Seismic-radar toolbox GUI

Final Report

Dr. Chris Gerbi

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

May 5, 2020
Final Report, Version 1.0.0

1. Introduction **4**

2. Process Review **4**

Pair Programming 4

UI Design 4

3. The Bumps Along the Way **5**

Legacy Code 5

The Elephant in the Room 5

4. Requirements Review **6**

Appendix A – Team Review Sign-off **7**

Appendix B - Team Contributions **8**

Appendix C - System Requirements Specification **9**

1. Introduction **9**

Purpose of This Document 9

References 9

Purpose of the Product 9

Product Scope 10

2. Functional Requirements **11**

Use Cases 11

Test Cases 15

3. Non-Functional Requirements **16**

4. User Interface **19**

5. Deliverables **20**

6. Open Issues **21**

Appendix D - System Design Document **22**

Introduction **22**

Purpose of This Document 22

References 22

System Architecture **22**

Architectural Design 22

Decomposition Description 23

Persistent Data Design **25**

Database Descriptions 25

File Descriptions 25

Requirements Matrix **25**

4.1 Function Descriptions 26

Appendix D - User Interface Design Document **28**

1. Introduction **28**

1.1 Purpose of This Document 28

References 28

User Interface Standards **28**

User Interface Walkthrough **31**

Data Validation **33**

Appendix E - Code Inspection Review **36**

1. Introduction **36**

1.1 Purpose of This Document 36

References 36

Coding and Commenting Conventions 36

Defect Checklist 36

Figure 1 - Defect Table by Type 36

2. Code Inspection Process 37

2.1 Description 37

2.2 Impressions of the Process 38

Inspection Meetings 39

3. Modules Inspected **39**

3.1 SeidarTGUI.py Imports 39

3.2 getImage() 39

3.3 run() 39

3.4 build() 40

3.5 Kivy File 40

4. Defects **40**

Figure 2 - In Depth Table of Defects by Module 40

Appendix F - Administrator Manual **43**

1. Introduction 43

1.1 Purpose of This Document 43

1.2 References 43

2. System Overview 43

2.1 Background 43
2.2 Hardware and Software Requirements 43
3. Administrative Procedures 44
3.1 Installation 44
3.2 Routine Tasks 44
3.3 Periodic Administration 44
3.4 User Support 45
4. Troubleshooting 45
4.1 Dealing with Error Messages and Failures 45
4.2 Known Bugs and Limitations 45

Appendix G - User Guide 46
1. Introduction 46
1.1 Purpose of This Document 46
1.2 Intended Readership 46
1.3 Applicability Statement 46
1.4 Purpose 46
1.5 How to Use this Document 46
1.6 Related Documents 47
1.7 Problem Reporting Instructions 47
2. Overview 47
3. Instructions 47
PRJ Info 48
PRJ Run 50
PRJ Plots 51
4. Error Messages and Recovery Procedures 51

1. Introduction

The software has been sent off to Dr. Christopher Gerbi as of May 5th, 2020. While we have yet to formally have a meeting with Dr. Gerbi's graduate workers, it is safe to say that the professional work performed by Team Penobscot for Dr. Christopher Gerbi is complete. Warts and all, the SeiDarT software delivered to the client is what we have delivered. Now that it's out of our hands, it's finally apt for us to take a look back on what we've done. This isn't a time to put on our rose-tinted glasses, this is a time for the team to truly reflect on what worked and what did not.

2. Process Review

Once we were tasked with creating a GUI for Dr. Gerbi's SeidarT software, we designated roles to each of the team members. Adam and Jens would be tasked with core system development through pair programming. Alex was tasked with everything related to the graphical design of the GUI. While Nathan scaffolded for the project and provided help when needed.

Pair Programming

Adam and Jens did a bulk of the software development through the process of Pair Programming. The pair did weekly meetings to work on the code in tandem, with Jens taking the primary coding role with Adam working as support and debugging. The pair completed a majority of the work in the five sprints where Team Penobscot was on campus and able to meet in person. From there, Jens and Adam worked separately to complete the work assigned to them.

Pair Programming was an effective way to get Adam, the less experienced programmer on the team up to speed. However when campus was shut down and Team Penobscot worked from home, Pair Programming became an obstacle more than a benefit to us. Working from home just does not intuitively work with the Pair Programming method. Work became slow after the quarantine began for multiple reasons, with Pair Programming contributing to our hesitance to get back to work.

UI Design

Alex designed and implemented the graphical user interface for the SeidarT software. Initially creating a preliminary design based on inputs stated in the software guide that the team was supplied by Dr. Gerbi. He had also supplied the team with an example of a similar program and pointed out design features from it that he would like in the SeidarT GUI. This design went through a few changes based on bi-weekly feedback from Dr. Gerbi during our weekly meetings. Once development began Jens and Adam created the required input elements based off of the preliminary gui design. Once the objects were created they were given to Alex to configure their graphical properties through the kv file. These properties determined the objects positioning, scale, color, font format, and input restrictions. As the GUI was being developed it too went through a feedback process that was included in some of our weekly meetings. Through those many adjustments had to be made to the GUI based on Dr. Gerbi's feedback. This included moving the software to a tab based design so three additional windows had to be designed. These additional windows were not as complicated as the first but needed to keep the GUI's general design consistent. The review process of the news windows was interrupted by COVID-19, our team was no longer able to hold weekly meetings with Dr. Gerbi and contact with him was very difficult. So the new window designs were not able to get reviewed by Dr. Gerbi, so those window designs remain in their preliminary state.

3. The Bumps Along the Way

As with any great plan, Noone can ever fully foresee what the future holds. This definitely held true for this project, with unseed difficulties with SeidarT's legacy code and also a global pandemic. Even with the negative impact these complications had, there was a lot learned from the unique challenge. Adaptation in a changing environment and dealing with unforeseen technical issues were both things that our team experienced and learned from during this project.

Legacy Code

The backend code was written in Fortran 95 before we were introduced to the project. This was done with limited availability of developers with knowledge on newer coding practices and language equivalence. The client stated that the software was faster in the selected language over something such as python or C. This caused a lot of issues with the automatic testing and initialization of the project. If we were to continue work on the project, it would be of interest to convert this code to a modern language. This would assist in maintainability and development workflow.

The Elephant in the Room

It would be an understatement to say that the widespread influence of the COVID-19 pandemic has been universally experienced. Some have been more affected than others, but nobody is truly unaffected. Dr. Gerbi being the head of his department led to Team Penobscot suffering from a lack of client discussion at the final stages of development. Fortunately, the laying out of specifications and desired functionality meant we were fine to coast alone for the time being. We were all affected by quarantine in different ways. Lethargy affected some, panic affected others, and some completely shut down. People we expected to be pillars of stability broke down, and those we expected to break down showed they truly shined deep down. End of the day we were able to deliver a product during one of the most unprecedented times in history, and that stands for something.

4. Requirements Review

#	Name	Summary	Status in Software
1	Import Model Geometry	The User imports a model geometry (.png) through the GUI into the back end.	This requirement is handled by communicating with the back end through the use of console commands calling the original SeidarT.
2	Import Image	The user imports a png file which will be used to construct a prj file.	The front end completes this requirement by importing a png based on the name inputted by the user. At the current moment there is no file navigation.
3	Input Parameters	The user inputs into the front-end the parameters of the survey being requested.	Through Kivy, the front end asks the user for all possible parameters for all runtypes simultaneously.
4	Run Plot	A graphic is generated at the request of the User.	This requirement is handled by communicating with the back end through the use of console commands calling the original SeidarT.
5	Preview Plot Metadata	A plot has been generated and its information will be displayed for the user	This requirement is handled by communicating with the back end through the use of console commands calling the original SeidarT.
6	Generate a Comparison Plot	The user supplies two plot files that have been generated and differences between them.	This feature was pushed to a later release due to legacy code restrictions. It will be handled by the original SeidarT Team. It was not required for the basic functionality the client desired.
7	Generate a Graphic	A graphic is saved locally for later retrieval	This functionality was paramount to the basic operation of the software. It is presently working as desired by the client.

