

# Threat Modeling Tool

Design Document of Threat Modeling Tool  
ECE48700 Fall 2022

Authors:

Aayush Mailarpwar  
Cooper Zuranski  
Kristopher Bright  
Joe Swiney

Sponsor:

Rolls Royce  
Luke Thomas  
[n.luke.thomas@rolls-royce.com](mailto:n.luke.thomas@rolls-royce.com)

Submitted on November 18, 2022

## Table of Contents

List of Tables and Figures .....	3
Background .....	4
Motivation .....	5
System/Subsystem Block Diagrams .....	6
Requirements .....	11
Technical Requirements .....	11
Standards.....	12
Safety Requirements .....	12
Other requirements .....	12
Constraints.....	13
Literature Search .....	14
Design Options .....	15
Organized Design Process .....	15
Alternate Approaches .....	17
Decision Making .....	18
Design Challenges .....	20
Risk Management .....	21
Design Details .....	22
Development Design Approach .....	22
GUI-Centered Design .....	23
Testing .....	34
ABET Outcomes .....	35
Appendices .....	36
List of Abbreviations .....	36
Bill of Materials .....	36
Lessons Learned .....	37
References .....	38
Verification Cross Reference Matrix .....	38
Project Plan .....	39

## List of Figures

Candidate User Interface .....	6
Process for changing a model component .....	6
Process for adding a new threat .....	7
Nominal sequence of user interaction .....	8
Nominal use cases .....	9
Block diagram of threat modeling program architecture .....	10
Screen 1, Initial .....	<b>23</b>
Screen 2, New Project .....	<b>24</b>
Screen 3, Model System .....	<b>25</b>
Screen 3, Model System Confirmation .....	<b>26</b>
Screen 4, Trust Boundaries .....	<b>27</b>
Screen 5, Threat Identification .....	<b>28</b>
Screen 6, Threat Discussion and Completion, Tool Selection .....	<b>29</b>
Screen 6, Threat Discussion and Completions .....	<b>30</b>
Screen 7, Threat Mitigation .....	<b>31</b>
Screen 8, Process Validation (with Example Use) .....	<b>32</b>
Screen 9, Export Results .....	<b>33</b>
Gantt chart of tentative project schedule .....	<b>39</b>

## List of Tables

Pro and Cons: Frontend Selection .....	<b>18</b>
Pros and Cons: Backend Selection .....	<b>18</b>
Pros and Cons: Integrated Development Environment Selection .....	<b>19</b>
Pros and Cons: Backend Testing Suite Selection .....	<b>34</b>
Pros and Cons: Frontend Testing Suite Selection .....	<b>34</b>
List of Abbreviations .....	<b>36</b>
Verification Cross Reference Matrix .....	<b>38</b>

## BACKGROUND

In relatively recent history, everyday devices have become more interconnected. Mobile phones were a large step, and with the shifting of communication methods, many more threats were introduced into everyday life. Threats have continued to evolve as technological capabilities and demands evolved. We are now at the point that we have watches with ECGs built into them, home assistants with microphones constantly listening, and cars that are told how to drive better with new software updates. With these innovations, new threats are a byproduct.

Threat modeling is the process of identifying and planning mitigation options for the security vulnerabilities of these and other systems. Though it sounds like it is comprised of only two steps, it is a circular process that doesn't stop until the product is decommissioned. As an application or system continues through the development process, new threats may emerge, and the mitigation techniques planned may no longer satisfy the need. Though we think of threats in the context of software, the applications are much broader. Threat modeling can be used for security systems, aircrafts, automobiles, embedded controls, phones, cameras, wireless equipment, and even things as seemingly simple as physical rooms. Though most of these things have software involved, that is not the point. A weak hinge is a security vulnerability to a room.

Many threat modeling programs are currently available, from open-source options like Cairis and OWASP Threat Dragon to expensive, licensed programs like SecuriCAD by Foreseeti (Mohanakrishnan). They allow users to create models of their system and evaluate the threats they identify. Many of the programs can aid in identification and mitigation by using sources such as MITRE's CAPEC and evaluate compliance based on rules set by policies such as PCI.

These tools model not only the threat but the context in which it resides. Parts of a system are not isolated to their individual components. Different parts of a system interact with each other, and as the system development evolves, these interactions become more complicated and more intertwined. Threats that were not considered during the initial development of one part of system must be re-evaluated and identified as that component's interaction count increases. Hence, the cyclical process in which new threats are identified and mitigated.

The most important detail about a threat model is that it is intended to be a living document. It should be structured in such a way that it can be edited and expanded upon. This is especially true when it comes to details. At first, the document will be very abstract, but it will become increasingly concrete throughout the development process.

## MOTIVATION

In many cases, security has been an afterthought, introduced far too late in the product production cycle. In most organizations, there isn't a security specialist on a design team, whether it be hardware or software, and most systems are a combination of both. Even if it is built in at the component level, security implications arising from the interactions between different components aren't always considered until it is too hard and expensive to make the necessary modifications. According to the International Council on Systems Engineering, the vast majority of cyber tools and techniques are applied very late on in the design process. This is a major roadblock as by this point, as many things may be unable to be changed, and if so, are incredibly expensive to do in the late stages. Part of the reason this occurs is because security experts are few and far between. Education on this topic was not even a standalone topic until recent years. Having to involve what few "experts" exist at a given company all the way from the design stages may not be possible, let alone cost-effective.

Herein lies the motivation for our threat modeling tool: a way to not only share a collective knowledge about systems security, but actively teach users about how to think about, identify, and consider mitigation techniques for the identified security risks. They almost certainly will not know the best solution to all these problems, but it sets a precedent for prioritizing and talking about security in every step of the design process. This leads to productive conversations with people who are experts, leading to more secure solutions without full involvement.

The programs in the Background section have a high barrier to entry when it comes to the knowledge required to use them. We seek to create a program to do threat modeling that lowers the barrier to entry from expert to the level to that of an average product engineer. This program will not help them identify all the vulnerabilities in their system, but it will help them learn to think about, recognize, and address a large subset of them. It will also allow users from different parts of the product development team to add their components to the model so they can be collectively evaluated and their threats mitigated based on new features and interactions.

The application doesn't serve to evaluate numerical probabilities or provide direct analysis, in the same way that UML diagrams are not how you actually create software. It merely represents the concepts as a discussion point. However, these analogies are separate in the fact that threat modeling needs to be visible and comprehensible to more than just those who work on security. The goal from here is to create awareness about the possible threats. Then, these can be modified as time progresses, allowing for a dynamic look at security. Assumptions will have the opportunity to become validated into actual threats, or simply thrown out. These models should help the user understand where vulnerabilities are, which are most relevant, how to prioritize solving them, and what constitutes "enough."

