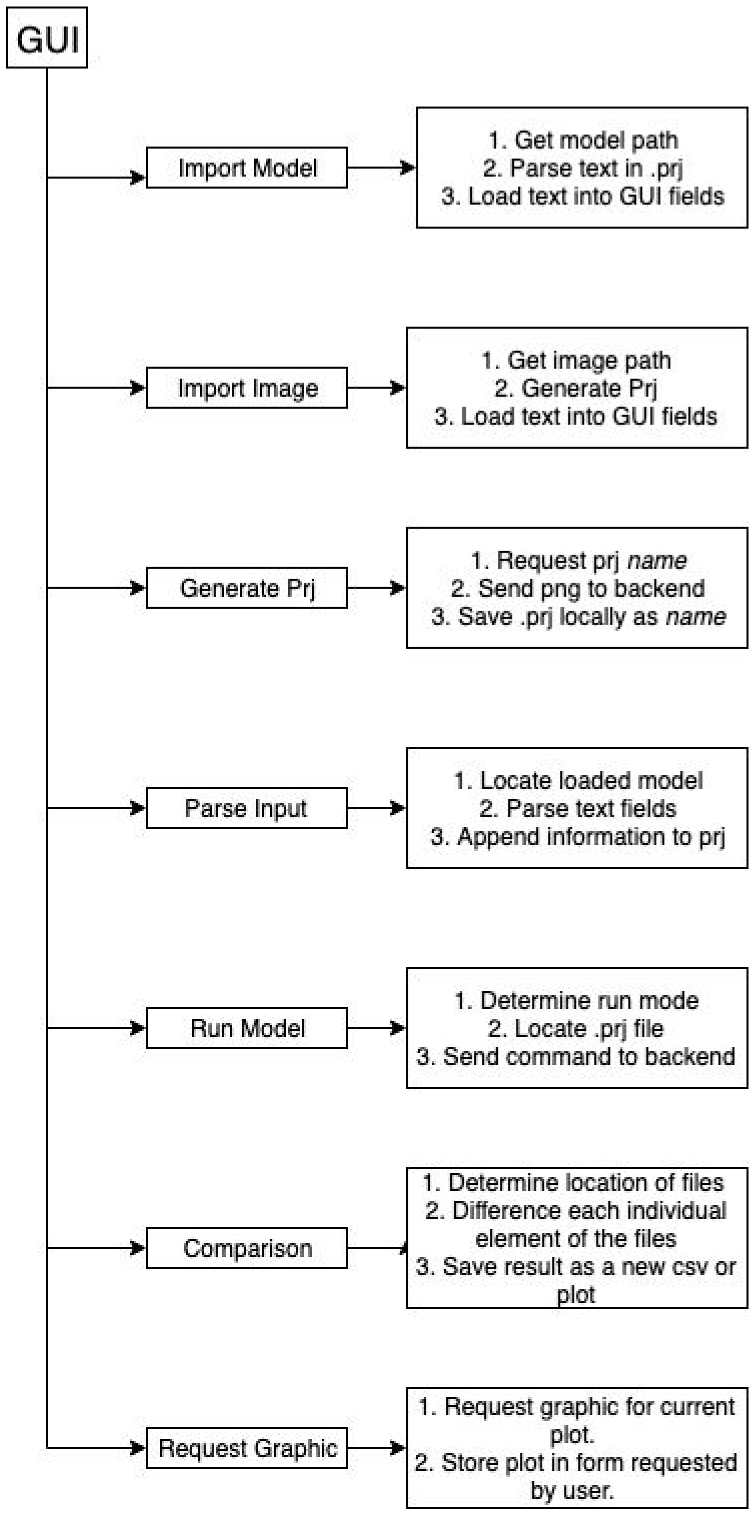
1. A slideshow describing the purpose of the backend software.
2. A GitHub Repository of the backend software.
3. Team Penobscot's own Software Requirements Specification.
4. Team Penobscot's own User Interface Design Document.