Appendix A – 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: 5/7/2020
Comments:

Signature: Adam Farrington

Team Member #2:
Name (Printed): Alexander Thacker Date: 5/7/2020
Comments:

Signature: Alexander Thacker

Team Member #3:
Name (Printed): Jens Hansen Date: 5/7/2020
Comments:

Signature: Jens Hansen

Team Member #4:
Name (Printed): Nathan Gazey Date: 5/7/2020
Comments:

Signature: Nathan Gazey

Appendix B - Team Contributions

- Adam - 25%
    - Introduction
    - Pair Programming
    - The Elephant in the Room
    - Appendices
    - Reviewed entire document
- Alex - 25%
    - Section Introductions
    - UI Design
    - Reviewed entire document
- Jens - 25%
    - Requirements Review
    - Reviewed entire document
- Nathan - 25%
    - Legacy code
    - Requirements review
    - Reviewed entire document
    - Document formatting

Appendix C - 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

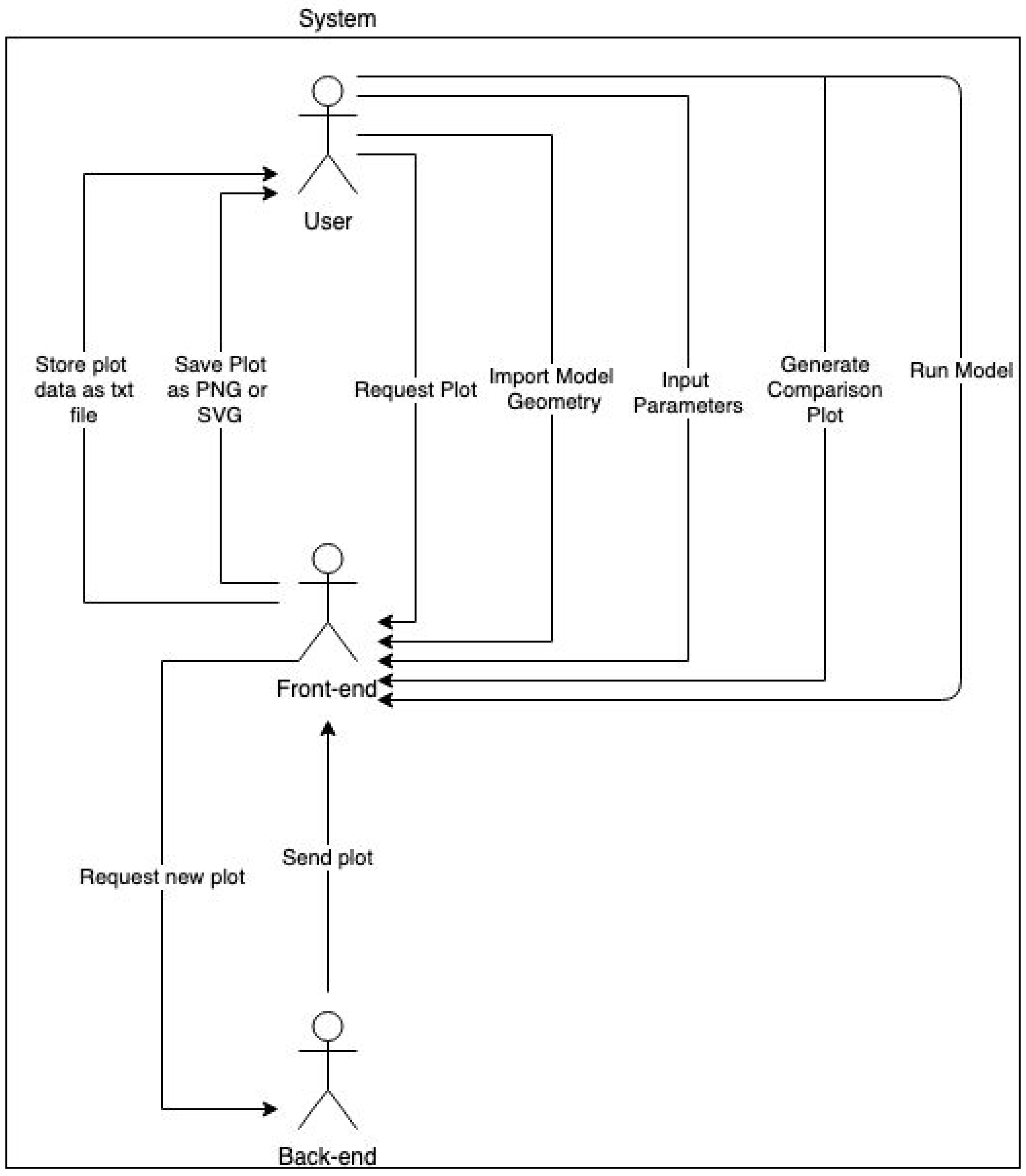
1. A slideshow describing the purpose of the backend software.
2. The software generates cross-platform executables out of python code.

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 D - System Design Document

1. Introduction

1.1 Purpose of This Document

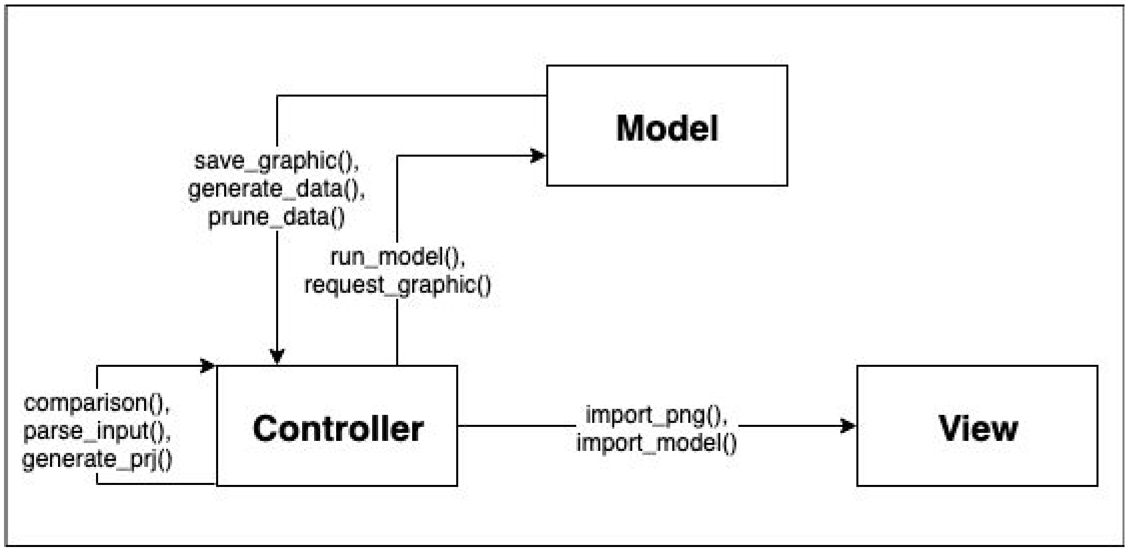
This is the formal agreement between Dr. Gerbi and Team Penobscot that describes in detail the design of the system being developed by Team Penobscot. This project is a user interface for a software system, SeidarT, currently accessed through command lines only. Included in this document are the proposed 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

