

Fall 2022

## Supporting the Discovery, Reuse, and Validation of Cybersecurity Requirements at the Early Stages of the Software Development Lifecycle

Jessica Antonia Steinmann  
Embry-Riddle Aeronautical University, [steinmaj@erau.edu](mailto:steinmaj@erau.edu)

Follow this and additional works at: <https://commons.erau.edu/edt>



Part of the [Artificial Intelligence and Robotics Commons](#), [Information Security Commons](#), [Software Engineering Commons](#), and the [Systems Engineering Commons](#)

---

### Scholarly Commons Citation

Steinmann, Jessica Antonia, "Supporting the Discovery, Reuse, and Validation of Cybersecurity Requirements at the Early Stages of the Software Development Lifecycle" (2022). *Doctoral Dissertations and Master's Theses*. 716.

<https://commons.erau.edu/edt/716>

This Dissertation - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in Doctoral Dissertations and Master's Theses by an authorized administrator of Scholarly Commons. For more information, please contact [commons@erau.edu](mailto:commons@erau.edu).

# Supporting the Discovery, Reuse, and Validation of Cybersecurity Requirements at the Early Stages of the Software Development Lifecycle

Jessica Steinmann

Fall 2022

Embry-Riddle Aeronautical University

Daytona Beach, Florida

This page is intentionally left blank.

## Acknowledgements

I would like to thank my advisor Dr. Omar Ochoa for all your feedback, endless revisions, guidance and encouragement. I would like to thank my committee Dr. Salamah Salamah, Dr. Massood Towhidnejad, Dr. Laxima Niure Kandel, and Dr. Kenji Yoshigoe for all of your feedback and improvements. I would like to thank the EECS department and recognize Dr. Radu Babiceanu, Dr. Remzi Seker for their leadership. I could not have fulfilled this dream without Professor Farahzad Behi, thank you for all the opportunities throughout the years. I would like to thank the research group for all your help, guidance, and feedback especially Khushboo Dhala, Yevgeniy Lischuk, Timothy Elvira, Tyler Procko, Sarah Reynolds, Lynn Vonder Haar, Juan Ortiz Couder, and Priscilla Carbo. I would also like to thank the students of SE/SYS 530 for participating in the study. I would like to express my deepest gratitude to Dr. Iteris Demirkiran, my McNair advisor, Dr. Nancy Lawrence, and Dr. Bereket Berhane for all of your advice, support, and encouragement through the years. Dr. Ashley Lear, I'm getting my shirt. Marian Yeneic who has supported me since the early days of undergrad, thank you.

Words cannot express my gratitude to Jennifer Heaton for all your encouragement and reminders. I would also like to thank Srikanth Venkataraman for reminding me this is a small period of time in my life.

Finally, I would like to acknowledge and thank my mother, Mariemina Anglade Steinmann if it weren't for all of your sacrifices, support, and encouragements none of this could have been possible. You have taught me that starting over is possible regardless of circumstances time and time again. It is because of all your support and bravery that I was in lucky enough to get the opportunities to not only follow my dreams but to achieve them. There is still much more to accomplish. This is only the beginning. Thank you!

## Dedication

For Fabien Steinmann and Hubert Steinmann, you are both missed dearly.

For Mariemina Anglade Steinmann, Thank you!

## Abstract

The focus of this research is to develop an approach that enhances the elicitation and specification of reusable cybersecurity requirements. Cybersecurity has become a global concern as cyber-attacks are projected to cost damages totaling more than \$10.5 trillion dollars by 2025. Cybersecurity requirements are more challenging to elicit than other requirements because they are nonfunctional requirements that requires cybersecurity expertise and knowledge of the proposed system. The goal of this research is to generate cybersecurity requirements based on knowledge acquired from requirements elicitation and analysis activities, to provide cybersecurity specifications without requiring the specialized knowledge of a cybersecurity expert, and to generate reusable cybersecurity requirements. The proposed approach can be an effective way to implement cybersecurity requirements at the earliest stages of the system development life cycle because the approach facilitates the identification of cybersecurity requirements throughout the requirements gathering stage. This is accomplished through the development of the Secure Development Ontology that maps cybersecurity features and the functional features descriptions in order to train a classification machine-learning model to return the suggested security requirements. The SD-SRE requirements engineering portal was created to support the application of this research by providing a platform to submit use case scenarios and requirements and suggest security requirements for the given system. The efficacy of this approach was tested with students in a graduate requirements engineering course. The students were presented with a system description and tasked with creating use case scenarios using the SD-SRE portal. The entered models were automatically analyzed by the SD-SRE system to suggest the security requirements. The results showed that the approach can be an effective approach to assist in the identification of security requirements.