2. System Architecture

2.1 Architectural Design


Model
save_graphic(), generate_data(), prune_data()
run_model(), request_graphic()
comparison(), parse_input(), generate_prj()
Controller
import_png(), import_model()
View

2.2 Decomposition Description


GUI
Import Model
1. Get model path
2. Parse text in .prj
3. Load text into GUI fields
Import Image
1. Get image path
2. Generate Prj
3. Load text into GUI fields
Generate Prj
1. Request prj *name*
2. Send png to backend
3. Save .prj locally as *name*
Parse Input
1. Locate loaded model
2. Parse text fields
3. Append information to prj
Run Model
1. Determine run mode
2. Locate .prj file
3. Send command to backend
Comparison
1. Determine location of files
2. Difference each individual element of the files
3. Save result as a new csv or plot
Request Graphic
1. Request graphic for current plot.
2. Store plot in form requested by user.

For our architecture, we are using a Model-View-Controller pattern. This pattern best serves our purpose as we are creating a graphical user interface for an existing already created backend used for data processing. The backend will serve as the model in this interpretation.

Our Graphical User Interface will split into the View and Controller portion of the MVC pattern. The View will reflect all user-supplied parameters for a given plot. The Controller will take parameters and model geometry provided by the user and send it to the backend for processing.

3. Persistent Data Design

3.1 Database Descriptions

The software requested does not require the use of a database.

3.2 File Descriptions

This application operates using or generating seven different files.

1. Model control files (prj): These are text files that represent a model that is to be run (at some point) on the backend to generate data files.
2. Input images (png): These are graphics supplied to the front end in order to generate the .prj files.
3. Meta Data (.txt): Files which contain information about the run parameters for a specific instance of running a model file on the backend.
4. Fortran Binaries (.DAT): These are artifacts generated by the backend which are stored locally. Users have no access nor need to access these files.
5. Export graphics (.svg): A graphic produced by running a plot on the backend.
6. Export graphics (.png): A graphic produced by running a plot on the backend
7. Resultant CSV (.txt): A subset of the information contained within the .DAT files.

4. Requirements Matrix

Below is a table showing the relation between the use cases stated in the System Requirements Specification document and the functions/methods that satisfy those requirements. Descriptions of the functions are listed in the following section.

Functional Requirements:	Relating Functions:

1. Import Model	import_model()
2. Import Image	load_png(), generate_prj()
3. Input Parameters	parse_input()
4. Run Model	
4a. Single Shot Run	parse_input(), run_model()
4b. Wide Angle Run	
4c. Common Offset Run	
4d. Common Midpoint Run	
5. Preview Plot Metadata	import_model()
6. Generate a Comparison Plot	comparison()
7. Generate a Graphic	generate_graphic()

4.1 Function Descriptions

- import_model:
  - This function allows the user to import information contained within a .prj file to be loaded into the appropriate fields within the GUI.
- import_png()
  - The user will be directed to import a png file which will be used to construct a .prj file.
- generate_prj
  - Passes the image to the backend which generates the template for a .prj file to be expanded later.
- parse_input:
  - This function will take the user-supplied information and add it to the .prj file currently imported.
- run_model:
  - run_model will operate in 4 primary modes: single shot, wide-angle, common offset, and common midpoint. Each of these will take as a parameter a finalized .prj file. The mode will be selected by the user via the GUI and will be performed accordingly.
- comparison:

    - This function will take two model output sets (.dat or .csv) as inputs. It will then, value by value, take the difference of each pixel region. The result of this differencing will then be stored as a third plot for later usage.
- generate_graphic()
    - Based on pruned CSV text data, a graphic is produced by the backend and stored locally on the host’s machines. They have the option of storing it as a .png or as a .svg