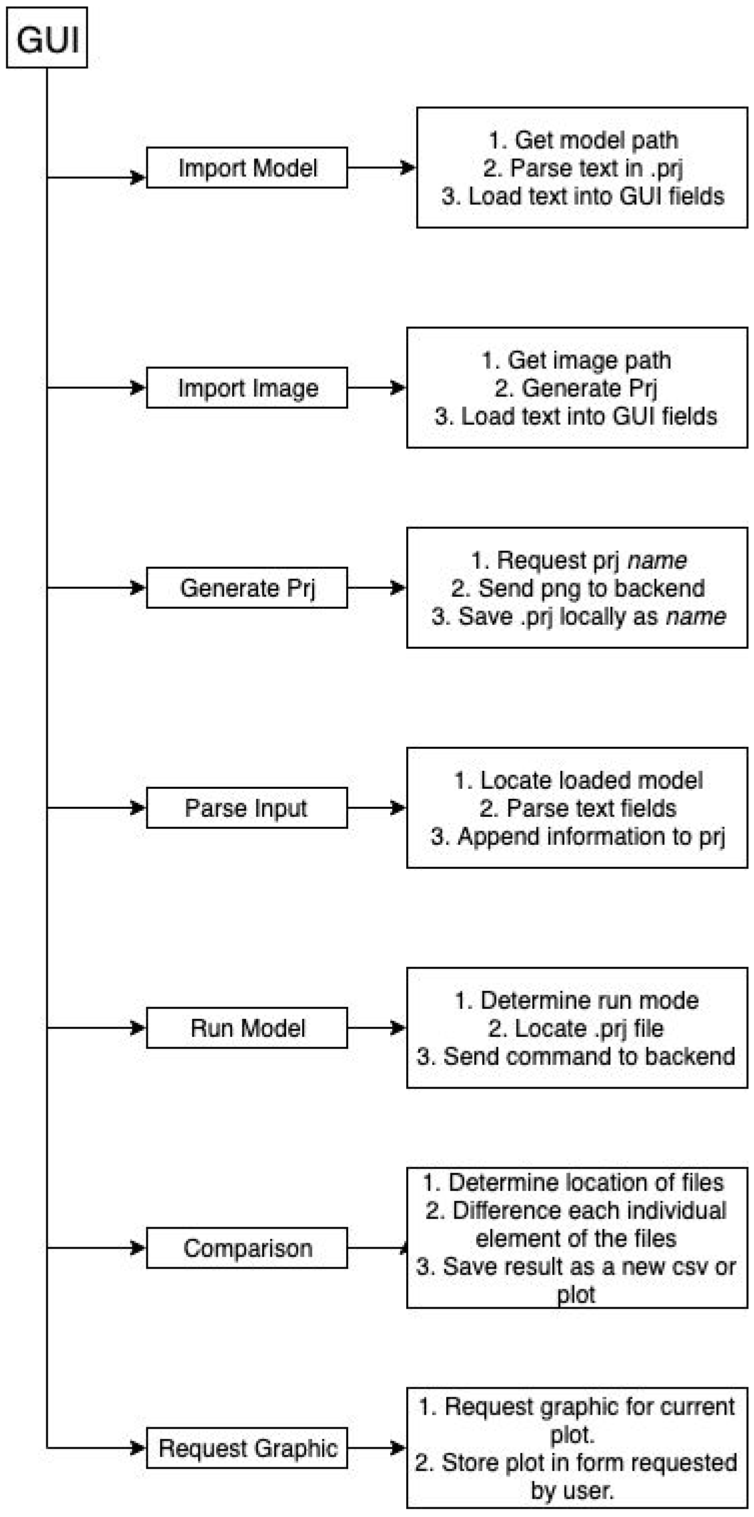
1. A slideshow describing the purpose of the backend software.
2. A GitHub Repository of the backend software.
3. Team Penobscot's own Software Requirements Specification.
4. Team Penobscot's own User Interface Design Document.

2. System Architecture

2.1 Architectural Design



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 D - User Interface Design Document

1. Introduction

1.1 Purpose of This Document

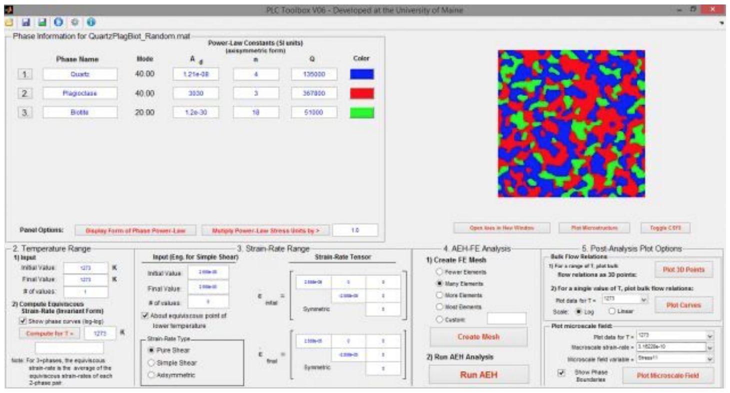
This is a formal agreement between Dr. Gerbi and Team Penobscot that describes the interface design of the system being developed by Team Penobscot. This project is a user interface for a software system, SeidarT, currently accessed through command lines only. This document will specify how a user will 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 C/I/T

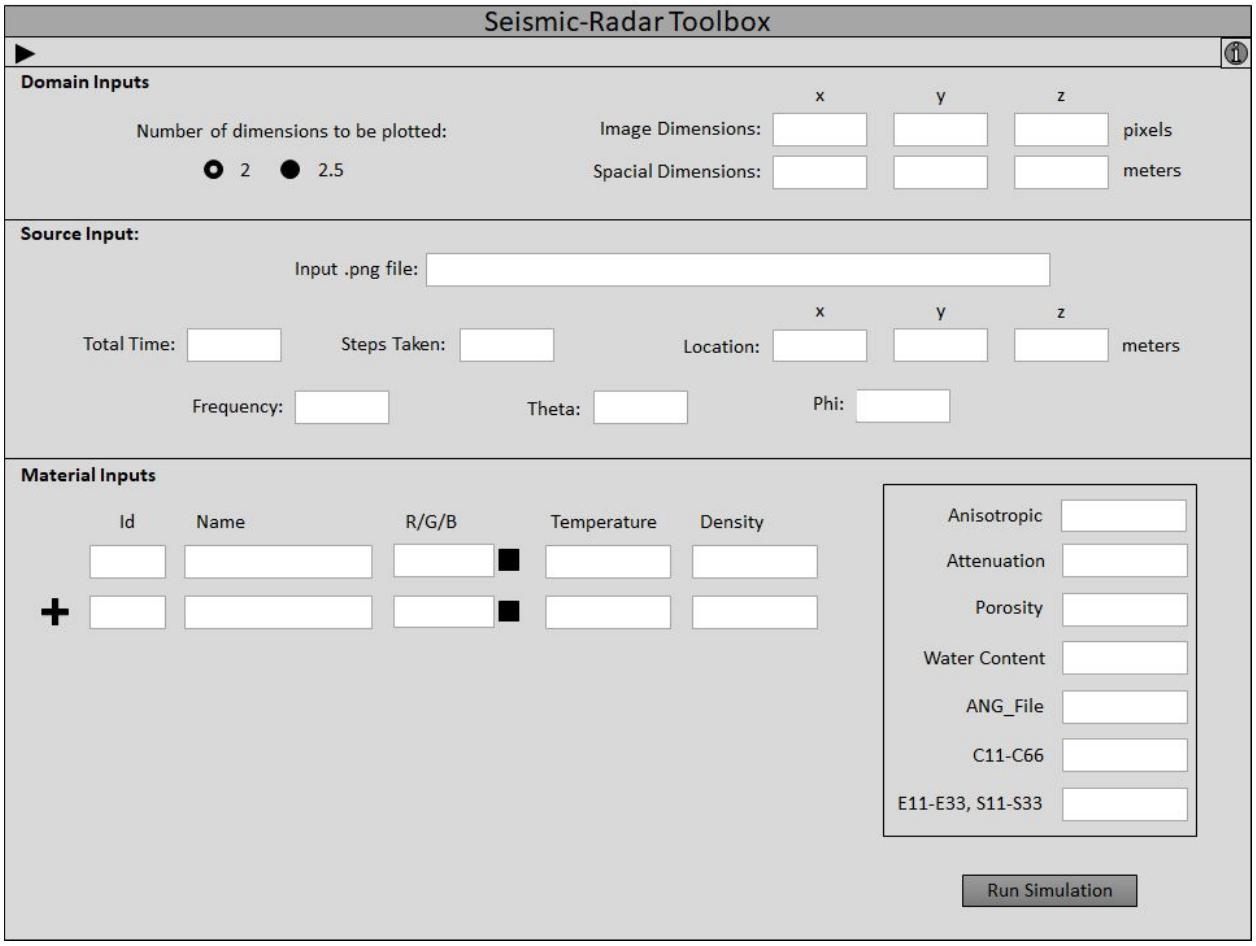
2. Temperature Range
1) Input
Initial Value: 1273 K
Final Value: 1273 K
# of values: 1
2) Compute Equiviscous Strain-Rate (Invariant Form)
✓ Show phase curves (log-log)
Compute for T = 1273 K
Note: For 3+phases, the equiviscous strain-rate is the average of the equiviscous strain-rates of each 2-phase 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
$\dot{\varepsilon}_{initial}$ = Symmetric
$\dot{\varepsilon}_{final}$ = Symmetric