# Table of Contents

Acknowledgements.....	i
Abstract.....	iii
Table of Abbreviations .....	x
Table of Equations .....	xiii
Table of Figures .....	xiv
Table of Tables .....	xvii
1 Introduction.....	1
1.1 The Need for Secure Software Systems.....	1
1.1.1 The Cost of Cybercrime.....	2
1.2 The Difficulties of Building Secure Software Systems .....	7
1.2.1 Why is it Hard to Build Software Systems? .....	8
1.2.2 The Challenges of Eliciting Cybersecurity Requirements.....	11
1.3 Motivation .....	12
1.4 Research Questions .....	14
1.4.1 How can the elicitation and analysis of functional features be leveraged to assist with the specification of cybersecurity requirements?.....	14
1.4.2 How can the use of existing best practices of cybersecurity be leveraged to assist in the identification of cybersecurity requirements?.....	15
2 Background.....	16
2.1 Software Development Life Cycle.....	16



2.1.1	Writing Requirements .....	18
2.1.2	Requirements Gathering .....	21
2.1.3	Use Case models .....	22
2.2	Security Requirement Elicitation Techniques .....	24
2.2.1	Threat Modeling.....	25
2.2.2	Abuse Cases .....	26
2.2.3	Misuse Cases.....	26
2.2.4	Attack Surface Analysis.....	28
2.2.5	Standards for Security Implementation .....	28
2.2.6	Security Requirement Frameworks.....	29
2.3	Semantic Web Technologies .....	30
2.3.1	What are Semantic Web Technologies? .....	30
2.3.2	The Challenges of Semantic Technology .....	32
2.3.3	Why Use Semantic Web Technologies?.....	33
2.3.4	With the improvements in other parts of Artificial Intelligence and Machine Learning is Semantic Web still relevant? .....	33
2.3.5	What is an Ontology?.....	34
2.3.6	Use of Semantic Technologies in Gather Requirements .....	39
2.3.7	The Use of Semantic Technologies in the Cybersecurity Domain .....	40
2.4	Machine Learning .....	40
2.4.1	Natural Language Processing .....	45
2.4.2	How to Know which ML model to Use? .....	48
2.4.3	How to evaluate Machine Learning Models? .....	48

3	Approach.....	54
3.1	The Goals of this Research.....	54
3.2	The Approach.....	54
3.2.1	Scenario Driven Security Requirements Elicitation (SD-SRE).....	54
3.2.2	The SD-SRE Process .....	55
3.2.3	The Information Gathering Phase.....	56
3.2.4	Developing the Ontology .....	57
3.2.5	Determining the ML Model .....	59
3.2.6	The Aggregator .....	60
3.2.7	Validating the SD-SRE.....	62
3.3	Rationale for Technical Decisions .....	62
3.4	How Will the SD-SRE Approach be Used?.....	63
4	Scenario Driven Security Requirements Elicitation (SD-SRE).....	64
4.1	Gathering Information.....	64
4.1.1	Use Case Scenarios .....	65
4.1.2	Security Concepts and Requirements .....	65
4.2	Secure Development Ontology .....	66
4.2.1	Security Concepts Competency Questions .....	68
4.2.2	Use Case Scenarios Parsing.....	72
4.3	Machine Learning Model.....	74
4.3.1	Login Requirements.....	76
4.3.2	Sensitive Information Requirements.....	78

4.3.3	Authentication Requirements.....	80
4.3.4	Authorization Requirements .....	82
4.3.5	Email Collection Requirements .....	84
4.3.6	Input Validation Requirements.....	86
4.3.7	Encryption Requirements.....	88
4.3.8	Random Number Generator Requirements.....	90
4.3.9	Database Requirements.....	92
4.3.10	File Upload Requirements .....	94
4.3.11	Logging Requirements.....	96
4.3.12	Weakness .....	98
4.4	Validation.....	99
4.4.1	Student Educational Backgrounds .....	101
4.4.2	Student Results.....	102
4.4.3	Results of the Suggested Requirements .....	104
4.4.4	Comparison between Using the Portal or Not Using the Portal .....	114
4.4.5	Comparison Between Recommended Security Requirements and Student Security Requirements .....	114
4.5	Research Questions Results .....	115
4.5.1	How can the elicitation and analysis of functional features be leveraged to assist with the specification of cybersecurity requirements?.....	115
4.5.2	How can the use of existing best practices of cybersecurity be leveraged to assist in the identification of cybersecurity requirements?.....	116
5	Related Work .....	118