Appendix F - User Interface Design Document

1. Introduction

1.1 Purpose of This Document

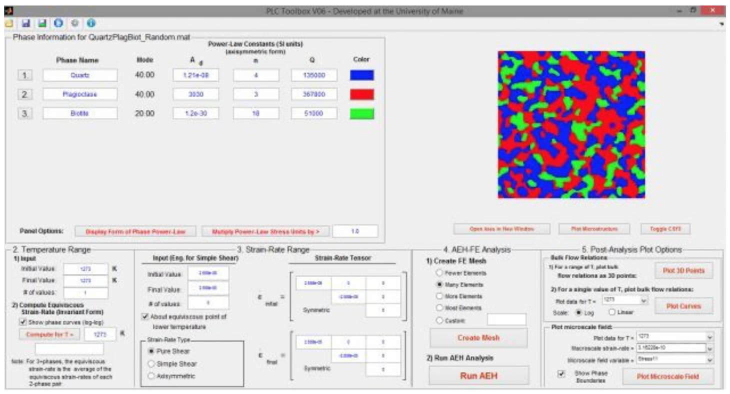
This is a formal agreement between Dr. Gerbi and Team Penobscot that describes the interface design of the system being developed by Team Penobscot. This project is a user interface for a software system, SeidarT, currently accessed through command lines only. This document will specify how a user will navigate between the various pages, the standards used in designing the GUI, as well as how user input will be validated by the interface.

1.2 References

1. Another application used by Dr. Gerbi and his associates. This will be used as a reference for the design of our interface, as the applications should be similar to draw upon the familiarity.
2. A Github repository which contains all relevant code and documentation regarding this project.
3. Team Penobscot's SRS, found in the above Github repository.
4. Team Penobscot's SDD, found in the above Github repository.

2. User Interface Standards

Dr. Gerbi supplied our team with an example of similar software to that which we will be designing the GUI for. This example can be viewed below and will be used as a guide for the new GUI, keeping the general theme between the two as similar as possible.


PLC Toolbox V06 - Developed at the University of Maine

Phase Information for QuartzPlagBiot_Random.mat

Power-Law Constants (SI units) (axisymmetric form)

	Phase Name	Mode	A_d	n	Q	Color
1.	Quartz	40.00	1.21e-08	4	135000	🟦
2.	Plagioclase	40.00	3030	3	367800	🟥
3.	Biotite	20.00	1.2e-30	18	51000	🟩

Panel Options: Display Form of Phase Power-Law | Multiply Power-Law Stress Units by > 1.0

Open Axes in New Window | Plot Microstructure | Toggle CSYS

2. Temperature Range
1) Input
Initial Value: 1273 K
Final Value: 1273 K
# of values: 1
2) Compute Equiviscous Strain-Rate (Invariant Form)
[✓] Show phase curves (log-log)
Compute for T = 1273 K
Note: For 3+phases, the equiviscous strain-rate is the average of the equiviscous strain-rates of each 2-phase pair.