## SYSTEM/ SUBSYSTEM BLOCK DIAGRAMS

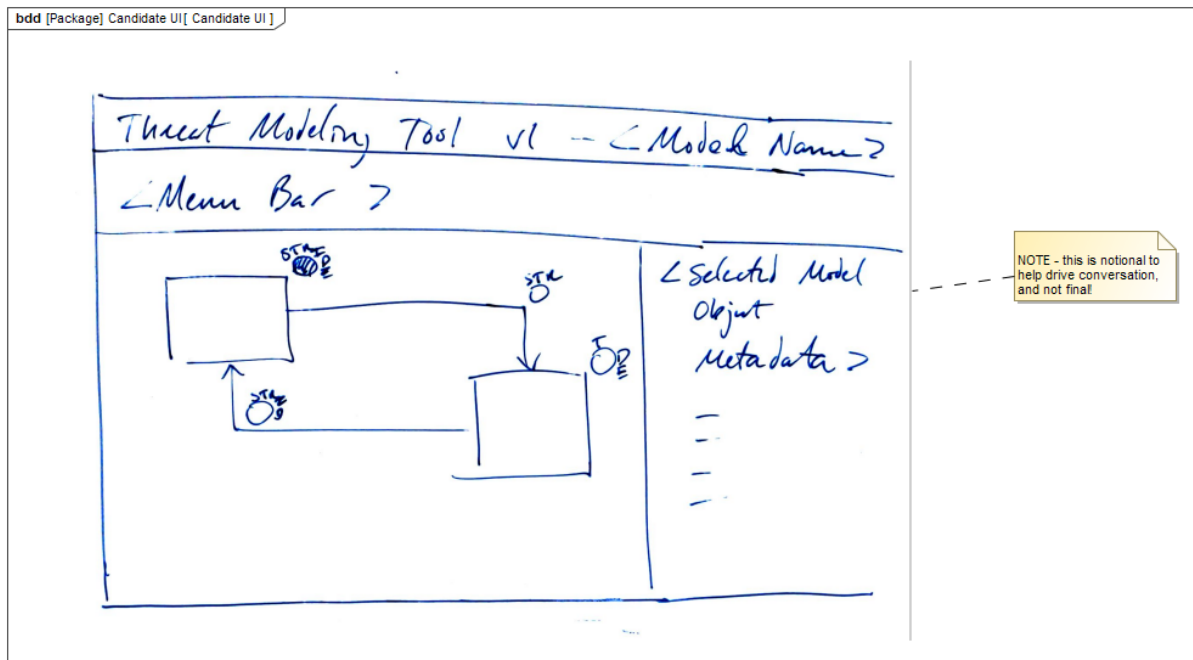


Figure 1: Candidate User Interface

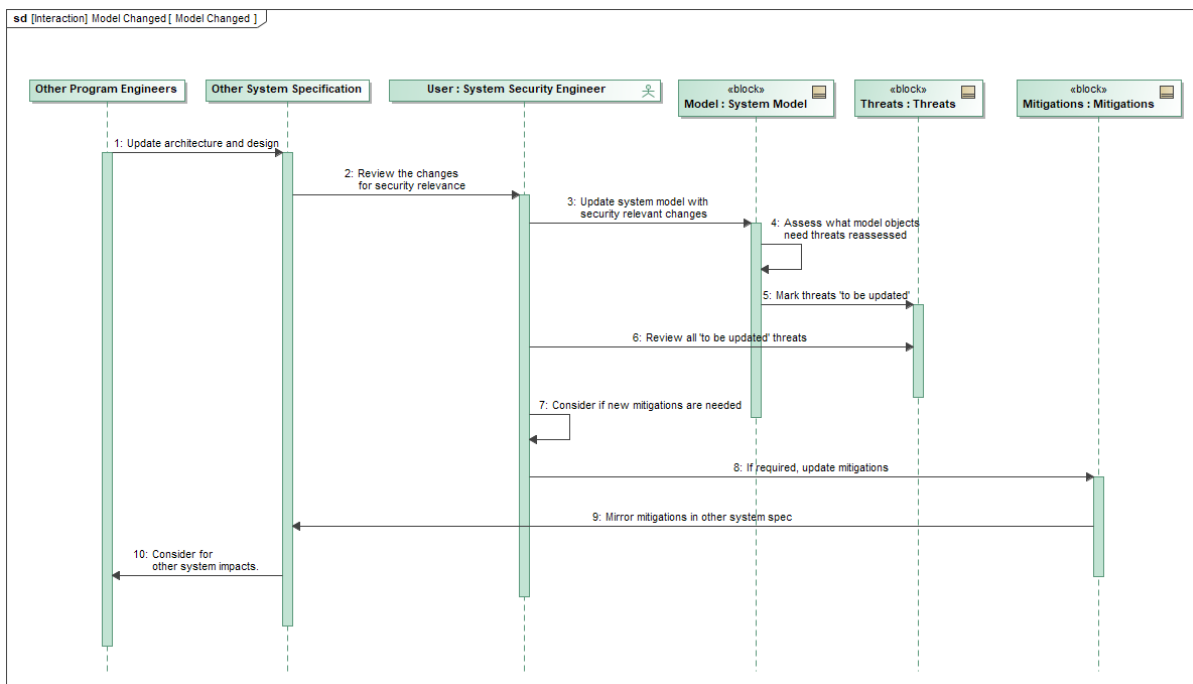


Figure 2: Process for changing a model component

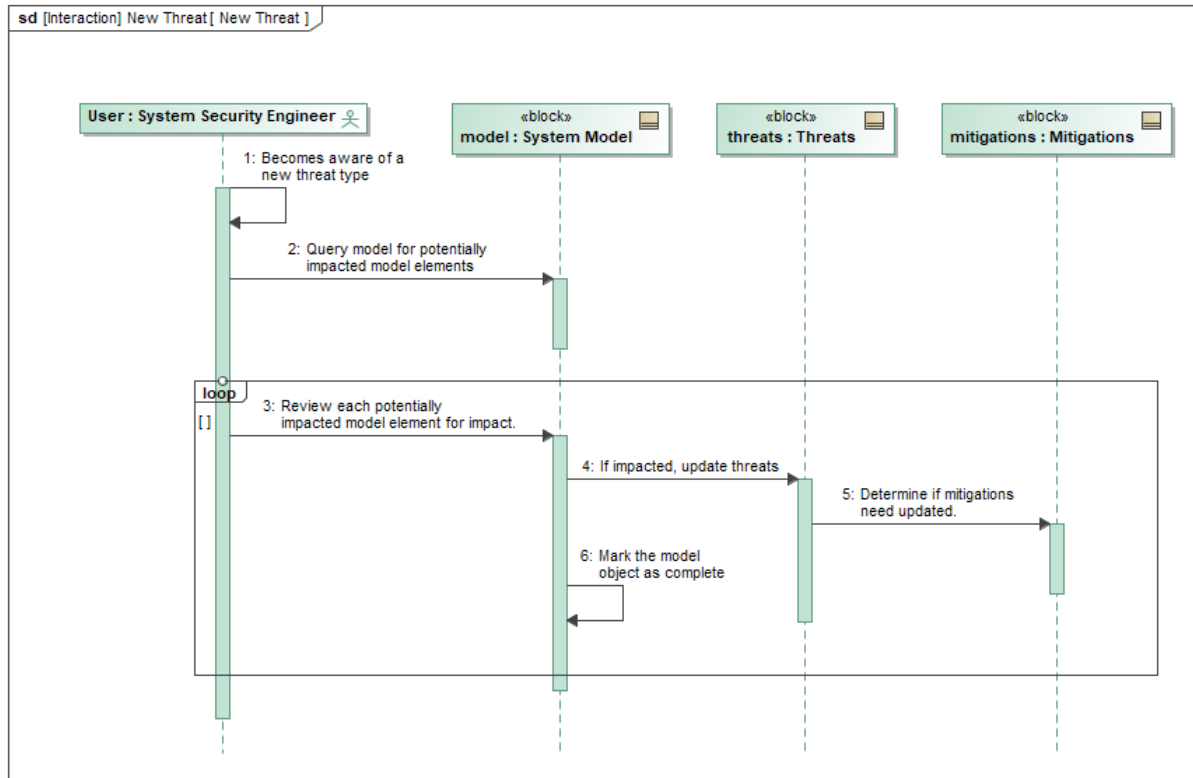


Figure 3: Process for adding a new threat

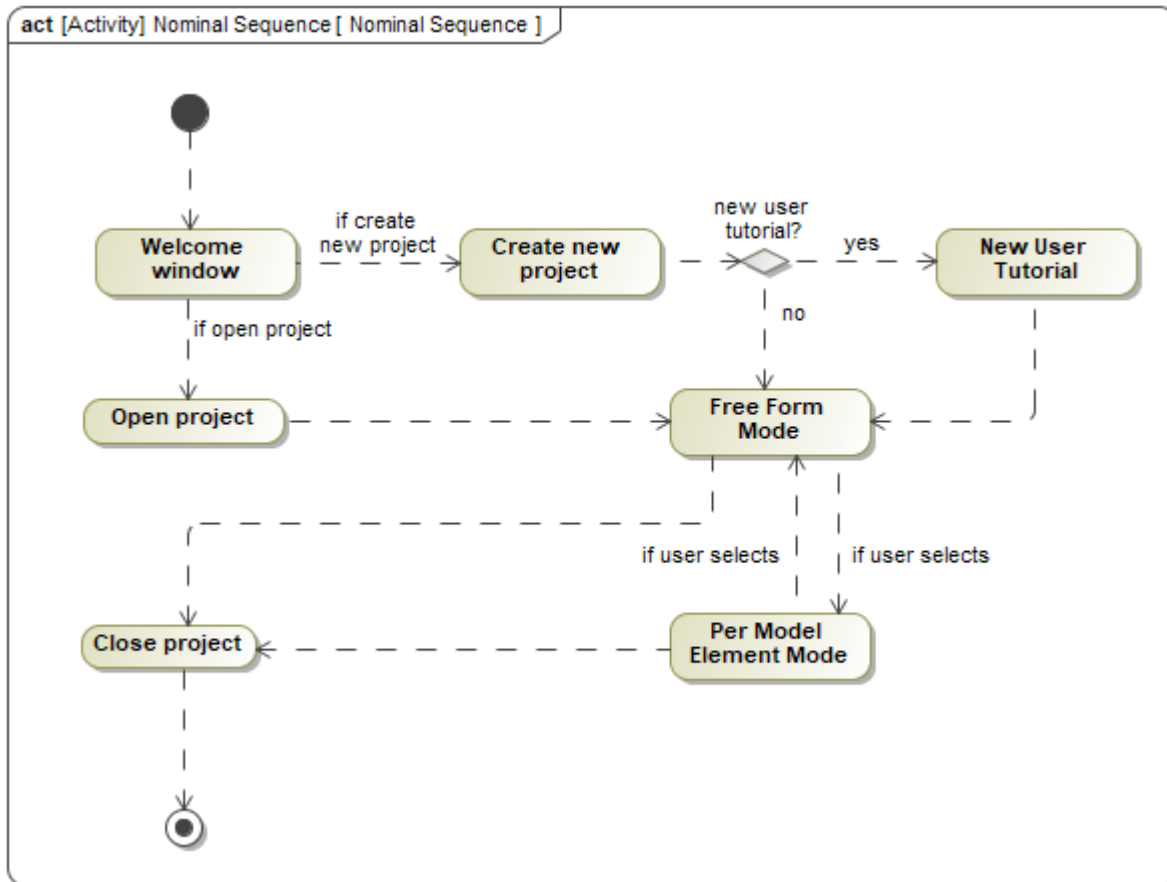


Figure 4: Nominal sequence of user interaction



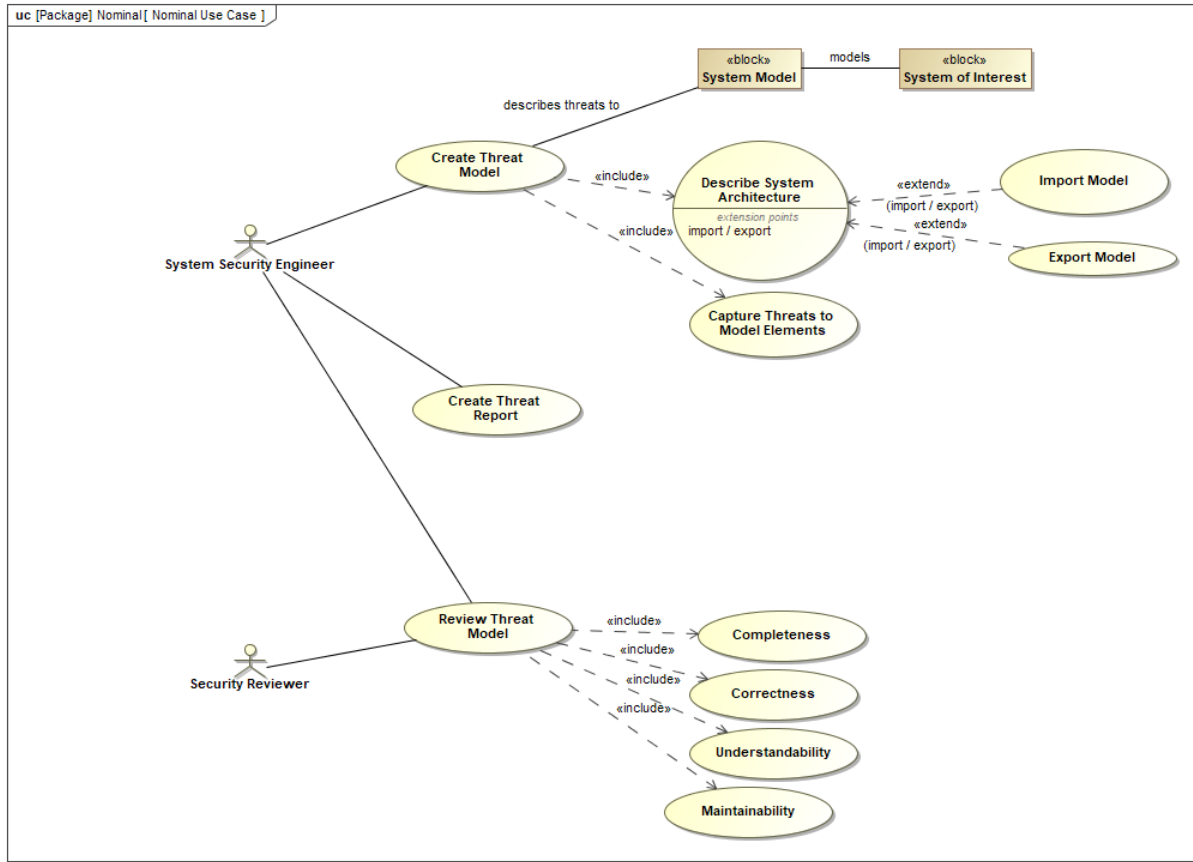


Figure 5: Nominal use cases

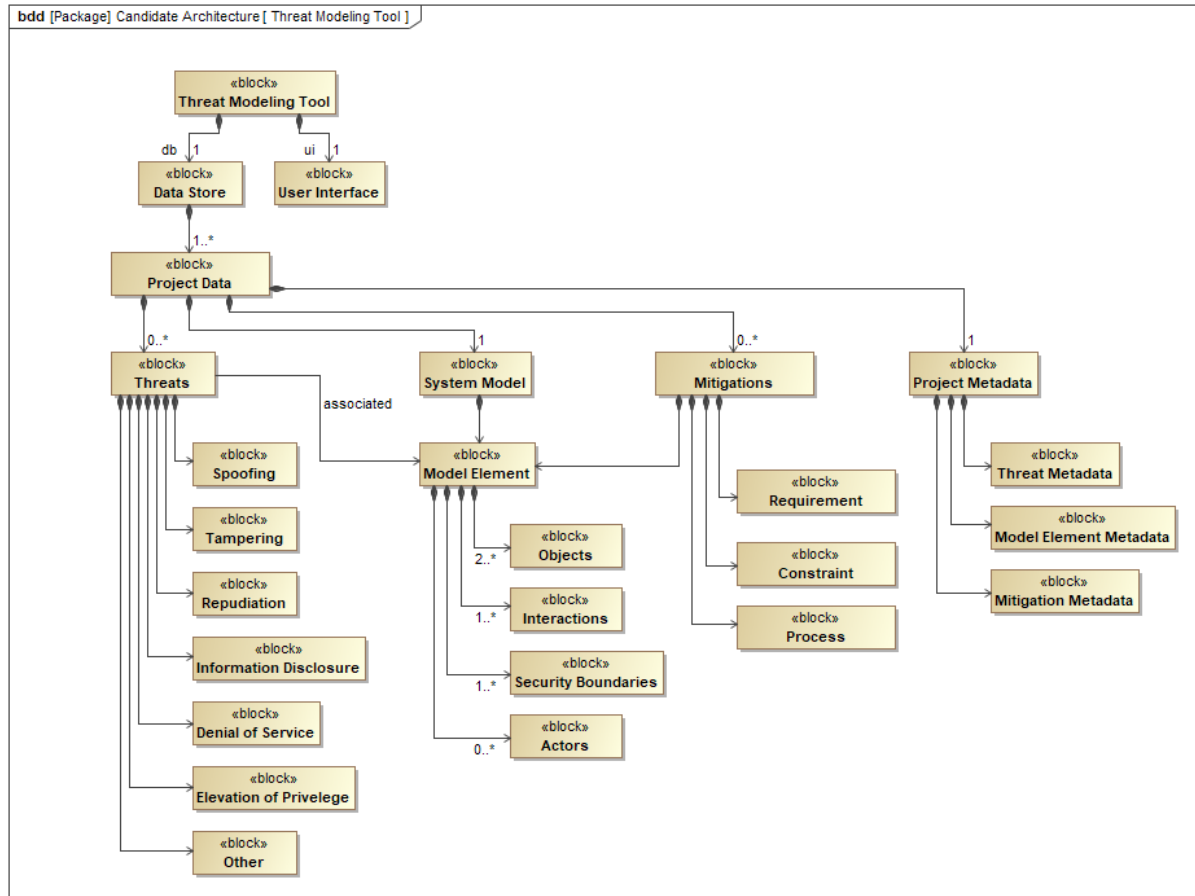


Figure 6: Block diagram of threat modeling program architecture

## TECHNICAL REQUIREMENTS

This section consists of an overview of the basic requirements for the Threat Modelling Software that the sponsor has provided to us. It provides a tentative list of requirements in multiple areas including:

### User Interface

This shall be the point of human and computer interaction and basic communication between what the user wants/needs related to what the system has to offer. Our goal is to make an extremely simple UI that any individual without software experience can use. It shall be composed of an interactive WYSIWYG (What You See Is What You Get) GUI (Graphical User Interface) that is situated in a web browser. This shall require the frontend to be written in JavaScript. The backend to handle data processing for the frontend shall be written in Python.

### Assisted Functions

This shall include help units including ToolTips for any kind of guidance that would facilitate a new user. Visual guidance (VG) will also be provided to illustrate STRIDE-based threats that have been considered on each model object. VG will have the option to turn on and off if existing users are familiar with the working of the system. Layout and color schemes will be used in order to be readable and usable for persons who are colorblind.

### Import / Export Functionality

One of the important functionalities shall be the ability of the threat modeling tool to be able to connect to various mechanism feeding in complex predictive data into an encapsulated version that is able to be exported into easily readable files including .docx (Word), and .pdf. This will essentially be not only human readable but also no software development skills will be required to perceive the threat information since all data will be stored in YAML format.

### Save and Load

The user shall be able to save the model and data at any instance of the session and this will be loaded in a future session exactly how it was left off. This includes proper exception handling in the software design to prevent loss or corruption of data.

### Version Control

From start to finish, our project will be regularly updated using GitHub which is an industry collaborative version control. This will be accessible only by developers and not users.

## STANDARDS

As this is open-source software, there are not many standards by which we are constrained. However, we will choose to adhere, at a minimum, to a standard of accessibility with regard to conditions such as colorblindness.

From a design perspective we must prioritize extendible code over designs which may make more contextual sense. If design is not modular, the open-source aspect is ignored as much effort will be required to change or add anything. This project is intended to have a long lifespan and must be treated as such.

Likewise, code must be maintainable. This project will change hands over the years, and code must be human readable and well-described. There is no place for secretive or elaborative code within this project. With the nature of what we are doing, speed is not a primary concern, as there is no complex computation in the backend.

Pre-established standards regarding code style for Python and JavaScript shall be used. These are common standards regarding specifics such as class, method, and variable name formatting, the use and placement of curly braces, etc. Variable names shall be descriptive regarding the data they represent. Since we will be using the JetBrains IDE we will have access to ReSharper which we will use as a code linter, which will enforce the standards.

Branches shall be created for all tasks in the Jira dashboard. There shall be only one task covered in an individual branch. A pull request shall be submitted for all code submissions. The request shall be reviewed and approved by a member of the team who did not write the code being merged. Under no circumstance should a change be pushed directly to the main branch. Sponsor review will be required before any releases.