5.1	Development of Cybersecurity Ontologies .....	118
5.2	Applications of Cybersecurity Ontologies .....	120
5.3	The Use of Machine Learning in Requirement Elicitation .....	121
5.4	The Use of Machine Learning in Cybersecurity Requirement Elicitation.....	123
5.5	The Use of Ontologies with Machine Learning .....	124
5.6	The Use of Ontologies with Machine Learning in Cybersecurity .....	124
5.7	Comparison to Approach .....	125
6	Results and Conclusion.....	127
6.1	SD-SRE Results .....	127
6.2	Results of Best Practices .....	128
6.3	The Benefits of the SD-SRE .....	130
6.4	Future Work .....	132
7	References.....	134
8	Appendix.....	145
8.1	The SD-SRE Portal .....	145
8.2	All Suggested Security Related Requirements.....	156
9	Glossary .....	164
10	Index .....	165

## Table of Abbreviations

Abbreviation	Definition
AHP	Analytic Hierarchy Process
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interfaced
AUC	Area Under the Curve
BPMN	Business Process Modeling Notation
BFO	Basic Formal Ontology
CLASP	Comprehensive Lightweight Application Security Process
CIA	Confidentiality Integrity Availability
CISA	Cybersecurity & Infrastructure Security Agency
CNN	Convolution Neural Networks
CWE	Common weakness Enumeration
DQN	Deep Q Network
DevSecOps	Development Security Operations
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
DOS	Denial of Service
DDOS	Distributed Denial of Service
FBI	Federal Bureau of Investigation
FN	False Negative
FP	False Positive
GAN	Generative Adversarial Networks
Glove	Global Vectors
IC3	Internet Crime Complaint Center
IDE	Integrated Development Environments
IDEF	Integrated Definition for Function Modeling
IDEF5	Integrated Definition for Ontology Description Capture Method
IEEE	Institute of Electrical and Electronics Engineers
IOT	Internet of Things
IT	Information Technology
IP	Intellectual Property

IRI	Internationalized Resource Identifier
LSTM	Long Short-term Memory Networks
ML	Machine Learning
MLP	Multilayer Perceptron
MODDALS	Methodology for Designing Layered Ontology Structures
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
OSCO	Ontologies of Secure Cyber Operations
OWASP	Open Web Application Security Project
OWL	Web Ontology Language
PASTA	Process for Attack Simulation and Threat Analysis
PBAC	Permission Based Access Control
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RIF	Rule Interchange Format
RNN	Recurrent Neural Networks
ROC Curve	Receiver-Operator Curve
SARSA	State Action Reward State Action
SD-SRE	Scenario Driven Security Requirements Elicitation
SDO	Secure Development Ontology
SKOS	Simple Knowledge Organization System
SML	Supervised Machine Learning
SPARQL	SPARQL Protocol and RDF Query Language
SRS	Software Requirement Specifications
STEM	Science Technology Engineering Math
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, elevation of Privilege
STUCCO	Situation and Threat Understanding by Correlating Contextual Observations
SUMO	Suggested Upper Merged Ontology
SWEBOK	Software Engineering Body of Knowledge
SWRL	Semantic Web Rule Language
TF-IDF	Term Frequency-Inverse Document Frequency
TN	True Negative
TP	True Positive

UCO	Unified Cybersecurity Ontology
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniformed Resource Locators
URN	Uniformed Resource Name
USML	Unsupervised Machine Learning
WWW	World Wide Web
XML	Extensible Markup Language

## Table of Equations

Equation 1: Term Frequency .....	46
Equation 2: Document Frequency .....	46
Equation 3: Reciprocal of Document Frequency .....	47
Equation 4: TF-IDF.....	47
Equation 5: Accuracy Metric .....	50
Equation 6: Precision Metric .....	50
Equation 7: Recall Metric .....	50
Equation 8: F1 Score Metric .....	51
Equation 9: Specificity Metric .....	51
Equation 10: Fall-Out Metric.....	52
Equation 11: Miss Rate Metric .....	52



## Table of Figures

Figure 1: The software development life cycle.....	16
Figure 2: Non-functional requirements classification.....	20
Figure 3: Example use case model.....	23
Figure 4: Example scenarios for the use case model shown in Figure 3 above.....	24
Figure 5: Example misuse case.....	27
Figure 6: Semantic Web Stack .....	30
Figure 7: Confusion Matrix .....	49
Figure 8: Example ROC curve.....	53
Figure 9: The SD-SRE process.....	55
Figure 10: SDO snapshot of permission_based_access_control_(PBAC) .....	58
Figure 11: SDO snapshot of password_spraying vulnerability .....	58
Figure 12: The ontology development process.....	59
Figure 13: Data flow diagram.....	60
Figure 14: Requirements aggregator.....	61
Figure 15: Information gathering for SD-SRE .....	64
Figure 16: Ontology use in the SD-SRE.....	67
Figure 17: SDO object properties .....	69
Figure 18: Access_control described_by .....	69
Figure 19: Access_control Class in the SDO.....	70
Figure 20: Multifactor_authentication mitigation example for access control.....	71
Figure 21: Machine leaning model in the SD-SRE .....	76
Figure 22: Login requirements model confusion matrix .....	76