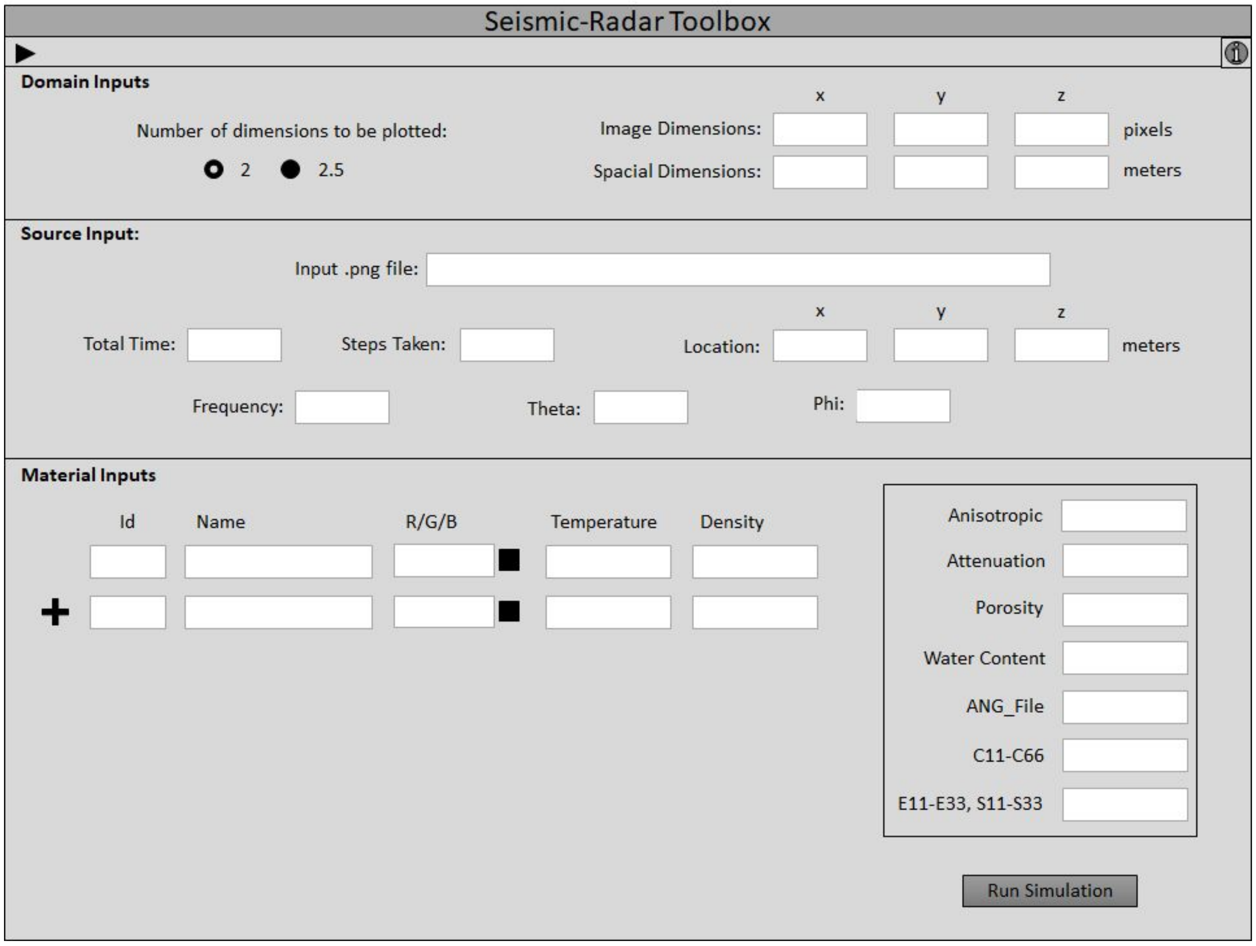
3. Strain-Rate Range
Input (Eng. for Simple Shear)
Initial Value: [illegible]
Final Value: [illegible]
# of values: 1
[✓] About equiviscous point of lower temperature
Strain-Rate Type
[✓ Pure Shear] Simple Shear Axisymmetric

Strain-Rate Tensor
$\dot{\varepsilon}_{initial}$ = [illegible] 0 0 / [illegible] 0 / Symmetric 0
$\dot{\varepsilon}_{final}$ = [illegible] 0 0 / [illegible] 0 / Symmetric 0

4. AEH-FE Analysis
1) Create FE Mesh
Fewer Elements [✓ Many Elements] More Elements Most Elements Custom
Create Mesh
2) Run AEH Analysis
Run AEH

5. Post-Analysis Plot Options
Bulk Flow Relations
1) For a range of T, plot bulk flow relations as 3D points: Plot 3D Points
2) For a single value of T, plot bulk flow relations:
Plot data for T = 1273
Scale: [✓ Log] Linear
Plot Curves
Plot microscale field:
Plot data for T = 1273
Macroscale strain-rate = 3.16228e-10
Microscale field variable = Stress11
[✓] Show Phase Boundaries
Plot Microscale Field