## SAFETY REQUIREMENTS

There are no applicable safety requirements for our software.

## OTHER REQUIREMENTS

All libraries and packages used in the programming process must be open-source and subject to an appropriate license such as MIT or Apache.

## CONSTRAINTS

There shall be:

- No admin permissions required to use
- Source control
- Version control
- The capability to run on an average laptop
- Metadata about model elements
- Unix style time recording
- No need for training to use the software
- Colorblind adaptability
- Export, save, and load abilities
- Human readable data files in YAML or similar format
- Testable code
- Open-source licensing
- Toggleable visual features

There should be:

- Local runtime in a browser without requiring remote hosting
- Backend written in Python
- Frontend written in JavaScript

## LITERATURE SEARCH

One piece of literature relating to Threat modelling that we were referred to by our sponsor is the book *Threat Modeling: Designing for Security* by Adam Shostack. We don't have access to it yet, but our sponsor has stated that he will be purchasing a copy for each of us. The book is designed for software developers, systems managers, and security professionals to show them how to use threat modeling in the development lifecycle and in the overall software and systems design processes. This book details the process of building improved security into the design of the software, services, and systems from the very beginning.

Another piece of literature is *Ten Immutable Law of Security*, created by Scott Culp (Manes). He created these laws while being a security researcher for Microsoft. The basic premise of the laws is that there is no way to create a completely secure system. If a bad actor is determined enough, skilled enough, or has the proper access, they will get into a system. These laws can serve as a starting point when thinking about the vulnerabilities of a piece of software or a system. It is a reminder of the human element, which is often forgotten. An example is a computer that is left unlocked in a public area, such as a receptionist's desk, when the user walks away. This could give an attacker physical access to the computer. Another example is a user who does not understand a risk downloading a program that they intend to fix a problem but actually contains malware that can give remote access to the computer or the information on it.

OWASP has an article written by Victoria Drake that gives a high-level overview of threat modeling and the objectives. It lists a four-question framework, which was also mentioned in our training with our sponsor:

1. What are we working on? (Describe the system.)
2. What can go wrong? (What are the vulnerabilities?)
3. What are we going to do about it? (What are the mitigation options?)
4. Did we do a good job? (Evaluation at the end of a project cycle)

There are also links to other resources provided in the article.

Cisco published an article on its site answering the question, "What is Threat Modeling?" It elaborates on the information in the OWASP article, adding specific steps in the process, measurements of effectiveness, and various modeling methods. Many of these methods, such as the CIA method and Persona non grata, are dependent upon the person doing the modeling have well-developed skills in threat identification and mitigation. Other methods, such as STRIDE, which we will be using, and attack trees could be used by people with less developed skillsets. STRIDE is an acronym for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of Privilege. They are different ways an attacker can get access to a system or part of a system which they should not be able to access. Attack trees diagram the system as a tree, with branches being points of entry for a potential attacker.

## ORGANIZED DESIGN PROCESS

The most critical part of this project is the graphical user interface (GUI). The software being created is not computationally heavy, nor heavily constrained in backend functionality. Conversely, the success of the project is critically dependent on the user's ability to interact with the program with ease, simplicity, and with a low barrier to entry. Therefore, the frontend design takes precedence, and the backend design merely exists to support its intentions. These factors drive the process taken to design this system as opposed to a more generic, or backend-focused project.