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

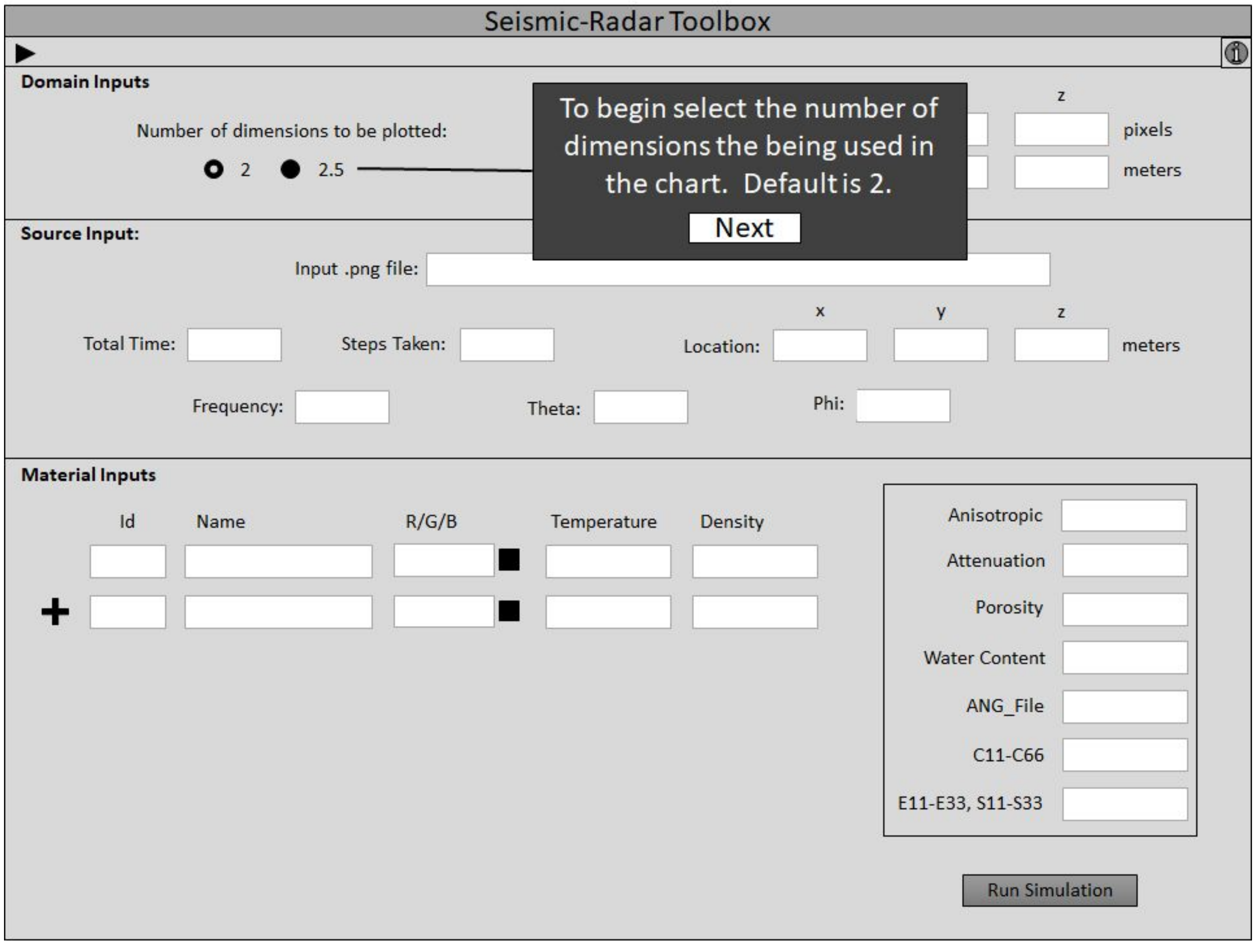
C11-C66

E11-E33, S11-S33

Run Simulation

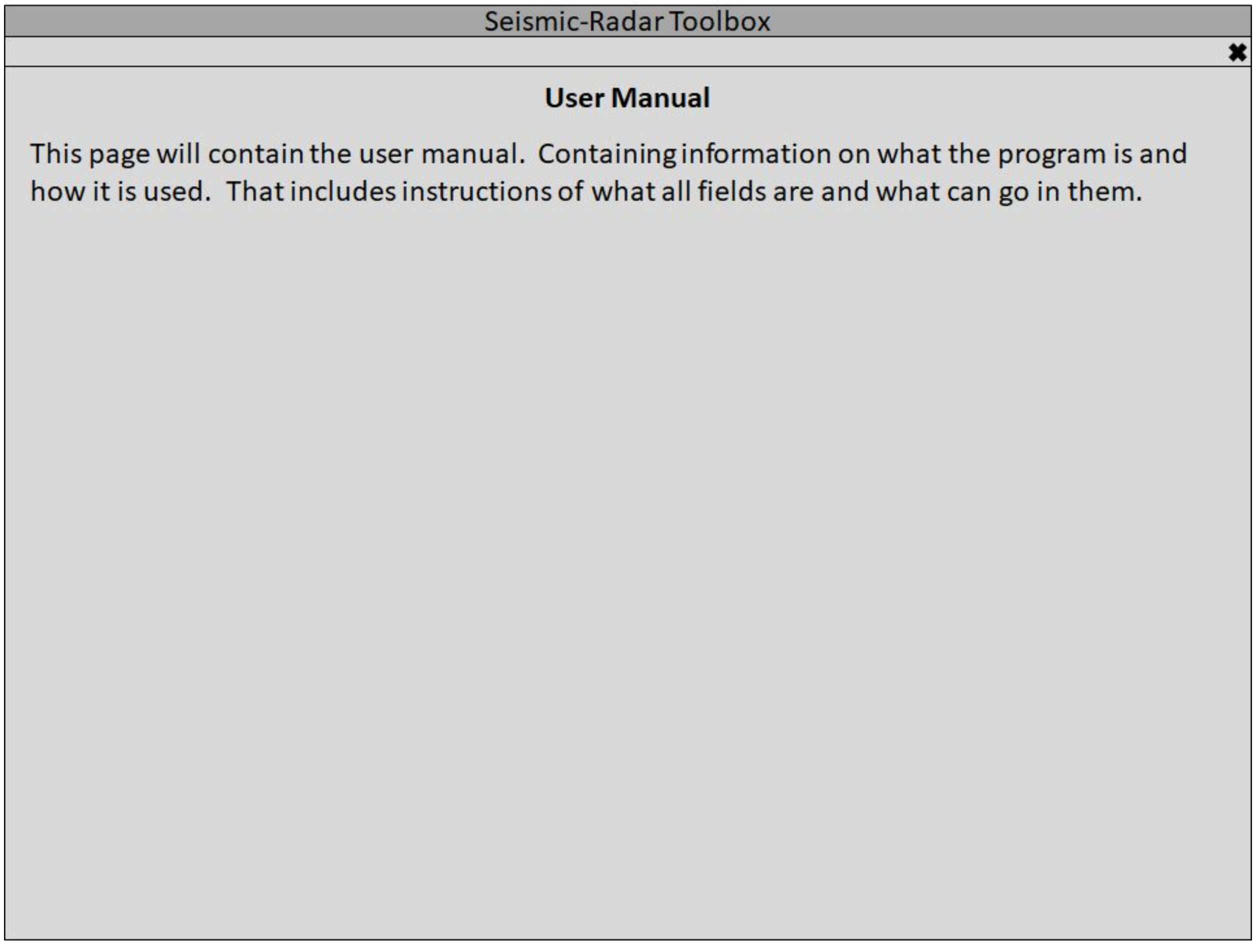
3. User Interface Walkthrough

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.