Figure 23: Login requirements model ROC and AUC curve .....	77
Figure 24: Sensitive information requirements model confusion matrix .....	78
Figure 25: Sensitive information requirements model ROC and AUC Curve .....	79
Figure 26: Authentication requirements model confusion matrix .....	80
Figure 27: Authentication requirements model ROC and AUC Curve .....	81
Figure 28: Authorization requirements model confusion matrix.....	82
Figure 29: Authorization requirements model ROC and AUC Curve.....	83
Figure 30: Email collection requirements model confusion matrix.....	84
Figure 31: Email collection requirements model ROC and AUC Curve.....	85
Figure 32: Input validation requirements model confusion matrix .....	86
Figure 33: Input validation requirements model ROC and AUC Curve .....	87
Figure 34: Encryption requirements model confusion matrix .....	88
Figure 35: Encryption requirements model ROC and AUC Curve .....	89
Figure 36: Random number requirements model confusion matrix .....	90
Figure 37: Random number requirements model ROC and AUC Curve .....	91
Figure 38: Database requirements model confusion matrix .....	92
Figure 39: Database requirements model ROC and AUC Curve .....	93
Figure 40: File upload requirements model confusion matrix.....	94
Figure 41: File upload requirements model ROC and AUC Curve.....	95
Figure 42: Logging requirements model confusion matrix .....	96
Figure 43: Logging requirements model ROC and AUC Curve .....	97
Figure 44: Portal interface of the SD-SRE for validation.....	99
Figure 45: DirectCoins app.....	100

Figure 46: SD-SRE Portal welcome page.....	145
Figure 47: SD-SRE Portal add projects ability .....	146
Figure 48: SD-SRE Portal blank project.....	147
Figure 49: SD-SRE Portal add actors' ability.....	148
Figure 50: SD-SRE Portal add scenario ability .....	149
Figure 51: SD-SRE Portal add scenario steps ability .....	150
Figure 52: SD-SRE Portal add requirement/specification ability.....	151
Figure 53: SD-SRE Portal add nonfunctional category requirement/specification ability.....	152
Figure 54: SD-SRE Portal example of instructional project.....	154
Figure 55: SD-SRE Portal example of use case scenario .....	155

## Table of Tables

Table 1: The cost and number of days to identify and contain a data breach by initial attack vector per IBM cost of a data breach report .....	3
Table 2: Example sparse matrix for text vectorization .....	46
Table 3: Train test split for model testing .....	60
Table 4: Use case scenario resources .....	65
Table 5: Security concept and requirements resources .....	65
Table 6: Security requirements .....	66
Table 7: Access_control class of the SDO .....	71
Table 8: Parsed used case scenarios classifications .....	72
Table 9: 4 security requirements for creating a username .....	73
Table 10: Login requirements model performance .....	77
Table 11: Sensitive information requirements model performance .....	79
Table 12: Authentication requirements model performance .....	81
Table 13: Authorization requirements model performance .....	83
Table 14: Email collection requirements model performance .....	85
Table 15: Input validation requirements model performance .....	87
Table 16: Encryption requirements model performance .....	89
Table 17: Random number requirements model performance .....	91
Table 18: Database requirements model performance .....	93
Table 19: File upload requirements model performance .....	95
Table 20: Logging requirements model performance .....	97
Table 21: Volunteer students educational background of portal users .....	101

Table 22: Volunteer students’ educational background of non-portal users .....	102
Table 23: Portal students submissions .....	103
Table 24: Student submissions not assigned the portal .....	103
Table 25: All student submission- the full validation set .....	104
Table 26: Validation results for login requirements .....	105
Table 27: Portal1 Login related Use Cases.....	106
Table 28: Example of Portal3 login Requirement Analysis .....	106
Table 29: Validation results for sensitive information requirements.....	107
Table 30: Validation results for authentication requirements.....	108
Table 31: Validation results for authorization requirements .....	108
Table 32: Validation results for email collection requirements.....	109
Table 33: Validation results for input validation requirements .....	110
Table 34: Validation results for encryption requirements .....	111
Table 35: Validation results for database requirements.....	112
Table 36: Validation results for file upload requirements .....	112
Table 37: Validation results for logging requirements .....	113
Table 38: Number of projects that return a requirements grouping.....	113
Table 39: Comparison of number of requirements student suggested vs the SD-SRE .....	115
Table 40: SD-SRE validation results .....	127
Table 41: All currently suggestable security related requirements.....	163

# 1 Introduction

This dissertation investigates supporting the discovery, reuse, and validation of cybersecurity requirements at the early stages of the software development lifecycle. The approach leverages the use of the popular requirements gathering techniques, use case models and scenarios, the capabilities of domain mapping/discovery technologies; semantic technologies and machine learning for the identification of security requirements. The proposed approach provides a rapid discovery and correction of software vulnerabilities in the development process at the earliest stages of the development life cycle. In addition, the approach also provides a way to reuse cybersecurity requirements.