1. Receive domain-specific training
  - a. The goal of this project is to create a software which aids an untrained user with Threat Modeling. Threat modeling is a niche topic, with few existing resources for untrained users. The project sponsor is an expert in this domain and will present us with four hours of training and reference material on the subject. This will give us the understanding needed to design the software.
2. Decide on creation methods
  - a. We will decide on what language, IDE, source control, and rules we will use for the implementation of the actual application. This is important to do early on, as it will drive later decisions. Goal based design is ideal, but design must ultimately utilize our tools and personal abilities.
3. Define GUI design goals
  - a. The GUI is where the majority of our design work is focused. Requirement-based items are only a matter of implementation, and usually don't require much thought. Additionally, goals for the backend behavior and specific capabilities are provided by the sponsor such as "ability to export," "colorblind support," "runs locally in browser," and "requires no admin abilities".
  - b. To achieve the goals of the application we will go through the resources provided and compare them with notes from our training session. We will organize the goals of the application around the step-based approach of threat modeling, with influence from the goals a threat modeling session leader has in mind when running a session. This is because our software is meant to aide in the facilitation of a threat modeling session and should guide the user without explicitly controlling them. We list out these goals with the step of threat modeling with which they correspond.
4. Define the user flow
  - a. We will create the flow of how users will use the application. This won't exactly represent the typical steps of threat modeling, as tailoring this process to software requires some extra steps. The flow also defines what should happen when a user makes a decision such as creation of a new project or loading of a past one.

5. Create GUI models of each screen in the user flow
  - a. Models of the GUI will represent our goals when creating the actual GUI. This is the most important aspect of the design process, as we will use the GUI goals to reason which tangible features should be implemented on the screen at any step in the process. For instance, if a goal is to constrain the user's ability to spend time on visual aspects, we may limit the color selection palate to 5 colors.
6. Collaborate and revise
  - a. We will discuss our thoughts on execution of the goals amongst ourselves and with the sponsor to determine what portions of our design are appropriate. This will be an iterative process with multiple rounds to settle upon something agreeable enough to progress to the next step.
7. Design backend to support GUI's goals
  - a. After the application's behavior is well defined through GUI and constraints, we will create a design for the backend to support it. Part of this may be deciding upon a certain framework to use but will certainly include creation of our own design patterns.
  - b. The backend design must also be designed to be extendable and maintainable. Our design must comply with standard design goals of software in addition to the application-specific ones.



## ALTERNATIVE APPROACHES

Our sponsor has outlined a list of requirements that do not leave room for alternative approaches. The design must have a backend that handles the data and a web-based GUI that handles user interaction and input. Ultimately, the only decisions that were left to us were the language and tool selection. The recommendation in the original requirements document provided were to use Rust for the backend and JavaScript for the front end.

Given that JavaScript is a language that runs natively in a web-browser, there aren't better alternatives. That only left a decision on the framework that would be used. The two most popular frameworks used for web-app development are Angular and React. Of those two, only Angular is a complete framework that doesn't require additional libraries to form a complete solution. With the limited timeframe provided to complete a project of this magnitude, a complete solution was preferable. Other than that difference, there aren't any significant reasons to use one over the other.

For the backend, we sought approval from our sponsor to change the language selection from Rust to Python. Three of the four members of the team already have experience using Python in a professional capacity. This limits the learning curve that would impact our timeframe. Since we will all have a steep learning curve with JavaScript, we had to limit the amount of research we would need to do to get the project off the ground. While Rust has the advantage of faster execution, the limited computational requirements of our web-app render this inconsequential.

For Integrated Development Environments (IDE), we chose to use the JetBrains portfolio. They offer solutions for all the languages that we need to use, as well as a linter that can handle coding standards. They have a great reputation for the quality of their products, and again, multiple team members have already used them, so it minimized the time taken to setup and acclimate to a new development environment. Additionally, JetBrains offers their professional products free to students for use with school projects. Other alternatives were not considered.

## DECISION MAKING

For our frontend that will be used to write the GUI, we needed to choose a JavaScript based framework. The two most popular options with an abundance of online resources are Angular from Google and React from Facebook. We did a small pros and cons:

React	
Pros:	Cons:
<ul style="list-style-type: none"><li>• Easier to learn</li><li>• An abundance of online resources</li><li>• Runs natively in browser</li></ul>	<ul style="list-style-type: none"><li>• Not a complete framework as it requires additional packages</li><li>• No team member has experience</li></ul>

*Table 1a: React frontend framework pros and cons*

Angular	
Pros:	Cons:
<ul style="list-style-type: none"><li>• Complete framework out of the box</li><li>• An abundance of online resources</li><li>• Runs natively in browser</li><li>• Aayush has experience</li></ul>	<ul style="list-style-type: none"><li>• Steeper learning curve</li></ul>

*Table 1b: Angular frontend framework pros and cons*

Based on these lists, we chose Angular. The strongest consideration was a team member having existing experience.

We also had to choose a programming language for the backend. Our sponsor had originally recommended Rust, but we were also granted permission to consider Python.

Rust	
Pros:	Cons:
<ul style="list-style-type: none"><li>• Faster execution</li><li>• Similar to Java</li><li>• Automatic garbage collection</li><li>• Memory efficient</li></ul>	<ul style="list-style-type: none"><li>• No team members have used Rust</li><li>• Slightly more difficult to implement</li><li>• Less third-party support</li><li>• Fewer online resources</li></ul>

*Table 2a: Rust backend pros and cons*

Python	
Pros:	Cons:
<ul style="list-style-type: none"><li>• Three team members have used it in a professional setting</li><li>• An abundance of open-source packages available</li><li>• Automatic garbage collection</li></ul>	<ul style="list-style-type: none"><li>• Minimal learning curve for the one team member without experience</li><li>• Less memory efficient</li><li>• Slower execution</li></ul>

*Table 2b: Python backend pros and cons*

Both have an equal number of pros and cons, but a large weight is applied to experience, and not having to learn a new language will help the team stay on track, which is essential to the completion of this project. Additionally, the faster execution time for Rust will not be a significant advantage, nor is memory efficiency since we won't be running on embedded system or other systems with memory limitations. For those reasons, we gave the edge to Python and selected it.

The final decision to make up to this point was on the Integrated Development Environment suite we would use. We had to choose between the JetBrains suite and Microsoft's Visual Studio.

<b>JetBrains</b>	
Pros:	Cons:
<ul style="list-style-type: none"> <li>• Specialized IDEs for each language</li> <li>• A large variety of add-ons for code linting and syntax highlighting</li> <li>• Built-in, user-friendly debugger</li> <li>• Professional versions available free for students</li> <li>• Built-in GitHub integration</li> <li>• Optimized for Mac and PC</li> <li>• Recommended by CS professor for JavaScript development</li> </ul>	<ul style="list-style-type: none"> <li>• Requires a different application for each language</li> </ul>

*Table 3a: JetBrains IDE pros and cons*

<b>Visual Studio</b>	
Pros:	Cons:
<ul style="list-style-type: none"> <li>• Built-in, user-friendly debugger</li> <li>• Add-ons for all necessary languages</li> <li>• Both languages can be written in the same application</li> </ul>	<ul style="list-style-type: none"> <li>• Initial setup can be difficult</li> <li>• Not optimized for Mac</li> <li>• Can be slow</li> </ul>

*Table 3b: Visual Studio pros and cons*

We were already strongly considering the JetBrains suite as one of our team members uses PyCharm, the JetBrains Python IDE, professionally. The pros and cons lists confirmed the selection.

Our decision-making process to this point has largely been dependent on our experiences and existing skillsets. We have had to make decisions based on what we know because we will not have time to learn a large number new of things. As we progress through the development cycle, we will encounter a lot of issues with which we don't have experience, and this will require the use of a formal decision-making process. We plan to use a combination of Pros and Cons and Pugh Matrix to help make decisions that are outside of our immediate field of knowledge. These will allow us to gather information about our options, how well they solve the problem, and their shortcomings.

Going forward, for minor decisions or decisions with few variables, we can use a simple Pros and Cons list. The advantage is that it is quick, and combined with some common sense, can be used to make a very fast decision. As time is of the essence, fast is a priority.

For more complex decisions, we will use a Pugh matrix. The matrix will contain columns with the different options available that meet some or all the needs of the challenge being addressed. The rows will contain the criteria by which we will evaluate the addressing of the needs. The overall goal will be to find the tool that satisfies the largest, and most important, needs. However, when designing software that is meant to be extended, as open-source software is, it is sometimes important to select options that exceed the need at present because it may save time in the future by preventing a large refactor if the option that was selected doesn't meet a need that may arise. The goal is to build a "futureproof" base for later forks of our project. If core parts of

the base must be continually refactored to add new functionality, the project won't ultimately be extended, and it will become a dead project and worthless as open-source software. However, again, time is of the essence, so it is important not to select an option that will delay the development process more than necessary. Therefore, a criteria added to every decision matrix will be ease of implementation.

## DESIGN CHALLENGES

A significant challenge in design of the GUI of our software is the team's lack of collective experience in frontend work. Some team members have limited experience in this domain, but not at the level required for the project. The GUI aspect that will challenge our lack of experience the most is the model creation tool. This needs to allow the user to place predefined and scalable shapes onto an infinite grid and allow connection between different shapes as well as the ability to link the objects with related text items. The level of intractability this requires is very advanced and designing the software in such a way that we can achieve this will be very difficult.

Another challenge is that the sponsor wishes for the application to guide the user through the threat modeling process in a way that teaches a "security mindset." This is an abstract idea and requires the application to guide the user without telling them every detail explicitly. Otherwise, they would simply memorize steps instead of a set of ideas and objectives to consider, as we intend to teach. To fulfill this, we must design with a balance between open-ended abilities to encourage thought, and restrictive features to ensure the user doesn't stray too far from what they should be focusing on at any given time. By nature, these goals contrast, leading to difficult choices between possible GUI layouts.

Furthermore, a central goal of this project is to lower the barrier to entry for threat modeling and security in general. As a result of this, we must avoid technical phrasing throughout the application, and replace it with phrasing which describes the idea indirectly. Another implication of this is that the design must take high-level steps within the security domain and distil them into sequences which a novice can follow while still accomplishing their original objectives.