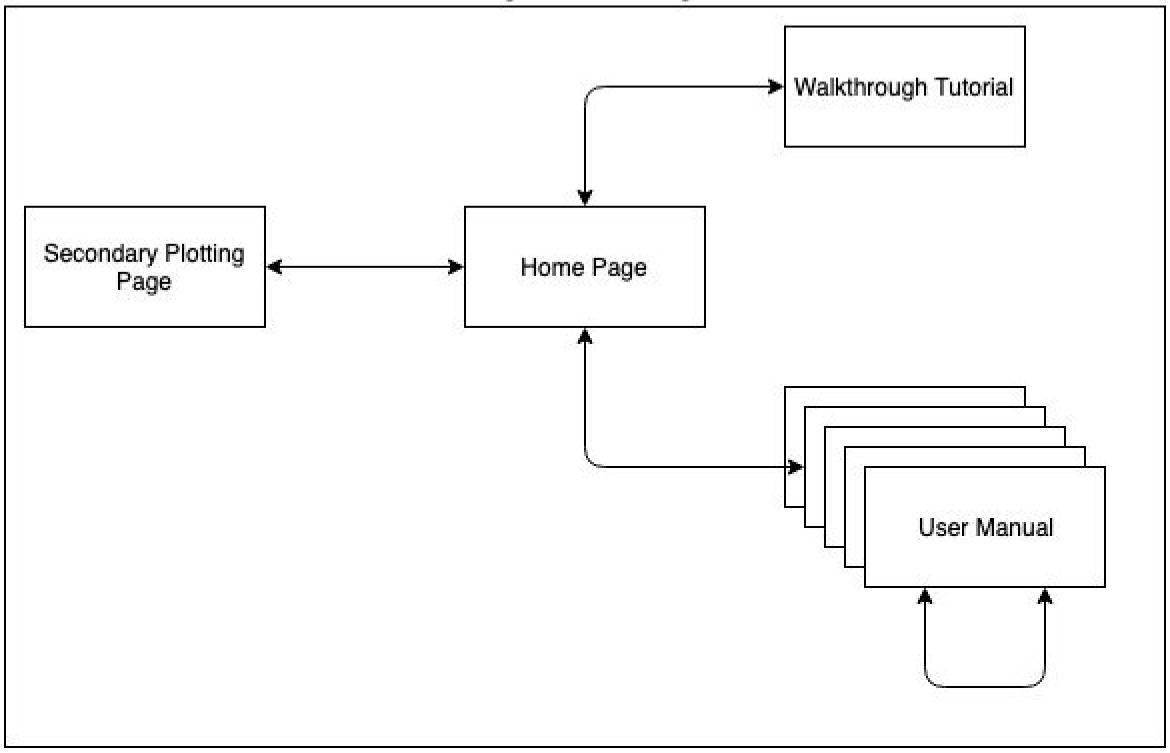


User Manual

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.

Appendix E - Code Inspection Review

1. Introduction

1.1 Purpose of This Document

This document covers the programming standards that are to be upheld by the members of team Penobscot throughout the development cycle of SeidarT. Team Penobscot, as part of this plan will conduct a number of code inspection during this cycle. During these code inspections our team will be inspected to ensure that these standards are upheld. This document also contains details of the team's first code inspection meeting. This will be used as a template to be used as a guide for future inspections.

1.2 References

1. A Github repository which contains all relevant code and documentation regarding this project.
2. Team Penobscot's SRS, found in the above Github repository.
3. Team Penobscot's SDD, found in the above Github repository.

1.3 Coding and Commenting Conventions

We are using the PEP-0008 standards that were set by Python in 2001. These standards are fully embraced except functional and variable names are to be camel case. This difference would change the standard typical "get_image()" to "getImage()". This change is made as the entire team agreed that they are used to this style and find it easier to read. This standard was chosen due to the ability for static analysis capabilities and industry-wide adoption.

1.4 Defect Checklist

Figure 1 - Defect Table by Type

Defect #	Module Location	Type of Defect
1	getImage()	Correctness

		User Friendliness
2	getImage()	Correctness
3	Kivy File	Coding Convention
4	Kivy File	Coding Convention
5	SeidarTGUI.py Imports	Coding Convention
6	SeidarTGUI.py imports	Coding Convention
7	SeidarTGUI.py configurations	Correctness
8	build()	Coding Convention
9	build()	Coding Convention
10	getImage()	Correctness
11	getImage()	User Friendliness
12	getImage()	Coding Convention
13	getImage()	Coding Convention
14	getImage()	Correctness
15	getImage()	Coding Convention
16	getImage()	Coding Convention
17	run()	User Friendliness
18	run()	Coding Convention
19	run()	Coding Convention
20	Throughout	Commenting Convention

2. Code Inspection Process

2.1 Description

Our team's first code inspection took place at Fogler Library on February 29th, 2020. The plan was to all meet to perform a walkthrough review session of everyone's contribution to the code. Unfortunately Nathan was not able to be there in person but he was able to participate remotely by using Discord. During the meeting Adam took the role of Recorder while Jens acted as the groups Moderator. We decided it would be best to go through our code chronologically, based on how the program would operate during normal operations. Due to how our group distributed work on the software we did not have any issues in terms of jumping back and forth between multiple authors during the inspection.

We started by going over the general appearance of the Graphical user Interface. Alex was in charge of everything having to do with the kv file, which managed all of the visual properties of displayed objects. There were not any major defects found in the kv file. The most notable issue was with the files coding/formatting convention. After that Jens and Adam took over as they walked through the main section of the program. They worked together on it through the extreme programming style of development. Similar to the previous section, most of the issues again were based on coding conventions. Other defects that were found were focused on code correctness and user friendliness. Nathan did not have code to present to the group since he was trying to set up Travis CI for continuous integration testing. That said, Nathan and the members who were not currently presenting their code were all contributed to the inspection. The code inspection was a success and helped our group identify many things that we needed to address in our software.

2.2 Impressions of the Process

The walkthrough process of reviewing the teams code worked well for our group. We were able to find 20 defects during our teams inspection review. We went through each person's code, taking the time to make sure that everyone understood what their code does. It was a great way to get everyone up-to-date with the current state of the entire program. A number of the defects that we discovered during the inspection process were pointed out by their author before others noticed them. This goes to show how important personal code reviews can be. The majority of the defects that were found during the code review were code convention violations. Our group never set up a standard coding convention before we started programming. These were small defects that didn’t impact the operation of the code, most were naming conventions. Nevertheless they still needed to be addressed. The group code inspection was very beneficial for our group and is something that we should do more often.

Of our modules listed in section 3, we would wager to guess that the most defect-free modules we have would be build(). While GUI building is imperative for the creation of our software, it is not an inherently debug filled process. The defects in this module are all about coding conventions and commenting conventions, no defects in that module point to correctness. Of our modules we would also wager that getImage() is our most defect-filled module. This module was the first built, and as such was built before most of our coding conventions were put into place. This in addition to the ever-difficult process of searching file locations and reading image files leads to it being the module most prone to error.

2.3 Inspection Meetings

As of the creation of this document our team has only held one full team inspection review. It was held on the 29th of February with all team members present. It started at 10am on a virtual call in the Folger Library. This was moderated by Jens Hansen with Adam Farrington as a scribe. We completed a module by module walkthrough with Alex and Nathan acting as the code reviewers.

3. Modules Inspected

3.1 `SeidarTGUI.py` Imports

SeidarTGUI.py is where a majority of our magic happens. It's where *getImage()*, *run()*, and *build()* live and as such uses the same imports across all three. While no functioning code exists outside another module, as all three modules work under this wrapper it is important to be reviewed as well for enforcing conventions.

3.2 `getImage()`

getImage() is the module that handles the user inputting the name of an image file and subsequent importing of that file into the GUI. This module both imports the image and creates an empty PRJ based on the image chosen. The module also reads a pre-existing PRJ file, and imports into the GUI all information found there that was inputted on previous runs. *getImage()* satisfies several functions listed out in the SDD. As it creates a blank PRJ, it works alongside *run()* to satisfy Generate PRJ, and handles Import Model and Import Image in completeness.

3.3 `run()`

run() is the module that handles the reading of user inputs into a PRJ and the running of computation models SeiDartT is known for. Like *getImage()*, *run()* is a workhorse module that handles a majority of the SDD's functions. It works in tandem with *getImage()* to create completed PRJ files, satisfying Generate PRJ. As it interacts with the Kivy File it also handles Parse Input, Run Model in completeness.

3.4 `build()`

build() works primarily alongside the *Kivy File* to create the GUI that the user interacts with. The *Kivy File* handles some aspects of input parsing, but *build()* alone doesn't engage in any functions listed in the SDD. Build simply imports the pieces build in the *Kivy File* and gives them names that other functions can find.

3.5 Kivy File

The Kivy File is another half of the GUI alongside *build()*. Kivy allows parts of a GUI such as a text input or a button to have labels that can be grabbed later via Python code. As such, it plays a passive yet still integral part in the Parse Input function specified in the SDD.

4. Defects

Figure 2 - In Depth Table of Defects by Module

Defect #	Module Location	Type of Defect	Description of Defect
1	getImage()	Correctness User Friendliness	If a file name not found is given by the user, the software crashes.
2	getImage()	Correctness	The first material input is inaccessible due to a layout issue.
3	Kivy File	Coding Convention	'Radar' and 'Seismic' are less descriptive than our conventions demand.
4	Kivy File	Coding Convention	Our convention is to put objects/modules in order or relevance. This convention is not followed in the Kivy file.
5	SeidarTGUI.py Imports	Coding Convention	Imports should be done as an on-need basis, where we have imported entire libraries.
6	SeidarTGUI.py imports	Coding Convention	Kivy imports are explicit and lead to legacy imports as modules change.