This chapter introduces the need for secure software systems, the difficulties of building secure software systems, the challenges of eliciting cybersecurity requirements, the motivation for this research, and the research questions.

## 1.1 The Need for Secure Software Systems

As our reliance on software systems continues to increase, so does the frequency and value of cyberattacks, making them more devastating than ever before. The frequency of cyberattacks, and the number of people affected, are increasing every year [1], [2], [3], [4]. The annual increasing cost of cybercrime is estimated to hit \$10.5 trillion globally by 2025 [5]. Due to the cost and risk associated with cyberattacks and the continuing growth in dependency on software systems one of the gravest issues facing society is the ability to improve cybersecurity capabilities of software systems.

### 1.1.1 The Cost of Cybercrime

The Federal Bureau of Investigation (FBI) Internet Crime Complaint Center (IC3) collects and analyzes cybercrime complaints. Every year they release a summary of the past year's complaints. The latest report is the Internet Crime Report of 2021. In the past 5 years the center has received 2.76 million complaints with a reported loss of \$18.7 billion. The center received 847,376 reports in 2021 up from 791,790 in 2020 with a reported loss of \$6.9 billion in 2021 up from \$4.2 billion in 2020. That is a 7.02% increase in complaints and a 64.28% increase in losses. Cybercrimes such as identity theft, personal data breach, and phishing/vishing/smishing/pharming are three out of top five crime types identified in the report. The report identifies how many complaints are made by 16 infrastructure sectors (defense, energy, food and agriculture, transportation, etc.). The top 3 infrastructure sectors affected are healthcare and public health, financial services, and information technology. The rise in a social engineering technique called tech support fraud<sup>1</sup> has more than quadrupled in losses over the last 5 years from \$14.8 million in 2017 to \$347.6 million in 2021. The IC3 on average receives more than 2,300 complaints daily for an average of 552,000 number of complaints received per year over the last 5 years. The report summarized a total of \$6.9 billion dollars reported to have been lost in 2021 to cybercrimes [3].

The IBM 2022 Cost of a Data Breach Report studied 550 organizations and found that 83% of the organizations they studied had more than one data breach with 45 % of breaches being cloud based. The average cost of a data breach is \$4.35 million up from \$4.24 million in 2021 but for a critical infrastructure data breach average cost are even higher at an average of \$4.83 million. Table 1 below shows the cost of a data breach in million and the total average time to identify and

---

<sup>1</sup> Tech support fraud is a type of fraud where scammer pose as legitimate technical support for various companies in order to gain access or information from victims.

contain a data breach by initial attack vector. According to the report it cost on average \$4.9 million when phishing is the initial attack vector, \$3.94 million when a lost device is the initial attack vector, \$4.10 million when social engineering is the initial attack vector, \$4.14 million when cloud misconfiguration is the initial attack vector, \$4.18 million when malicious insider is the initial attack vector, and \$4.55 million when vulnerability in third party software is the initial attack vector.

Initial Attack Vector	Average Cost in Millions	Average Time to Identify in Days	Average Time to Contain in Days	Total Average Time to Identify and Contain in Days
System Error	\$3.82			
Accidental data loss or lost device	\$3.94			
Physical security compromise	\$3.96	217	63	280
Social engineering	\$4.10	201	69	270
Cloud misconfiguration	\$4.14	183	61	244
Malicious insider	\$4.18	216	68	284
Stolen or compromised credentials	\$4.50	243	84	327
Vulnerability in third-party software	\$4.55	214	70	284
Business email compromise	\$4.89	234	74	308
Phishing	\$4.91	219	76	295

*Table 1: The cost and number of days to identify and contain a data breach by initial attack vector per IBM cost of a data breach report [6]*

The study also concludes that on average the cost of a single record in a data breach is \$164. These cost lead to 60% of organizations impacted by a data breach to raise their prices. Key factors discovered in the study were that data breach cost increased due to security skills shortage on average \$206 thousand, lost or stolen devices on average \$227 thousand, and security system complexity on average \$290 thousand.