## POTENTIAL RISKS AND MITIGATION PLANS

Due to the nature of this project, there are little to no risks which we may run into. Especially as it pertains to safety or malfunction. The threat modeling software has no hardware component and has no internet capabilities. All its components are deterministic, and any misbehavior of the application is simply a bug which is technically behaving as defined.

With any software, a concern is memory access and manipulation. Since our program has the ability to save and load, as well as input/output this is possible. However, these concerns are mitigated by simply using a modern program language. Modern languages handle garbage collection inherently, and do not allow unsafe memory access.

Another universal software risk is hidden bugs. These do not present themselves in the current state of the application, or most parts of it, but eventually these bugs appear. When they do, so much time has passed, that it is difficult to determine what is causing them, and fixes become increasingly costly as time progresses. The solution to this is thorough test coverage. This catches things that may not present themselves in actual use of the software by the user, but are still present bugs, nonetheless. Testing also works in the reverse, such that additions to the software that would break a previously working feature will be caught.

Due to the team-based nature of this project, it is possible to have issues with source control. Though tools like Git are the basis for modern software development, they don't prevent all issues. A team member could potentially push broken code to master, thereby ruining it. This is mitigated through caution, but when it inevitably occurs, the command `git reset --soft HEAD~` will fix rollback master to the last version. Merge conflicts are similarly unavoidable. They can be lessened by team members agreeing on a code linter to reformat code so that Git can differ between format change and actual content changes. But when this isn't enough, merge tools (such as P4Merge) allow a human to resolve conflicts between different files manually.

Ultimately, the largest risk we run is not completing the project on time or failing to meet the arbitrary guidelines set by the class schedule. Software is component based, and component ideas and interactions evolve throughout the development process. While we are expected to have our final project, designs figured out by the end of the semester, the Agile process will render anything we come up with by that time obsolete by the end of the project. As we move through the design project, ideas will change and features that seemed important will be superseded by newer ideas. At best, by the end of the semester, we'll have a segment of the program designed. The best way to minimize the risk of falling behind is to maintain extendibility in our code and cover the basics first, producing a minimum viable product. With proper design, we can then add additional features once the minimums have been met.

## DESIGN DETAILS

Our approach to design the application centered around its GUI capabilities, with user experience taking precedence over speed or memory performance. We chose to execute this design approach by defining goals such as discouraging the user from spending time on aesthetics, capturing thought processes, encouraging different thought perspectives from each user, and avoiding any decision making within the threat modeling process.

At this time we have corresponded with our sponsor to confirm our design choices that pertain to workflow and backend design choices as well as the initial GUI design. Our design will almost certainly change as time progresses, but the starting design has been solidified, and will be presented below. It is primarily centered upon the meeting with our sponsor to learn about the domain, the “four-question” framework, and STRIDE methodologies explained in the Literature Search section of this document and presented to us by our sponsor.

Our project’s design does not need to heavily consider risks or alternative approaches. Many technical requirements are thoroughly specified, so much of the design work is centered around user experience. Likewise, little literature exists on our domain, and we will continue to rely upon the resources provided by our sponsor.

## DEVELOPMENT DESIGN APPROACH

- The backend will be written using Python
  - Python is one of the most popular and easy-to-learn languages, which is important for open-source projects
  - It has many open-source, third-party packages which can speed up the development process.
  - It manages memory automatically and minimizes allocation issues
  - There are various API’s to facilitate data transfer between the backend and front-end
- The front-end will be written using the Angular framework
  - Angular is developed by Google, so it will likely be supported long-term
  - It is a complete front-end framework, which minimizes the time spent looking for third party add-ons to complete the framework
  - An accompanying document outlines some of the basic GUI elements that the front end will be built to emulate
- The Agile development process will be used to avoid large refactors due to errors early in the development process
  - We will use a virtual Kanban board using Jira to assign and move tasks through the development process
  - Code reviews will be performed by fellow team members
  - Progress from each week will be demonstrated at the team meeting

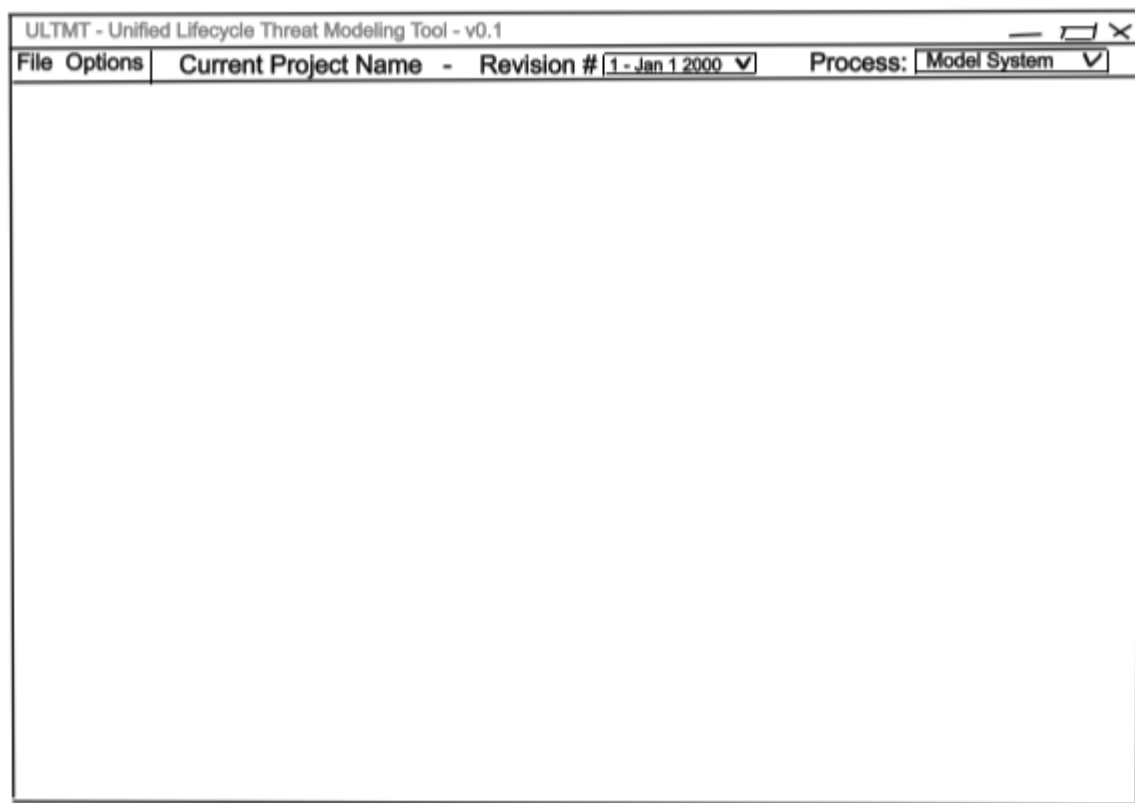
## GUI CENTERED DESIGN

Our approach to design of the application centered around its GUI capabilities, with usability and features at the forefront. Therefore, our design does not include a backend perspective. Backend work will be done purely to support the front-end functionality presented here. The features of the software are outlined via the user-flow of the program. The tool is loosely based on the “four-questions” discussed in Literature Review. Although to align with the goal of creating a tool with a low barrier to entry, extra steps are added to the process. This does not change its form or intent, but does make the process easier to follow, creating a learn-by-doing experience for the user.



*Figure 7: Screen 1, Initial*

The initial screen of the program simply shows load and new options. It has standard application interface features such as file, options, minimize, expand, and exit. It shows “Ultimate” which is the English equivalent of the acronym for the software, along with a short description of what the mission of the tool is. The acronym stands for Unified Lifecycle Threat Modeling Tool, which highlights the tool’s emphasis on collaboration and lifecycle support. The current version of the software is shown above the toolbar of the program. Load allows the user to open an old project to modify. This aligns with the goal of allowing users to support a product throughout its entire lifecycle.



*Figure 8: Screen 2, New Project*

Upon creation of a new project, users see new boxes appear. One of these is the current filename of the project, which was inputted by the user. Along with it is the current revision number of the user's project – not to be confused with the version number of the software. This box functions as a dropdown, allowing the user to see past versions of their file. This can be useful to track where changes to the threat model came from, and serve as a living document of all threat-modeling-related work on a project. The file will require manual saving, as AutoSaving is complex and creates complications with “undo” and “redo” capabilities, as well as version hashing utilizing SHA-256. AutoSaving would be an appropriate “stretch-goal” in the future.

The “Process” dropdown box also appears. This allows the user to progress through the steps of threat modeling, with a clear understanding of what they should be doing at each step. This aligns with the idea behind the “4-step” approach, and one of the goals presented by our sponsor: the tool should constrain the user in their abilities so that they do not become distracted or focused on specific details, which are outside the scope of their current task. The step-dropdown feature prevents the user from moving ahead into future steps unless they have completed the prior steps. Although it may be desirable to have a separate banner appear at the top which shows all process steps, highlights the current one, and greys out the rest for easier navigation.