7	SeidarTGUI.py configurations	Correctness	Minimum height and width are not being enforced.
8	build()	Coding Convention	Panels are named panel1-4, not descriptive of their purpose.
9	build()	Coding Convention	Input forms are not unified in naming style. “Seismic_stuff” for instance.
10	getImage()	Correctness	The image imported before is not erased, leading to image overlap.
11	getImage()	User Friendliness	Image scale is not controlled at the moment, leading to a massive image potentially overlapping everything.
12	getImage()	Coding Convention	System calls in coding are correct, but not recommended by our convention.
13	getImage()	Coding Convention	colorsTuple may not need the float type casting it is given.
14	getImage()	Correctness	“The spacing on this is too large, but decreasing it causes some rows to disappear.”
15	getImage()	Coding Convention	Logical misordering of getting materials, making the materialBox, and scraping colors.
16	getImage()	Coding Convention	Several variables (‘content’, etc) are named in nondescript ways.
17	run()	User Friendliness	Before running, we re-build the prj to protect the user from mistakes such as editing the prj file. However this is computationally heavy. A decision needs to be made regarding if we want to protect the user from a niche example or save the time of re-building the prj file.
18	run()	Coding Convention	Variables are not named descriptively.
19	run()	Coding Convention	Duplicate code potential in writing to file.

20	Throughout	Commenting Convention	Comments are insufficient throughout several modules. Comments primarily separate by group, but these group comments require more description.

Appendix F - Administrator Manual

1. Introduction

1.1 Purpose of This Document

This document covers the relevant information for an administrator looking to use SeiDarT in a professional or educational setting. This document goes over the uses of SeiDarT both professional and education, and shows how to begin working with the software. This document also covers the installation process for SeiDarT and its dependencies, directing the user to where each dependency can be found. Lastly this document helps guide an administrator in managing day-to-day operation of the software, including user support and dealing with limitations in the software.

1.2 References

1. Team Penobscot's own Software Requirements Specification.
2. Team Penobscot's own System Design Document.

2. System Overview

2.1 Background

The SeiDart GUI application is two-fold. The GUI and CLI versions require two separate install processes. The main graphical version requires no additional work on a routine basis to maintain. This version will only require changes after completion of new features to the CLI version of the application. For additional information regarding this process, please refer to section 3.3 of this document.

2.2 Hardware and Software Requirements

Use of SeiDart can be done via the enclosed executable, titled "GUI", or via the python script, "GUI.py". Each of these methods requires the installation of:

- Mac:
  - Homebrew
  - Xcode (Installed via Homebrew)
  - GCC (Installed via Homebrew)
  - Ghostscript (Found at https://www.ghostscript.com/download.html)
  - ImageMagick (Found at https://imagemagick.org/index.php)

- Windows:
  - Ghostscript (Found at https://www.ghostscript.com/download.html)
  - ImageMagick (Found at https://imagemagick.org/index.php)
  - GCC (Found at https://sourceware.org/cygwin/)
  - Fortran95 (Found at Here)
- Linux:
  - Ghostscript (Found at https://www.ghostscript.com/download.html)
  - ImageMagick (Found at https://imagemagick.org/index.php)
  - Fortran95 (http://www.g95.org/)

Further, if you choose to use the python script rather than the executable, you will also be required to install Python3 (Version 3.7 or greater). Once installed, the following libraries must be installed:

- Numpy
- Scipy
- Matplotlib
- Glob3
- Mplstereonet

Installation of the above packages is most easily accommodated using pip.

3. Administrative Procedures

3.1 Installation

The program will be provided as a portable executable for each major desktop operating system. The Microsoft Windows, Apple OSx, and Linux versions will be incompatible with other systems and should be used only on the respective system.

3.2 Routine Tasks

Our software is a GUI for the SeiDart software. Since the GUI is a standalone product no routine administrative tasks are required. If any maintenance or change needs to be made to the SeiDart GUI they will be made through a software update.

3.3 Periodic Administration

The SeiDart GUI will require minimal administration and maintenance. As the backend software is updated the GUI will have to be changed to accommodate any backend changes. This should only occur when any IO data is changed or when the application programming interface (API) is altered.

3.4 User Support

There will be three sources for user support for SeiDart . There will be a help page built into the software which will include basic instructions for operating the software. In addition, a complete User Manual that will include more detailed instructions on how to operate the software will be provided by the Team Penobscot. If neither of those supply the needed information for the user they will be instructed to contact the software admin, which at the time of this documents creation would be Dr. Gerbi. He can be reached at earthclimate@maine.edu.

4. Troubleshooting

4.1 Dealing with Error Messages and Failures

There are no error messages programmed into the SeiDart software. If an error message is encountered, attempt to restart the SeiDart software. If the error recurs then note how the process leading up to the error and report it as a bug to Dr. Gerbi.

4.2 Known Bugs and Limitations

Caution is needed when choosing images to process through the software. Specifically, images using anti-aliasing as a way to smooth the edges on images cause a very strange effect in SeiDarT. Anti-Aliasing is a method to smooth the edges on graphics by averaging the colors between edges, putting for example a grey gradient along the border of a black line and a white background. This causes a drastic increase in colors detected, as each part of the gradient is seen as a unique RGB combination when creating the prj file used by the GUI.

Appendix G - User Guide

1. Introduction

1.1 Purpose of This Document

This document explains in detail the software created in the agreement between Team Penobscot and Dr. Christopher Gerbi. That agreement resulted in the creation of a graphical user interface for the software system SeiDarT. SeiDarT is a learning tool for both professors and students that simulates seismic and radar propagation. The software has been augmented by Team Penobscot with the goal of making the software more accessible to its desired users.

1.2 Intended Readership

This document is intended for reading by end users of SeiDarT, whether they are students using this themselves or administrators teaching students how to use the software second-hand. In both cases, the only assumed experience of the end user is a rudimentary understanding of how seismic and radar waves propagate through surfaces. At the moment SeiDarT has English as its only language option.

1.3 Applicability Statement

This document refers to the software delivered to Dr. Gerbi as of the end of the school semester of Spring 2020. This is considered Version 1.0 for the purpose of the rest of this document.

1.4 Purpose

The purpose of this document is to enumerate the steps to run simulations of seismic and radar activity. To do this, this document walks the user through the inputting of images and information into the software's user interface.

1.5 How to Use this Document

This document begins with an overview of the system as a whole. The Overview serves to give the user a grasp of the functionality of the system. From there this document goes into more in-depth instructions about the multiple functionalities of the system separately. The goal of this layout is that users looking for a simple summary can find what they need in the Overview, and the users needing to learn a specific functionality can find it in Instructions.

1.6 Related Documents

Document Number	***Description***
#1	*Team Penobscot's Github Repository for SieDarT*
#2	*A Slideshow Demonstrating SeiDarT*

1.7 Problem Reporting Instructions

At the time of the writing of this document, the administration and maintenance of SeiDarT is being performed by Dr. Gerbi. If one wishes to report an issue in the software, Dr. Gerbi can be contacted regarding issues and help with SeiDarT at earthclimate@maine.edu.

2. Overview

The SeiDarT GUI is based around a linear workflow to create simulations using data provided by the user. The user is assumed to understand how to use a file browser and where to properly store a resultant image.

SeiDarT uses user provided parameters to output a simulation of a seismic or radar event on a given cross section of material. This idea can be explored using the CLI or the GUI, however the program assumes the user has full knowledge of how each parameter effects the end result.

3. Instructions

This section will detail usage of the SeiDarT GUI. As such, it is highly recommended that users first familiarize themselves with the primary functionalities of SeiDarT. Additional resources for this application can be found at GitHub and slideshow.