Below is the basic GUI design that our team has come up with. We will maintain a design similar to the example given above. The layout of the GUI is meant to be procedural in nature so users have simple steps to follow to use the program. As part of this procedure, there will be input verification to ensure valid input. Minor color and layout tweaks may be needed as development proceeds but they will be discussed with Dr. Gerbi as they arise. The input areas are separated into different sections based on data validation, listed in section 4. Domain Inputs are for the general conditions of the world in which the simulation will be in. Source Input is where the .png will be added, along with other key parameters. Lastly, the Material Inputs covers all material being analyzed in the simulation and allows adjustments to all aspects of each individual material. Only the most important aspects of a given material will be displayed in each row, Initially, upon entering an id, or name, the rest of the boxes will automatically be populated with default values for that material. The information button on the top right of the screen will pop up the option to open the user manual or the walk-through process.



Seismic-Radar Toolbox

Domain Inputs

Number of dimensions to be plotted: ○ 2 ● 2.5

	x	y	z	
Image Dimensions:				pixels
Spacial Dimensions:				meters

Source Input:

Input .png file:

Total Time: Steps Taken:

	x	y	z	
Location:				meters

Frequency: Theta: Phi:

Material Inputs

Id	Name	R/G/B	Temperature	Density
		■		
		■		

Anisotropic

Attenuation

Porosity

Water Content

ANG_File

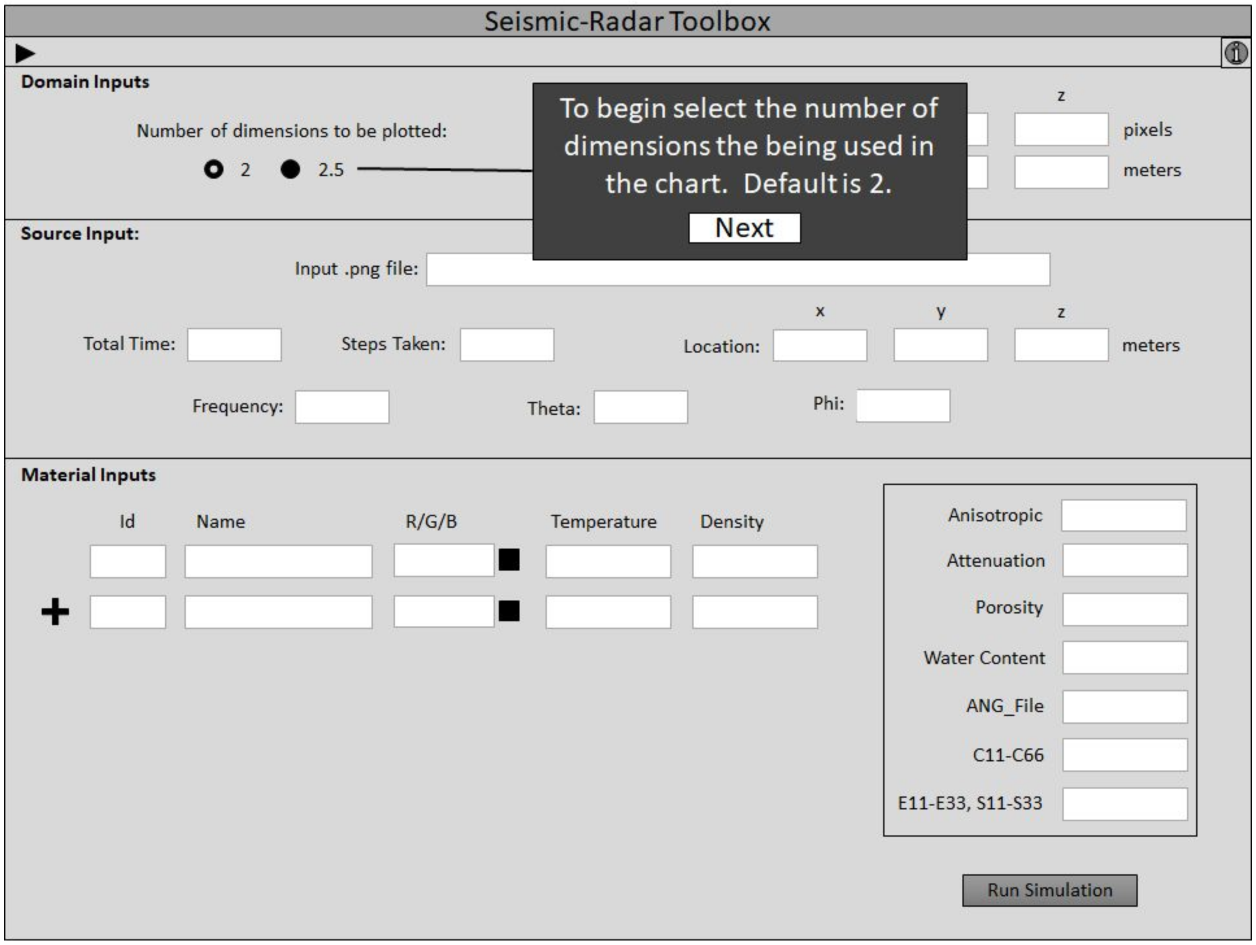
C11-C66

E11-E33, S11-S33

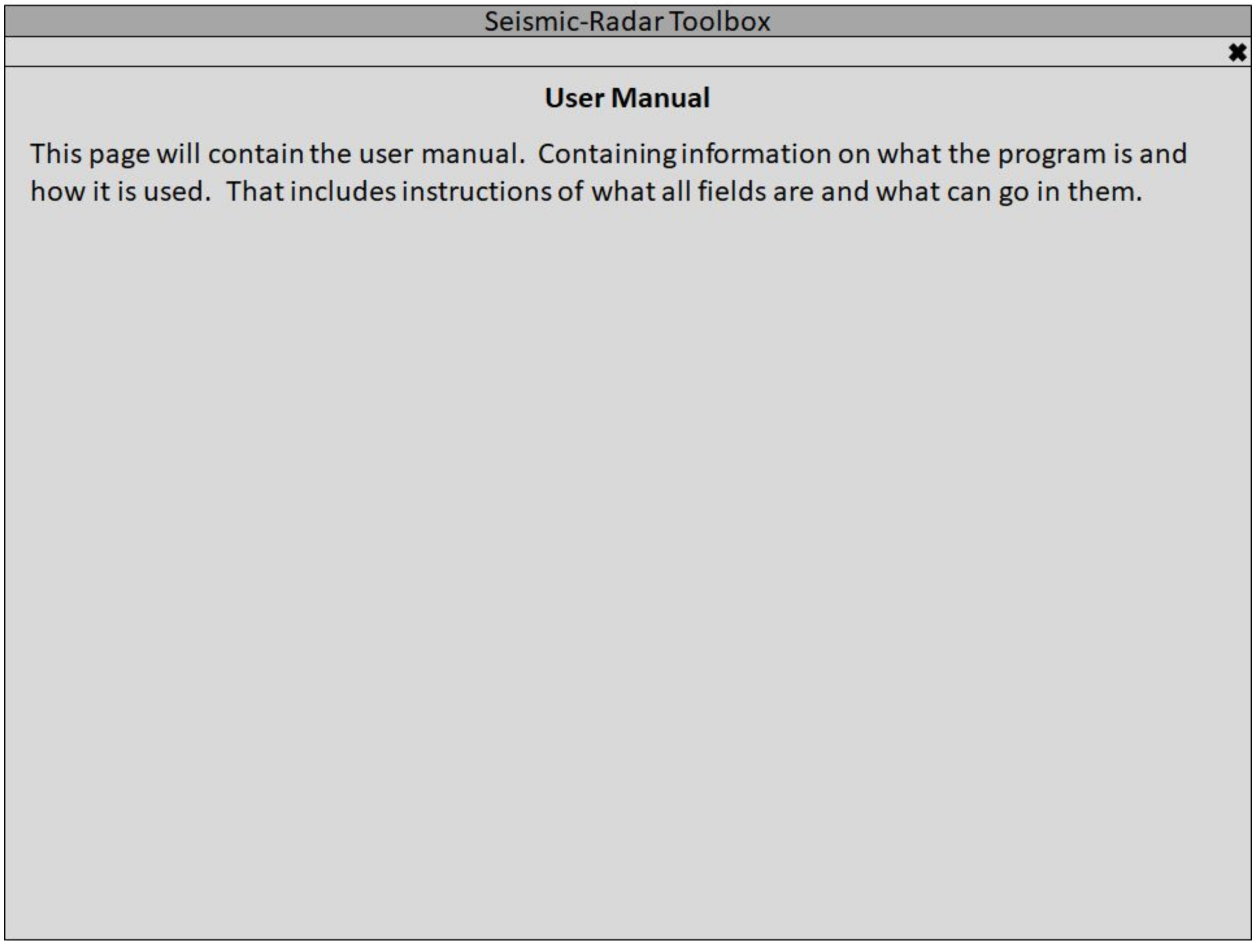
Run Simulation

3. User Interface Walkthrough

The below image is an example of a step in the walkthrough process. It will pop up a message to the user with a line to the field in which it is discussing. The message will inform the user what is needed to be done to that field and suggested values if defaults are involved. Once the desired field has valid input and the user has clicked next (or pressed the tab/enter key), the current prompt will disappear. A new prompt will then appear directing the user on the next step of the process.


To begin select the number of dimensions the being used in the chart. Default is 2.

The last page of the GUI will be the User Manual. This is just a place holder at the moment due to the fact that the user manual will not be created until next semester. The x at the top right of the screen will return the user to the home page.