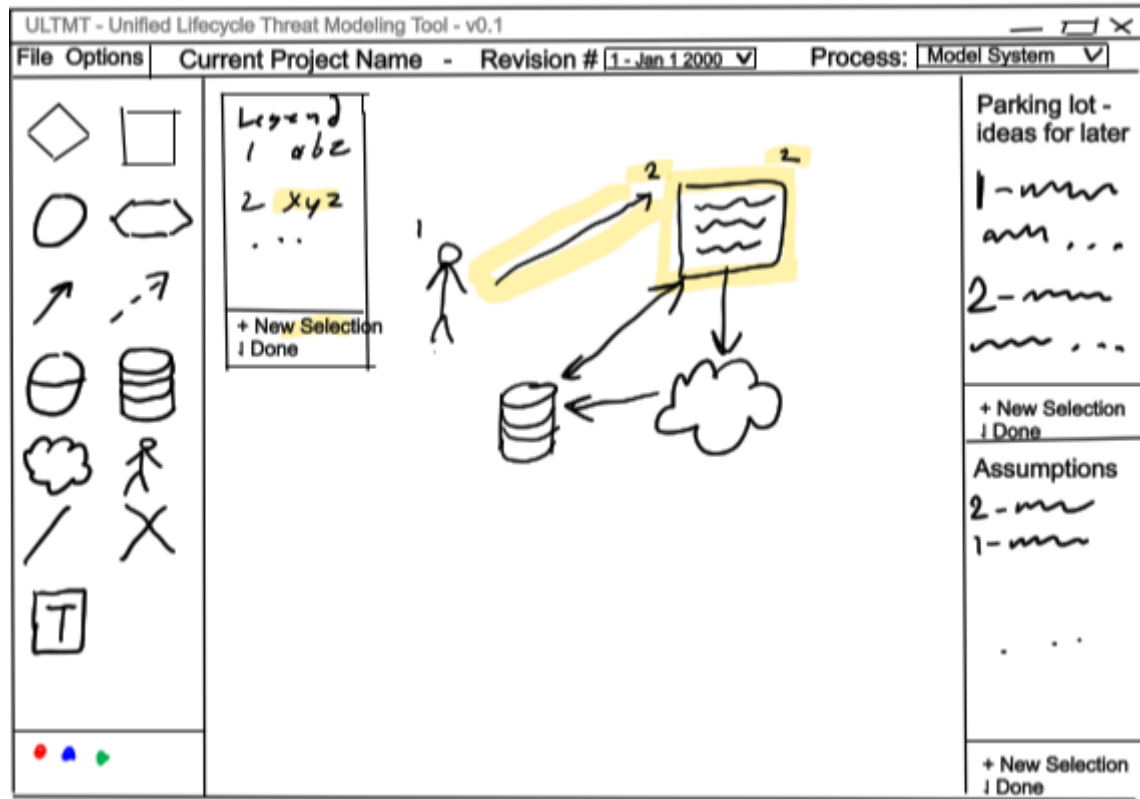


Figure 9: Screen 3, Model System

The first step in the modeling process is to model the system. The “Model System” process enables the user to create a visual diagram of their system. However, it does not constrain the user into a certain type of visual aid or modeling approach. Although our software shall support future addition of model templates should that be deemed useful later on. We also have a stretch-goal of including the ability to allow multiple views of a system through the use of an additional dropdown, such as “manufacturing” or “use-case.” Our design does not force the user to chose a specific type of model as this leads to time focusing on making the model fit “proper form.” Instead a pane appears on the left side of the window, which contains drag-and-drop elements that one would typical find in system-models, UML diagrams, general shapes, lines, and textboxes. However for the same reasons mentioned before, there are not an exceedingly large amount of options within this panel, limiting the time and thought a user spends on selecting visual components.

There is also a legend in the top right of the diagram pane. This allows the user to select individual components (highlighting as they go) and name them as a grouping (either with numbers or single-word descriptors). The legend feature seems counterproductive towards the goals of our work, but some level of visual organization is necessary.

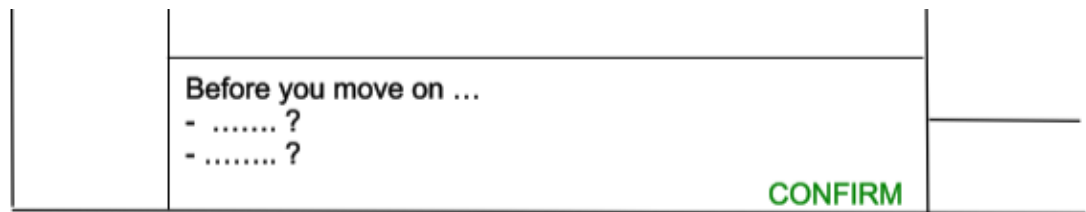
The parking-lot pane implements the parking-lot methodology in group design – it is a place where users may place useful ideas to be discussed later, but would side-track the current focus of an ongoing discussion. The assumption panel tracks assumptions made during the creation

of the system model. This most utilized in early stages of a product’s lifecycle, as many details are not known yet. A stretch-goal for both of these features would be the same highlight/multi-select functionality that the legend possesses, but it is not necessary. Both panes can be expanded by clicking on them, which allows a full listing of all items instead of the two-line summary shown on the right normally.

For all text features, the user will be constrained to one font, one color, and a 100-word limit (the limit shall be configurable under “options”). This reduces bloat and forces the user to truly convey the idea at hand, instead of verbose technical jargon. Likewise, are be limited to basic rainbow colors.

The user can hover over items to make a tooltip appear which explains the intent of the related feature. This aligns with one of the key intents of this tool: it should help guide the user through the threat modeling process in a learn-by-doing fashion. The user can also zoom using the scroll-wheel of the mouse.

All aforementioned features reinforce that the model being evaluated is not the focus of their work. The focus is the thoughts provoked about security within the system and capturing those thoughts to analyze later in the process.



Before you move on ...

- ..... ?
- ..... ?

CONFIRM

*Figure 10: Screen 3, Model System Confirmation*

When the user uses the process dropdown box to move to the next step in the sequence, a popup will appear asking them to confirm that their model meets certain criteria. This aligns with our software’s intention to create a “threat-modeling mindset” within the user, while they learn by doing and do not explicitly learn instructional material. Examples of some of the questions shown to a user include: “is the scope appropriate?”, “are different viewpoints of the system accounted for and aligned within this model?”, “Are the ideas captured effectively, even if they are not perfect?”, and “are all assumptions made recorded?” etc.

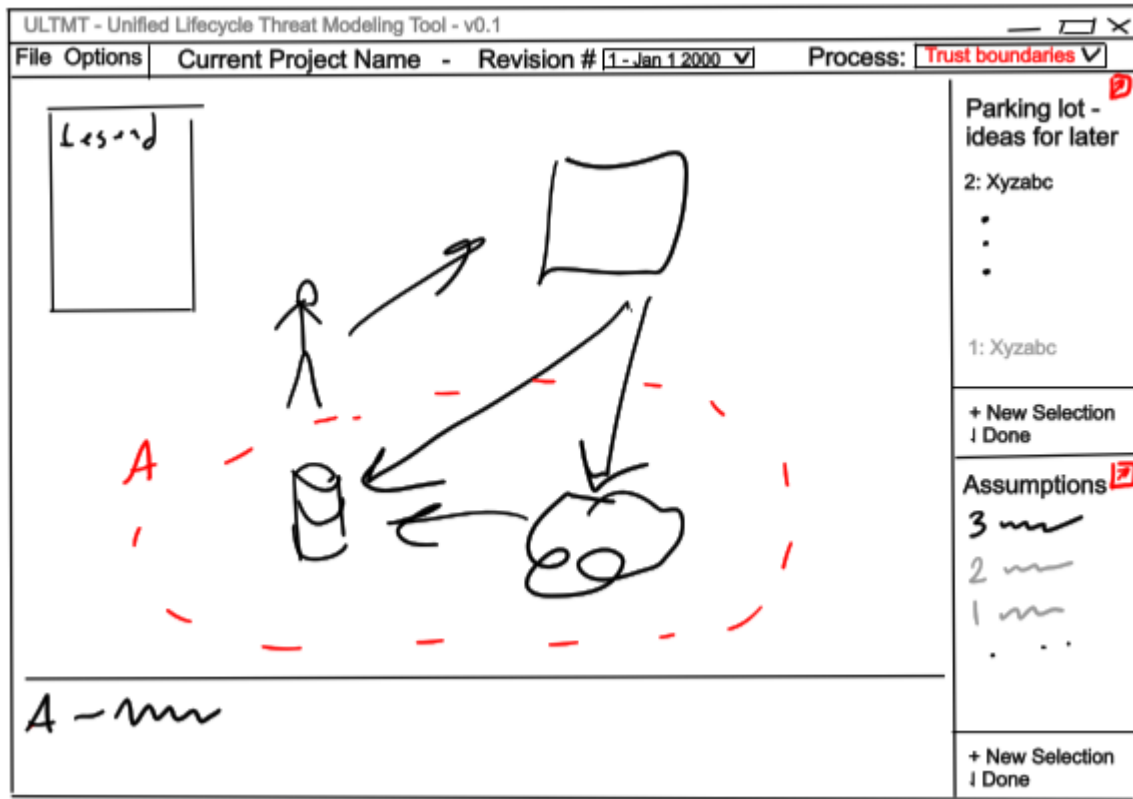


Figure 11: Screen 4, Trust Boundaries

The next process is the trust boundaries stage, where the user can draw an outline within their model to represent this boundary. A new pane appears at the bottom which defines trust boundaries, and then allows the user to type a description with a length the same as mentioned prior. These boundaries are colored uniquely and labeled within the pane automatically.

As the user progresses, the parking lot and assumptions panels introduced earlier are persistent. When parking lot items have been resolved, they can be double clicked which turns them grey and places them at the end of the list. Assumption items from past stages are also turned into a unique color to clarify that they do not correspond with the current process.


		Threat Identificat
<p><b>Dump everyone's INDIVIDUAL initial thoughts on possible threats here. The next step will allow group discussion to flesh them and the rest out in an organized manner.</b></p> 		<p><b>Parking lot - Ideas for later</b></p> <p>2: Xyzabc</p> <ul style="list-style-type: none"> <li>.</li> <li>.</li> <li>.</li> </ul> <p>1: Xyzabc</p> <p>+ New Selection ↓ Done</p>
<p><b>New Potential Threat:</b></p> <p>Avoid trying to think from the viewpoint of an attacker or compliances...</p> <p><b>Parts affected:</b></p> <p></p> <p><b>Impact:</b></p> <p></p> <p><b>Related subdomains:</b></p> <p></p> <p><b>Suggested by:</b></p> <p></p> <p><b>List so far:</b></p> <ul style="list-style-type: none"> <li>- qwertyulopasdghjklzxcvbnm ..</li> <li>- qwertyulopasdghjklzxcvbnm ..</li> <li>- qwertyulopasdghjklzxcvbnm ..</li> </ul>		<p><b>Assumptions</b></p> <p>3: Xyzabc</p> <ul style="list-style-type: none"> <li>.</li> <li>.</li> <li>.</li> </ul> <p>2:xyzabc 1: Xyzabc</p> <p>+ New Selection ↓ Done</p>

Figure 12: Screen 5, Threat Identification

The threat identification process aligns with step two in the “four-question” method mentioned within Literature Review. A message appears at the top of the screen explaining the intent of this process: “Dump everyone’s INDIVIDUAL initial thoughts on possible threats here. The NEXT step will allow group discussion to flesh them out. This process seeks to capture them.” This provides clarity, but mostly limits scope creep and time spent within this step.