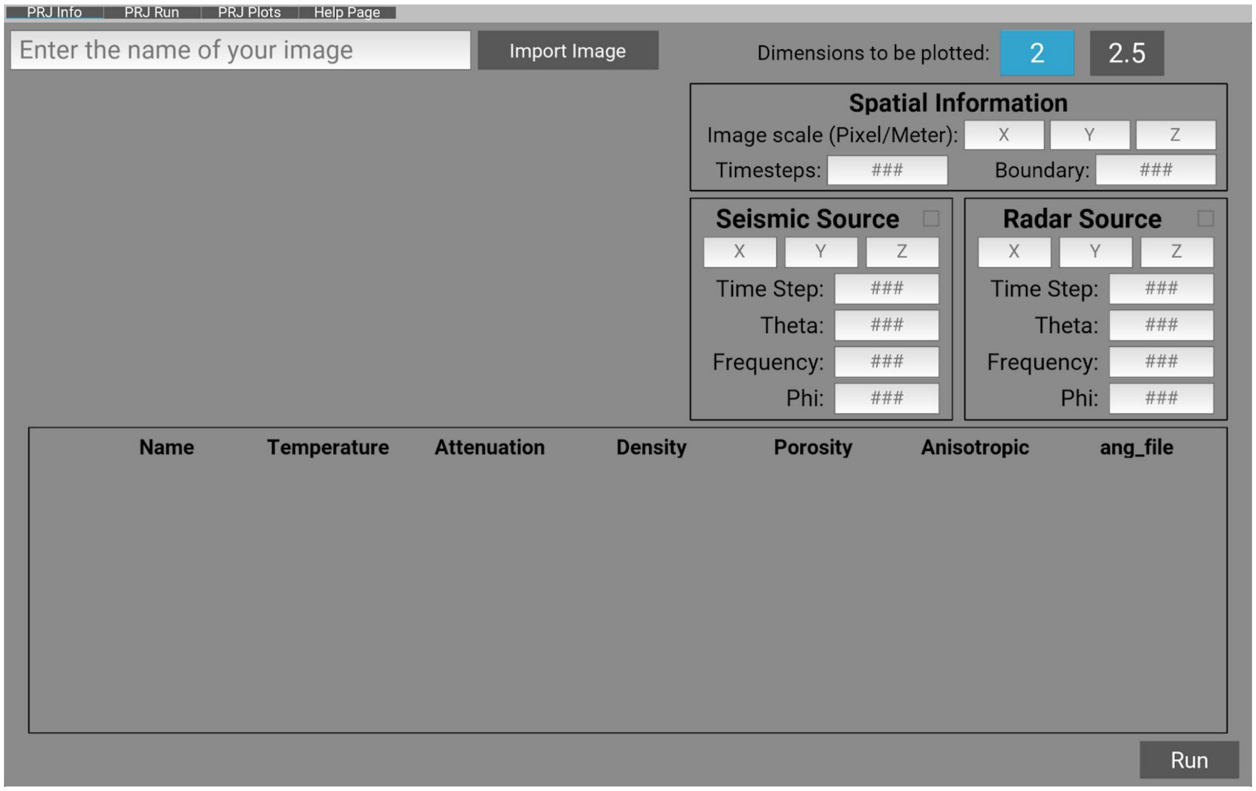
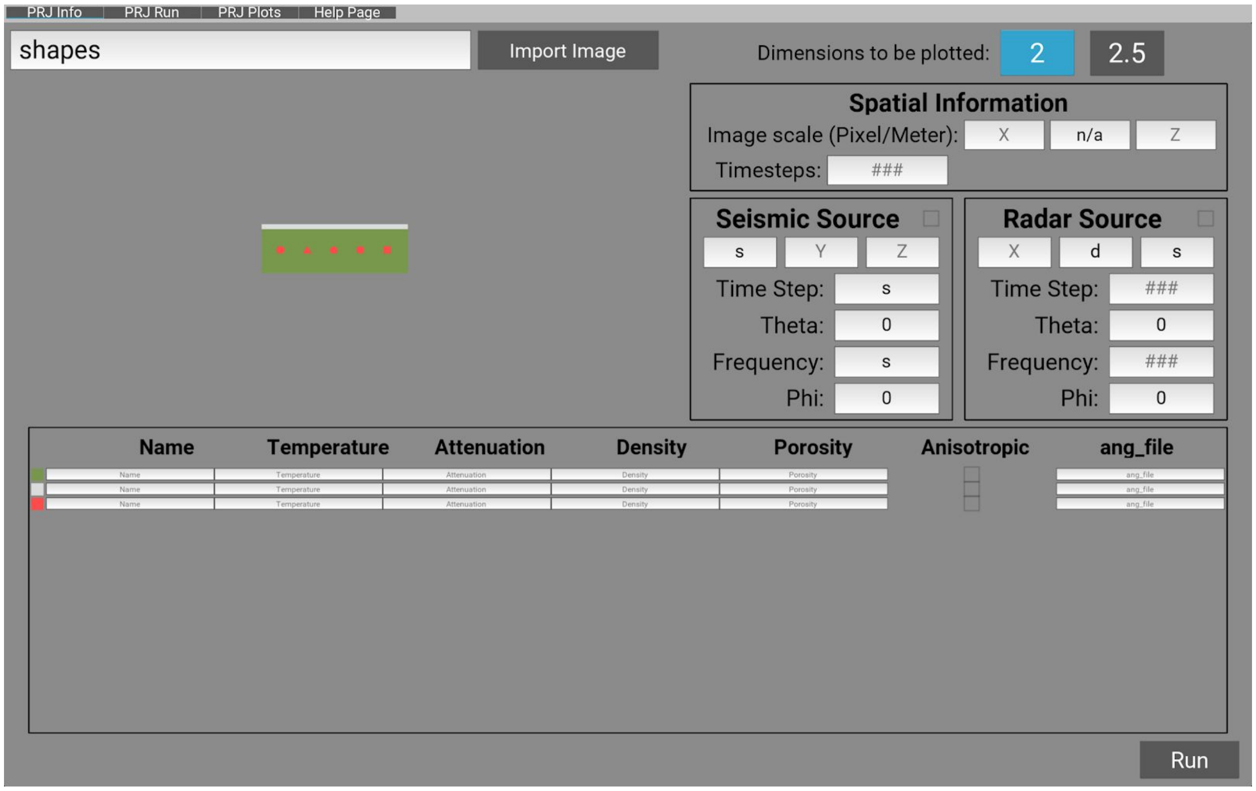


Figure 1

Once opened, the application will appear as in *figure 1* above. As seen at the top of the image, there are four tabs. The first three, labeled *PRJ Info*, *Multishot Run*, and *Plots*, contain the primary functionality of this application. The general workflow involves entering the requisite information in each of these three tabs, in sequence. As such, this guide will begin with a description of the *PRJ Info* tab, followed by the *PRJ Run* and *PRJ Plots* tabs, in that order. Prior to this though, the user must first take the .png image that they wish to process and place it within the GUI folder.

PRJ Info

The first step in the *PRJ Info* tab is importing the image. In the top left hand corner there is a text box. In this box type the name of the image, **excluding** the file extension. Below is an example where the image "*shapes.png*" is being processed.



shapes

Dimensions to be plotted: [✓ 2] 2.5

Spatial Information

Image scale (Pixel/Meter): X n/a Z

Timesteps: ###

Seismic Source			
s	Y	Z	
Time Step:	s		
Theta:	0		
Frequency:	s		
Phi:	0		

Radar Source			
X	d	s	
Time Step:	###		
Theta:	0		
Frequency:	###		
Phi:	0		

	Name	Temperature	Attenuation	Density	Porosity	Anisotropic	ang_file
🟩	Name	Temperature	Attenuation	Density	Porosity	☐	ang_file
⬜	Name	Temperature	Attenuation	Density	Porosity	☐	ang_file
🟥	Name	Temperature	Attenuation	Density	Porosity	☐	ang_file

Figure 2

Once the image has been entered, the user may choose which of the remaining fields to fulfill first. We will begin in the large box near the bottom. Once your image has been input, a series of input rows will appear. Note on the left hand side there are small rectangles which are colored similarly to the image input. Each of the rows corresponds to the requisite information about the material represented by that color of pixel. For instance, we might begin by naming the gray pixels "*air*" by typing "*air*" into the "*name*" box in the second row. After the name, temperature, attenuation, density, porosity, and the anisotropic nature of the materials have been entered, this section is complete. Next, we move to the dimensionality in the upper right hand corner. Users may elect for a 2 or a 2.5 dimensional plot by selecting which they desire. In the example image above, we have selected 2 dimensions. Moving downward now, we have three boxes to denote the scale of the image. If you have selected a 2-dimensional plot, the *y* box may be ignored. Each of the *x, y* and *z* boxes correspond to how many meters there are per pixel in that direction. Following this, the *timesteps* box denotes the number of steps the backend will compute before saving the contents. Finally, there are two similar boxes beneath the *spatial information* box. The left box contains information for a seismic simulation, while the right encapsulates a radar simulation. Users are to fill out the information for *x, y, z, time step, theta, frequency,* and *phi* per the SeiDarT requirements. Finally, there are checkboxes located to the right of "*Seismic Source*" and *"Radar Source"*. By checking either, you choose to run the simulation in that manner.

Users may select neither, radar, seismic, or both for running. Once all information has been entered, users should click the button titled "*Run"* in the bottom left to generate a .prj file.