Organizations on average took 277 days to identify and contain a data breach this is down from 287 days in 2021 but the overall lifecycle of a data breach on average takes 304 days to detect



and respond. The study found 43% of organizations that were in early stages of development did not apply security practices to safeguard their cloud environments and 59% of the organizations didn't deploy the zero trust<sup>2</sup> framework. The average cost saving with a mature zero trust deployment vs early adoption of zero trust is \$1.51 million. The average cost savings with fully deployed security AI and automation is \$3.05 million. The total number of days to identify and contain a data breach by a fully deployed AI and automation security system is 249 days compared to 299 with partial deployment and 323 days without any deployment. The deployment of artificial intelligence platforms decreases data breach cost by \$300 thousand, Development Security and Operations (DevSecOps) approaches by \$276 thousand, extensive use of encryption by \$252 thousand, participating in threat sharing \$237 thousand, identity and access management by \$224 thousand, multifactor authentication by \$186 thousand [6].

The McAfee foundation distributed a report about the hidden costs of Cybercrime in 2020 in which they found the cost of cybercrime globally increased 50% in 2 years from \$600 billion in 2018 to \$900 billion in 2020. The reports highlights cost other than cash such as opportunity cost, system downtime, reduced efficiency, brand damage and loss of trust, Intellectual Property (IP) theft, incident response cost, outside assistance, cyber risk insurance, and damage to employee morale [4].

The 2019 Ninth Annual Cost of Cybercrime Study by Accenture Security highlights information theft as the most expensive and fastest rising consequence of cybercrime. The number of data breaches in the study increased from 130 to 145 which is an 11% increase with an overall 67% percent increase over the previous 5 years. Their estimated annual cost in cybercrime

---

<sup>2</sup> Zero Trust is a security framework/architecture NIST 800-207 that assumes no implicit trust is granted to assets users must be authenticated, authorized, and regularly validated for access given [84].

increased from \$11.7 million in 2017 to \$13 million in 2018 which is a 12% increase in cost with a 72% increase over the 5 previous years. Malware attacks were the most expensive attack type for an organization with malicious insider having the highest increase in frequency and cost. Malware, web-based attacks, and denial of service attacks were the most prominent in 2018 [2].

The Accenture State of Cybersecurity Resilience 2021 report surveyed 4,744 executives with 4,244 being security professionals from 18 countries and 23 industries. The report found that attacks are up 31% from 2020 to 2021. Only 50% of companies using cloud service were discussing or considering security, 18% had limited discussion about cloud security, while 32% has not considered it [7].

Risk IQ's 2021 Evil Internet Minute Report estimates that the average cost of a breach is \$7.2 per minute with the lost to cybercrime costing \$1.79 million a minute. The total estimated threats are 648 per minute while the estimated number of compromised records per minute is 525,600 [8].

Over the past century there have been news reports on data breaches from some the most recognizable companies. Yahoo had the biggest data breach in 2013 affecting 3 billion accounts that allowed hackers access to security questions and answers, and plain text passwords. Alibaba had a data breach in 2019 where another shopping website targeted them and scrapped the information of 1 billion pieces of user data. In 2019, First American Corporation announced that hackers were able to bypass their authorization process on their website and obtained 885 million mortgage documents [9]. In June of 2021, LinkedIn was also the target of someone scrapping 700 million user's profile information through their API and selling it on the dark web. In April of 2019, 533 million Facebook users had their Facebook data exposed to the internet. Experian experienced a data breach in 2013 where it exposed 200 million records of personal data such as

credit card and social security numbers when they fell victim to a social engineering attack. In October 2013, Adobe had 153 million user records which included customer credit card records and login data stolen by hackers [10].

On January 17, 2022, 500 cryptocurrency wallets were compromised on crypto.com because attackers were able to bypass the two-factor authentication. On March 20<sup>th</sup>, 2022 Microsoft was targeted by a hacker group called Lapsus\$. The hacker group retrieved some information from Microsoft by compromising Cortana and Bing. No customer data was compromised, and Microsoft was recognized for their quick response. In April 2022, A former Cash App employee targeted Cash App and compromised user records. The employee was only able to steal a limited amount of identifiable data, but Cash App had to inform more than 8 million users about the incident [11].