A new pane appears, splitting the model window vertically. It has multiple text boxes to guide the user into capturing information surrounding the problems, but not solutions. The field names are as generic as possible in order to focus on capturing significant thoughts: not emphasizing form or location. These include a description, parts of the system affected, and impact. All three of which will always be present. The related subdomains field is not necessary but encourages everyone participating in the use of the tool to have domain-specific thinking, as well as consider domains outside of their own. The suggested by field is simply useful when looking back at the file in the future to retrace events and is included for documentation purposes. Hovering over the top of this pane shows a tooltip with an example threat written out.

Once a threat is completed, it will be added to the list directly under it. Clicking on any existing threat allows the user to further edit it. A stretch-goal is to create the ability to map these threats to the visual model on the left with markers, or even additional model symbols. Markers would allow for clarity but adding more model-symbols may stray from our goal of fostering user focus on thought over form.

Process: Threat Discussion & Completion >

Select a specific threat modeling tool (optional):

STRIDE

Description ...

Typical use cases...

Kill Chain

Description ...

Typical use cases...

...

↑

↓

Parking lot - Ideas for later

2: Xyzabc

•

•

1: Xyzabc

+ New Selection

↓ Done

Assumptions

3: Xyzabc

•

•

2:xyzabc

1: Xyzabc

+ New Selection

↓ Done

Figure 13: Screen 6, Threat Discussion and Completion, Tool Selection

The next process aligns with the third step in the four-question framework. This is where threat discussion and completion happens based off of the prior capture phase. If the user did not already select a threat analysis framework in the initialization of their project, a pop up will appear like the one shown above. It shows multiple common frameworks such as STRIDE and kill chain. Each item listed has a short description and typical use cases to help the user select which one they would like to use for the discussion and completion stage. This aligns with the tool's goal of creating a lower barrier to entry as well as diverse use cases.

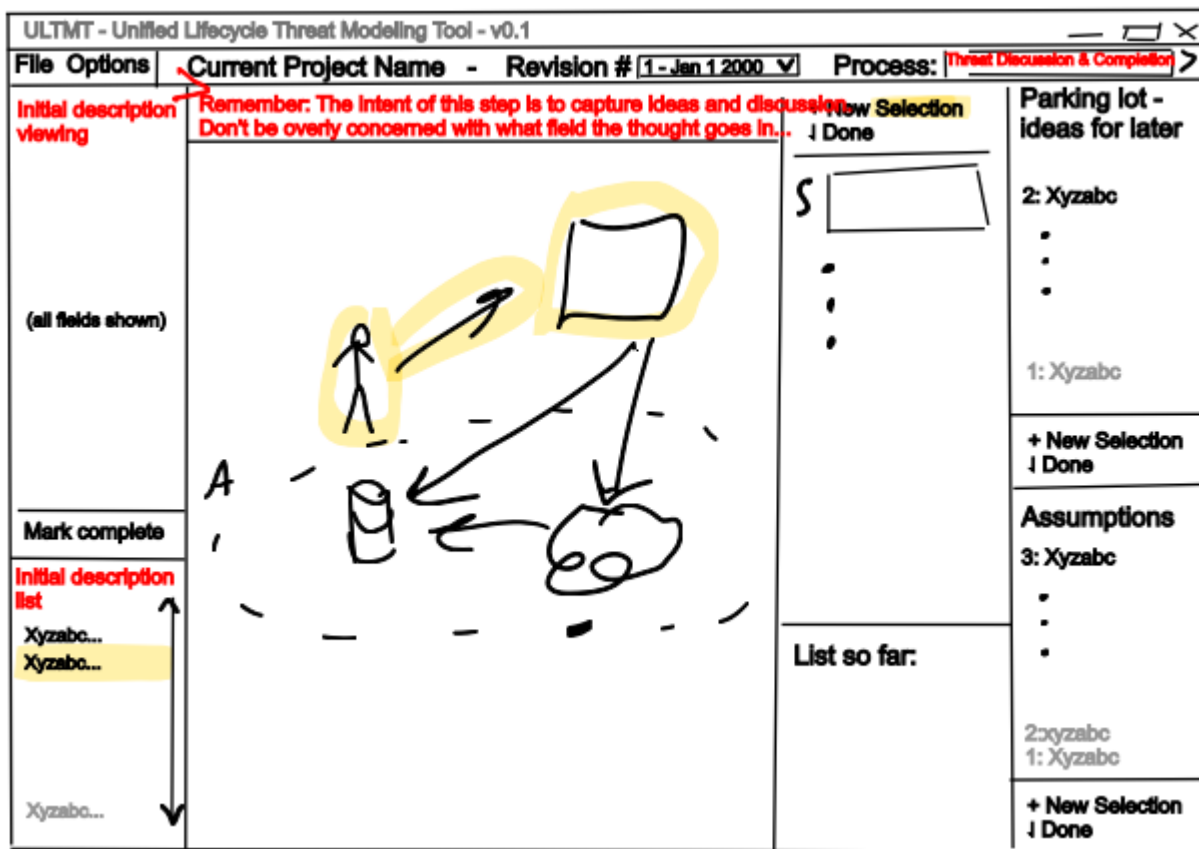


Figure 14: Screen 6, Threat Discussion and Completion

After the pop-up is addressed, the main screen of the Threat Discussion and Completion shows. It looks similar to before, as the parking lot, assumptions, and model are persistent. However, the threats identified from the prior process appear in a pane now at the bottom left of the window. When clicked, the fields of that threat (from before: description, impact, etc.) show in a pane directly above it for reference as the threat discussion occurs amongst the users. As an extension of the stretch-goal mentioned earlier (threats associated with model components), the relevant model components become highlighted upon selection of the threat. On the right side of the model panel, two more new panels appear. The top is where the user can fill out the details of the threat discussion using the framework they selected prior.

A note at the top of the model pane aligns with the tool's goal of scope maintenance and thought capture over form. It reminds the user "the intent of this step is to capture ideas and discussion. Don't be overly concerned with what field the thought goes in." Many parts of the tool do not have explicit instructions in order to align with the goal of learning-by-doing, but this note is necessary given the large amount going on at once in this specific process.


ULTMT - Unified Lifecycle Threat Modeling Tool - v0.1			
File Options		Current Project Name - Revision # 1-Jan-12000	Process: Mitigate threats
<p>Threat description viewing</p> <p>(all fields shown)</p>		<p>Remember: The intent of this step is to capture possible ideas, possible mitigations, and the discussion that led you there... Not finite action</p> 	
<p>Threat description list</p> <p>Xyzabc...</p> <p>Xyzabc...</p> <p>Xyzabc...</p>		<p>Eliminate</p> <p>Mitigate</p> <p>Accept</p> <p>Transfer</p> <p>Mark complete</p> <p>List so far:</p>	
		<p>Parking lot - ideas for later</p> <p>2: Xyzabc</p> <p>1: Xyzabc</p> <p>+ New Selection</p> <p>↓ Done</p>	
		<p>Assumptions</p> <p>3: Xyzabc</p> <p>2:xyzabc</p> <p>1: Xyzabc</p> <p>+ New Selection</p> <p>↓ Done</p>	

Figure 15: Screen 7, Threat Mitigation

For the mitigation process, all panels shown before persist except for the threat discussion panel. It is replaced by a panel with five textboxes, allowing the user to describe mitigation details generically, which aligns with the tool's goal of content over form. Additionally, the left-hand panel for threat viewing now shows the more detailed threat discussion captured in the prior process. When the user selects one of the threats listed on the left (same as before), they can fill out this new panel. Any proposed solution has a description, and fields to note how the mitigation eliminates, mitigates, accepts, or transfers aspects of the threat currently selected. Hovering over any of these terms shows an example, and a definition of the term as it pertains to threat mitigation.

In the case that two or more possible approaches are discussed, the user can press the new selection button to add another. The panel directly under this accumulates the list of possible mitigations associated with the relevant threat. It will not always be a one-to-one mapping. Once discussion concludes for the time being, the threat becomes grey and moves to the end of the list.

For the same reasons mentioned prior, a message is displayed at the top of the model pane to remind the user of their goals.

ULTMT - Unified Lifecycle Threat Modeling Tool - v0.1			
<b>File</b>	<b>Options</b>	<b>Current Project Name</b> - <b>Revision #</b> 1-Jan 1 2000	<b>Process:</b> Process Validation
<p>This step in the process is to evaluate how well the team's efforts have unified into the threat model created. It should run like a retrospective, capture the occurring discussion flow, and aim to answer questions such as:</p> <ul style="list-style-type: none"> <li>- Did we agree what we were working on?</li> <li>- Did we have specific ideas on what could go wrong in every part of the system?</li> <li>- Did we think of appropriate mitigations for these?</li> <li>- Did we capture thought over form?</li> <li>- Did we need more information?</li> <li>- Are all views accounted for?</li> <li>- How was participation throughout?</li> </ul>			<b>Discussion Thread:</b> <div style="text-align: center;">↑ ↓</div>
<div> <div> <input type="checkbox"/> <input checked="" type="checkbox"/> </div> <div> I like I wonder </div> </div>			<b>Conclusions/ Actions</b> <div> <div></div> <div></div> <div></div> </div>

Figure 16: Screen 8, Process Validation (With Example Use)

Process validation is the last step in the four-questions. The intent is to provide a forum to discuss the threat modeling process and experience of those who participated. To facilitate this, the tool has the most common questions to consider during this process listed at the top in order to encourage productive and valuable discussion. It is done here in an AGILE development manner.

Instead of having users select a specific retrospective style, the tools are provided to create one on a blank canvas (line segment and textbox). This forces users to focus on content over form, and regardless, creating templates for all exhaustive retro-styles is impossible. A stretch-goal is to create templates for the most common retrospective styles.

As discussion occurs, one team member usually takes note on the discussion to refer to later. A new panel on the right-hand side appears for this. It functions simply as a bulleted list. Similarly, a conclusion/ action panel appears directly under it. After discussion of retrospective items concludes, potential actions are often derived so that they can be accounted in a task tracking service.



ULTMT - Unified Lifecycle Threat Modeling Tool - v0.1

File Options    Current Project Name    Revision # 1 - Jan 1 2000    Process: Export Results

Current Project Name, Revision #

Numbered legend

1 ...  
2 ...  
...

1,2,A - Full threat details in model form  
- possible mitigation details

3,B - Full threat details in model form  
- possible mitigation details

...