PRJ Run

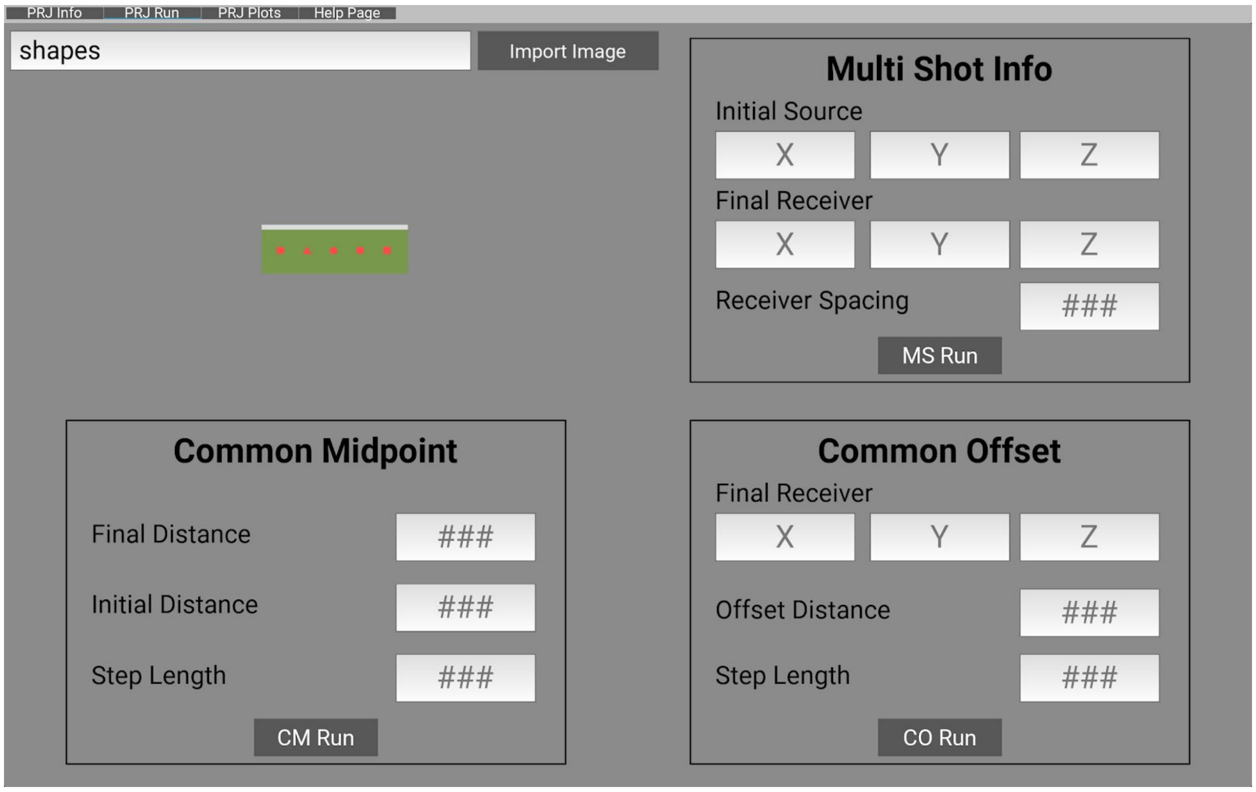


Figure 3

The first aspect of note in the second tab is the upper left hand quadrant. This is identical to that of the first tab. This should *not* be changed, unless a user wishes to run a .prj different from the one that was just created. The text that appears in the text box in the top left is assumed to be the name of the *.prj* file being used. As such, changing this information can negatively impact your experience. With that in mind, there are three other quadrants, each of which denote a different method of running the *.prj*, as described in the SeiDarT documentation. Merely fill out the required fields and click the corresponding run button.

PRJ Plots

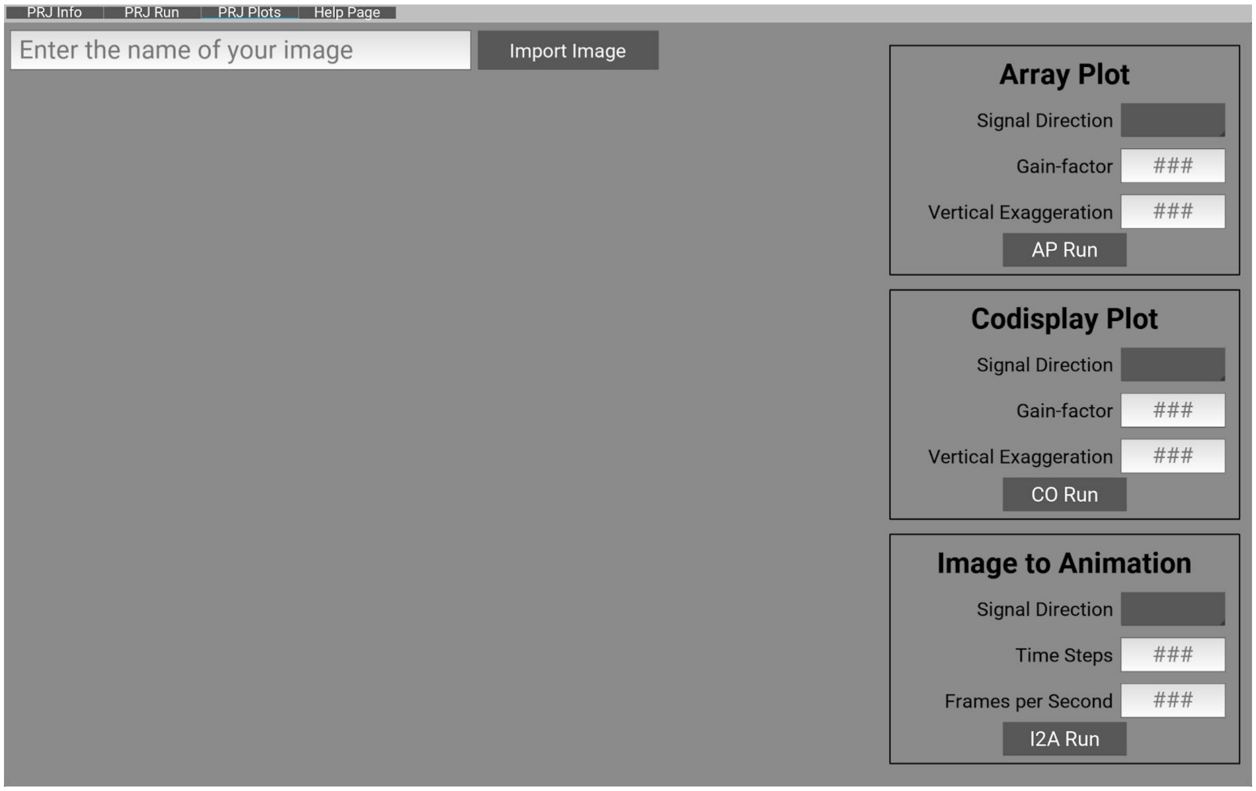


Figure 4

Finally, we have the *Plots* tab. This tab is meant to create and display the plots desired by the user. As noted in the above figure, the image region is again displayed. As above, this is not to be modified at this stage in the process. In the top right and bottom left quadrants are the *Array Plot* and *Codisplay Plot* regions. Using these, a user may generate a relevant plot (graphic) described by the run method chosen in the previous tab. Finally, a user might generate an animation using the region in the bottom right.

4. Error Messages and Recovery Procedures

The SeiDarT GUI is designed with the expectation that the user has the basic understanding of the backend SeiDarT software. The SeiDarT software currently does not give the user any error messages when problems occur. Below is a list of possible problems a user may experience while using the SeiDarT Software.

- Software goes unresponsive:
  - If the SeiDarT GUI goes unresponsive then force close the application and restart a new session. Some data from the previous session may be lost if

    it had been entered before any operation (build, run, or plot) was completed. Note that some of the SeiDarT operations take time so some calculations may take a few minutes to complete depending on the scale of the entered data.

- Software GUI has a visual glitch:
    - If the SeiDarT GUI has a visual glitch that prevents use of any feature try to close and restart the software.
- Software crash:
    - If the SeiDarT GUI crashes unexpectedly simply restart the software. Some data from the previous session may be lost if it had been entered before any operation (build, run, or plot) was completed.
- Data related problems:
    - The SeiDarT GUI has basic input validation in the form of input restrictions. The software will not warn the user if any data fields are missing or in the wrong format. Please refer to the SeiDarT resources that can be found at the items listed in section 1.6.

If none of the above helps to fix the experienced issue and the issue is preventing proper use of the software please report the issue to earthclimate@maine.edu