Data breaches are the most notorious, well documented, and often easy to equate to dollars lost types of attacks by hacker but there are many more different types of attacks to be concerned about such as remote start apps exposing thousands of cars to hackers [12]. Data and money are not always the motive for attacks. In 2015, security researcher Charlie Miller demonstrated the ability to remotely shut down a Jeep going 70 mph. In 2016, the democratic national committee had their emails hacked and leaked possibly affecting the outcome of the election [13]. In October 2022, a Russian hacker group by the name Killnet targeted 14 airport websites in the US through a Distributed Denial Of Service (DDOS) attack [14]. Another famous hacker group, Anonymous, is famous for their hacktivist attacks by constantly conducting data breaches and denial of service attacks such as: Project Chanology in 2008, Project Skynet in 2009, Operation Didgeridie in 2009, Operation Titstorm in 2010, Operation Payback in 2010, Operation WikiLeaks in 2010, Arab Spring in 2011, HBGary Federal in 2011, Geohot in 2011, Operation Egypt in 2011, Occupy Wall

Street in 2011, Operation Syria in 2011, Operation Darknet in 2011, Operation Russia in 2012, CIA Attack in 2012, and more recently targeting QAnon [15], [16], [17].

## 1.2 The Difficulties of Building Secure Software Systems

The need for secure software systems is growing but cybersecurity features are often addressed late in the software development life cycle, e.g., released in later versions when the system is already in use. These features are often only addressed once a vulnerability has been found and exploited. Eliciting and specifying the requirements, the initial stage of the software development life cycle, identifies the behaviors that a system must exhibit. This is a critically important step in the creation and implementation of software systems. Building secure software systems requires a greater need for secure software engineering approaches that can detect and address security vulnerabilities.

The development of systems is often a difficult task due to the size of systems and lack of clear customer expectations. Cybersecurity requirements are even more difficult to elicit as customers are not aware of the threats their systems may face and many developers lack the necessary skills to properly elicit the security requirements on their own. Developers mainly focus on eliciting functional requirements which are the expected behaviors of a system, but cybersecurity requirements are nonfunctional requirements which are the expected qualities of the system. Developers often work with various stakeholder to develop functional requirements. These stakeholders are often users of the system or have domain knowledge to support the functional requirements. Security requirements require domain knowledge to gather, are often not a behavior of the system. Developers often fail to partner with security experts to develop security requirements and as such delayed security features to later phases of the development process.

Thus, there is a need to effectively elicit the correct cybersecurity requirement at the initial stages of the development process.

### 1.2.1 Why is it Hard to Build Software Systems?

Software systems are difficult to build for several reasons: the software industry is relatively young, there are no barriers to becoming a programmer, there can often be a lack of user input, customers often do not know what they want until they see it, estimating is seen as art and not a science, every line of code is a potential failure point with most systems having hundreds of thousands even millions of lines of code, there are limits to the amount of testing that can realistically be carried out, and systems are impacted by external factors.

#### *1.2.1.1 Software Industry is Relatively Young*

Software engineering had been around since the late 1940s but emerged as a profession in the 1980s. The timespan between 1965-1985 is considered the software crisis because developers struggled to deliver projects on time, on budget, and that met the customers' expectations. Developers and researchers worked on developing methods and tools to address the problems of the software crisis. The internet helped speed up the need for more formalized approach to software development during the 1990s. The methodologies developed during that time provided some relief but overburdened developers with processes and still led to projects begin late, over budget, and not what the customer expected. In the early 2000s lightweight methodologies were developed to assist with connecting the customer with prototypes throughout the whole process [18].

#### *1.2.1.2 General Education of Programmers*

There is a lack of barriers to becoming a programmer. For the past decade there has been a myth that there is a shortage of programmers which has led to many schools providing 3-month boot

camp programmer development programs. With the vast array of resources available on the internet anyone can teach themselves how to program. The main barrier is having a computer and access to the internet, as many development environments are free. There is also confusion around the difference between a programmer and a software engineer. A programmer is similar to a line worker while a software engineer designs and plans the work to be done on the line. In 2018, many have come to conclude that there is not a lack of programmers or software engineers but there is a lack of good programmers and software engineers [19]. In the past couple of years, the number of computer science graduates has not had extreme growth in the US causing a shortage in skilled workers with the basic knowledge of software engineering. Even with a college degree, a person may not be qualified. A college degree often provides the basic knowledge to be a software engineer but there is a need for mentorship and experience developed thru hands on projects [20]. Many companies have become desperate for good programmers and software engineers, e.g., Google and Apple, that they have drop the requirement for college degrees and instead are relying on experience [19]. The need for skilled software engineers and programmers is not going to decrease as our reliance on software systems continues to increase [20].

### *1.2.1.3 The Nature of Software System Development*

Karl Wiegers and Joy Beatty, in their 3rd edition of Software Requirements, state “Despite decades of industry experience, many software organizations struggle to understand, document, and manage their product requirements. Inadequate user input, incomplete requirements, changing requirements, and misunderstood business objectives are major reasons why so many information technology projects are less than successful... Customer’s often do not have the time or patience to participate in requirements activities. In many cases, project participants even disagree on what a “requirement” is [21].”