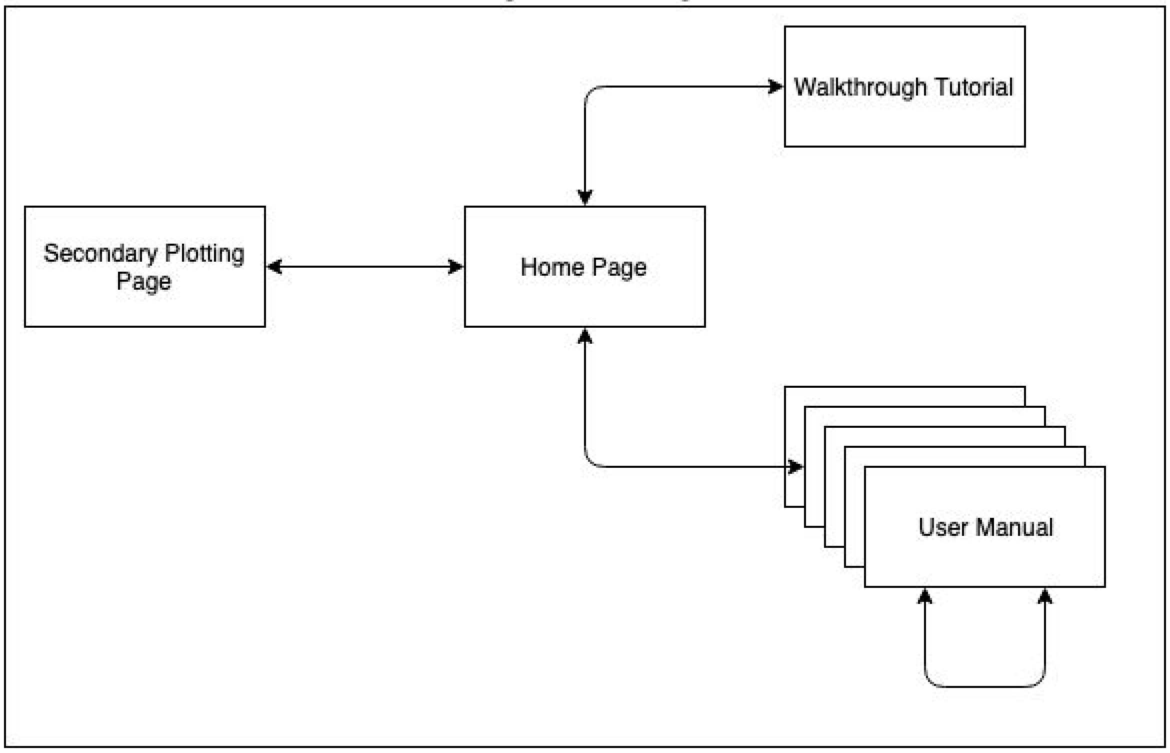


User Manual

This page will contain the user manual. Containing information on what the program is and how it is used. That includes instructions of what all fields are and what can go in them.

Below is a brief diagram that describes the various pages and where one might go from that page. An arrow represents the ability to move from one page to another. Note: As the user walkthrough pages are sequential, one has the ability to move only from each step in the walkthrough to the next or previous.

Page Traversal Diagram



4. Data Validation

Listed below are all pieces of information ascertained by the user interface to generate a .prj file. Some information is obtained automatically by sending a .png file to the system, however, most information is acquired by the user filling in input fields. Data types are modeled after Fortran data types, as Fortran is the language used by the part of the system that uses a .prj file.

Domain Inputs		User Inputs regarding the inputted image.	
Name	Data Type	Limits/Format	Description
dim	STR	Either ‘2’ or ‘2.5’	
nx, ny, nz	INT	N/A	The dimensions of the image.

dx, dy, dz	REAL	N/A	The spatial step size for each dimension in meters.
Material Inputs		User Inputs regarding each unique R/G/B value found in the inputted image.	
Name	Data Type	Limits/Format	Description
id	INT	N/A	The identifier which has been given to each unique material.
R/G/B	STR	0-255	R/G/B value of the material.
Temperature	REAL	Celsius	The temperature of the material.
Attenuation	REAL	N/A	Attenuation length of the material.
Density	REAL	kg/m^3	The density of the material.
Porosity	REAL	Percentage	The porosity of the material.
Water_Content	REAL	Percentage	Percentage of pores that contain water.
Anisotropic	BOOL	True or False	Whether the material is anisotropic (True) or isotropic (False).
ANG_File	STR	N/A	If Anisotropic is True then the full path to the .ang file is supplied. The .ang file is a delimited text file that contains the 3-by-n array of Euler rotation angles in radians.
C11-C66	REAL	N/A	The Stiffness coefficients of the material.
E11-E33, S11-S33	REAL	N/A	The permittivity and conductivity coefficients.
Source Inputs		User Inputs regarding the source of the seismic or electromagnetic activity.	
Name	Data Type	Limits/Format	Description
dt	REAL	N/A	dx/(2*max velocity)

steps	INT	N/A	The total number of time steps.
x,y,z	REAL	N/A	Locations in meters. +z is down, +y is into the screen.
f0	REAL	N/A	Center frequency for the Gaussian pulse function if source_file isn’t supplied.
theta	REAL	N/A	Source orientation in the x-z plane.
phi	REAL	N/A	Source orientation in the x-y plane. Only used if dim is ‘2.5’.
source_file	STR	N/A	The pointer to the text file that contains the source time series as a steps-by-1 vector.