Discussion Thread:  
~~~~~

Action Items:  
~~~~~

I like	I wonder

Export Results

Save within project and quit

Figure 17: Screen 9, Export Results

Although all detail will be captured by the tool and saved in the user's file with the ability to view by revision and process, there are multiple reasons to allow for a concise export of all data. A simple PDF is much easier to share and reference after the fact, especially if edits do not need to be made. Another reason is that the program's flow is intended to encourage and capture human thought, which leads to a lack of coherence. There is little to no focus during the threat modeling process of form or cleanliness – in fact, this is the opposite of the goal (content over form).

The tool shows a preview of the compiled PDF, and allows the user to edit the text fields from before. This is mostly due to the fact that almost all text fields have character limits on them initially and are intended to be done quickly, while the PDF is meant to be presentable and coherent.

The general layout of the document is shown above. The complete model is shown enlarged along with the legend for the model, trust boundaries, and threat items. All assumptions made are listed along with the corresponding process step they were created in, along with the legend reference of the model items they refer to. Any unresolved parking lot items appear as well. All retro items and action items are listed at the end.

This entire process step is considered a stretch-goal at the time, as it is not essential for the functioning of the tool. It should also be simple to add later by other developers. Though it is the highest priority stretch-goal as it adds much polish to the overall product.

## TESTING

Testing will be done in multiple phases throughout the design process. Since the Agile design process will be used, automated unit testing will be done for each section of code before presenting it to the group at the weekly standup meeting. A task will not be considered complete until the code has successfully completed unit testing. When a group of related tasks have been completed and each individual component has been unit tested, integration testing will be performed. For the backend tasks, the PyTest module included in the default Python libraries will be used. It includes all the features of the older unittest module, but also offers more than 800 external plug-ins.

<b>unittest</b>	
Pros:	Cons:
<ul style="list-style-type: none"><li>• Automated testing process</li></ul>	<ul style="list-style-type: none"><li>• No external plug-ins</li></ul>

*Table 4a: Pros and cons of the unittest testing module.*

<b>PyTest</b>	
Pros:	Cons:
<ul style="list-style-type: none"><li>• Automated testing process</li></ul>	<ul style="list-style-type: none"><li>• Over 800 external plug-ins to speed up creation of test cases</li></ul>

*Table 4b: Pros and const of the PyTest testing module.*

For frontend components, JEST will be used to test inputs and output and the responses from the GUI. The user-friendly syntax and documentation was the primary deciding factor. This can be used to assert that certain output values are the expected values and that components that are expected to be displayed are displayed. As more modules are built, larger integration tests will be written to test interoperability.

<b>Mocha/Chai</b>	
Pros:	Cons:
<ul style="list-style-type: none"><li>• Automated testing process</li><li>• Can continuously test or perform tests on-demand</li><li>• Can test complex functionality</li><li>• Human-readable syntax</li><li>• Free</li><li>• One team member has experience, though the experience is limited</li></ul>	<ul style="list-style-type: none"><li>• Complicated syntax can be confusing for newer users</li><li>• Documentation isn't always clear</li></ul>

*Table 4a: Pros and cons of the unittest testing module.*

JEST	
Pros:	Cons:
<ul style="list-style-type: none"> <li>• Simple syntax that is human-readable</li> <li>• Good documentation available online</li> <li>• Can test complex functionality</li> </ul>	<ul style="list-style-type: none"> <li>• No group member has experience</li> </ul>

*Table 4b: Pros and cons of the PyTest testing module.*

In addition to software testing, hands-on testing will be performed by each member of the team. This will be a quick way to test basic functionality. The advantage of using hands-on testing is that it does not require extensive test cases to be written and can more quickly identify small problems that can be debugged. However, automated testing is important because it can continue to automatically test code as more components are added without requiring the team member to manually complete the tasks repeatedly.

## ABET OUTCOMES

### Abet Outcome 2:

- Implementing threat modeling into product design allows for safety to become a more important and more central part of the design process. This is shown in the Motivation section of our paper. Implementing threat modeling early in the design process reduces the risk to end users of having their data exposed or giving an attacker access or control they shouldn't have. Economic factors are also considered due to the implementation of threat modeling into the early lifespan of a product being cheaper than in the middle of a product. This is shown in the Background section of our document.

### Abet Outcome 7:

- As a group, we learned the entire outline of the threat modeling process from our sponsor in a four-hour class. This can be seen in the GUI design of our project. It guides the user through the threat modeling process, as we were taught.

## APPENDICES

### LIST OF ABBREVIATIONS

SHA-256	256 Secure Hash Algorithm
GUI	Graphical User-Interface
API	Application Program Interface
IDE	Integrated Development Environment
UML	Unified Modeling Language
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service,
ECG	Echocardiogram
PDF	Portable Document Format
VG	Visual guidance
UI	User Interface
OWASP	Open Web Application Security Project

*Table 5: List of Abbreviations*

### BILL OF MATERIALS

There is no bill of materials for this project. Python and Angular are open-source languages that do not require any licensing. The PyTest package for testing the Python code for the backend is open-source. JEST, used for testing the frontend, is also open source. The JetBrains IDEs used are free for students through their partnership with GitHub. They only require a GitHub Student account linked to a JetBrains account.

## LESSONS LEARNED

- Joe
  - I learned how threat modeling is done and the importance of incorporating it into the beginning stages of a product. I also learned the importance of choosing a correct programming language for the project at the beginning stages. There are more things to consider when deciding languages than previously thought.
- Aayush
  - I learned the different aspects of Threat Modeling and how much of an effect it has on project when done early in the development cycle. Until this point in the entirety of the project, I learnt about each stage of a Software Development project including Requirement Analysis, converting those to deliverables, and much more.
- Cooper
  - I learned how difficult it is to design a GUI that makes complex concepts seem simple to the user while still maintaining clarity and all the features that the program is required to have. In addition, I learned more about the processes which occur between a production team and sponsor to collaboratively create what the sponsor wants. There is more discussion than I anticipated, as well as uncertainty – which is far more realistic than any regular school assignment I had experienced prior. The entire concept of threat modeling was new to me, and I have spent most of the time learning more about what makes it effective.
- Kristopher
  - I learned how to do threat modeling and the importance of incorporating it from the very beginning of the research and design process. I also learned the possible negative consequences of addressing it too late in the process.

## REFERENCES

- Drake, Victoria. "Threat Modeling." *Threat Modeling*, OWASP Foundation, [www.owasp.org/www-community/Threat\\_Modeling](http://www.owasp.org/www-community/Threat_Modeling). Accessed October 16, 2022.
- Manes, Casper. "Security 101 - the 10 Immutable Laws of Security Revisited." *TechTalk*, GFI Software, 25 Aug. 2015, [www.techtalk.gfi.com/security-101-the-10-immutable-laws-of-security-revisited/](http://www.techtalk.gfi.com/security-101-the-10-immutable-laws-of-security-revisited/). Accessed October 14, 2022.
- Mohanakrishnan, Ramya. "Top 10 Threat Modeling Tools in 2021." *Spiceworks*, 7 Dec. 2021, [www.spiceworks.com/it-security/vulnerability-management/articles/top-threat-modeling-tools/](http://www.spiceworks.com/it-security/vulnerability-management/articles/top-threat-modeling-tools/). Accessed October 14, 2022.
- "What Is Threat Modeling?" *Cisco*, [www.cisco.com/c/en/us/products/security/what-is-threat-modeling.html#~methods-and-tools](http://www.cisco.com/c/en/us/products/security/what-is-threat-modeling.html#~methods-and-tools). Accessed October 16, 2022

## VERIFICATION CROSS REFERENCE MATRIX

Requirement	Verification Method	Success Criteria
Program is easy to use without training.	Sponsor check-in.	Sponsor approves design.
Program does not lose data.	Unit testing, integration testing, and system testing.	All tests pass and exceptions are handled without data loss.
Version control.	Each iteration saves with a unique name.	Old version can be loaded.
Assistance functions.	All tools have a tool tip icon.	Clicking the tool tip gives a valid description.
Import functionality	Tool can load a model from a file.	The model loads correctly without errors.
Export functionality	The tool can create a PDF of the model.	All model content is correctly displayed in the PDF.

*Table 6: Verification Cross Reference Matrix*

## PROJECT PLAN

Determine Requirements	Design Initial Iteration	Threat Modeling Training	Modify Initial Iteration Design	Finalize Design	Sponsor Check In	Initial UI Implementation	UI Demonstration
Weeks 1-5	Weeks 5-7	Week 8	Weeks 9-11	Weeks 11-13		Weeks 13-15	Week 16

*Figure 18: Fall 2022 Project Plan*

Integrate Backend	Write Tests	Sponsor Check In	Integration Testing	Sponsor Check In	Systems Testing	Project Refinement
Weeks 17-20						
	Weeks 20-22					
			Weeks 23-25			
					Weeks 26-29	
						Weeks 29-32

Figure 19: Spring 2023 Project Plan

Our project plan does not have specific sub-tasks for each member. All users will be expected to be cross-functional, handling tasks assigned in Jira and being self-sufficient in researching solutions to each issue. Each member of the team will be responsible for writing component test cases for their code and integration tests for component interactions. Additionally, tasks will be added to Jira weekly, and they will be dependent upon the tasks completed the previous week and decisions on how to proceed with development. This is the nature of the Agile process.

Each week, there will be a stand-up, where team members will demonstrate the work completed in the previous week and address challenges that prevented them from completing an assigned task. Teamwork is encouraged, and any description of an incomplete task should include the steps taken and team members engaged to attempt to complete it on time. If a task is deemed to be too great for any of the team members to complete, an alternative approach must be implemented, which will require a change in the design of subsequent tasks. It may also require completed tasks to be refactored to adapt to the new solution.

Much of the “Finalize Design” task in weeks 11-13 will be development of skills needed to complete the front end, where the team has the most knowledge to gain. The goal is to have an initial iteration of the project’s user demonstration completed by the end of the Fall Semester. This will include a basic version of the User Interface for the purpose of demonstration due on the last week of classes. Winter break will be utilized to Integrate the Backend which must be reviewed and Unit Tested right after. middle of the Spring semester. This plan allows half of a semester to add new features or refactor code to make it more readable and extendable. The group plans to get a lot of the code written over Christmas break to minimize the impact on other classes, but it is not realistic to get a working version of a piece of software this complex completed, even as minimum viable product or prototype, within a single semester. This will be followed by Integration Testing

and System Testing with multiple sponsor check-ins. The final part is for all polishing and refactoring of the project working close with the sponsor.