The first stage of the software development process is requirements elicitation, which results in a requirements specification document, and several other artifacts that can be used to assist in the understanding of the specification, e.g., use case models, scenarios, class diagrams, and data flow diagrams. The requirements specification document represents the contract between developers and customers. These requirements are to be traced throughout the development process to guarantee their implementation. Requirements gathering is difficult as customers may not be the system users, customers may not have the time and patience to deal with the requirements process, and customers often do not know what they want and do not want until they see it. It is difficult for humans to communicate ideas that provide a common vision due to differences in experiences; this is a challenge that carries over into requirements gathering activities. It takes an experienced developer to ask the right questions to elicit the proper requirements. The next two stages are design and development. This is where most software defects and bugs are introduced. The design stage lays out the plan for development while the development stage is the implementation of the design plan. This requires a common language to communicate amongst all developers and coordination to integrate the different parts of the system.

The last three stages of the software development life cycle are testing, implementation, and maintenance. Testing is challenging as 100% test coverage is difficult to achieve due to time, resources, and the combination of system executions. Many techniques have been developed to try to mitigate the inability to perform 100% test coverage.

Testers and customers will use the requirements specification to verify that the system meets the specifications, this is referred to as the verification process. Testers also use the specifications to validate the system, which determines if they built what the specifications

required. System testing also faces other challenges such as the user will execute code that has never been tested, the order in which statements are tested are executed differently than how they will actually execute in the production environment, the user will likely use a combination of untested inputs, and the users operating environment is never tested [22]. Implementation is turning the system over to the customers for users to use. Maintenance is the upkeep of the system. The implementation and maintenance stages often identify new and late requirements.

Due to the complexity of developing software systems, it is challenging and expensive to add requirements late in the development life cycle. Analysis would need to be conducted to determine which requirements the new requirement would potentially impact, the parts of design impacted, which part of the system the new requirement would be implemented in, and the implementation of the new requirement could potentially require bringing the system offline.

Researchers are constantly attempting to improve the development process of systems. Much progress has been made but there is still quite bit of room for improvement. John Dooley, author of *Software Development and Professional Practice*, summed it up perfectly stating "Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any—no inventions that will do for software productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did for computer hardware [23]."

### 1.2.2 The Challenges of Eliciting Cybersecurity Requirements

Most requirement gathering activities focus on the elicitation and analysis of functional requirements. Functional requirements are requirements that describe the behavior of the system. Cybersecurity requirements are nonfunctional requirements, they are a quality of the system. The source of many cybersecurity requirements are customers, expressing them as non-functional requirements, that is desired qualities of the system. However, the implementation of these must



be done through specific functional requirements, and therein lies the challenge. There has been previous work that expand on use case modeling to discover cybersecurity requirements: abuse cases and misuse cases [24]. Abuse cases identify the ways the system could purposefully be threatened while misuse cases identify unintended actions the system may accidentally allow users to commit. Both result in a loss or damage to the system or actors. For example, imagine a system with a coupon code that is intended to be used once. If the system allows the same user to reuse that coupon code multiple time, this would be an example of misuse. If a user creates multiple accounts in order to reuse the code multiple times this would be considered an abuse case. In both cases, the company loses more money than intended. When developers elicit requirements, they consult with several stakeholders. The problem with these approaches is that they do not require the consultation of cybersecurity experts, instead the approaches are limited to the imagination of the developers. Even if a developer can successfully identify threats against the system this does not mean that they have the necessary knowledge to properly mitigate those threats.

Another challenge with cybersecurity requirements is that they depend on knowledge of the functional requirements. Many of the techniques that exist, such as SQUARE [25], depends on security goals and risk assessment of the system in order to elicit cybersecurity requirements. This adds levels of activities to perform on top of an already complicated processes. This can become a challenge especially as functional requirements change.

### 1.3 Motivation

During the earliest era of development of software systems, the exploitation of these systems was not considered. Once development of software systems began to become more popular, development life cycles and techniques for stages of these life cycles were created to make implementation of these systems more efficient. It is quite difficult to communicate the expected

behaviors of a system and even more difficult to try to estimate how that system may be exploited. Cyberattacks are becoming more technically sophisticated in nature, there is a need to design systems that can withstand malicious attacks [26]. These attacks often do not follow generic trends; specific environments, programming languages, and tools to exploit system vulnerabilities [27]. Due to the unpredictability, formal approaches to counter cyberattacks can be challenging to adopt. Information about weaknesses can be discovered at any time so there is a constant need to monitor and continuously update the system when vulnerabilities are discovered. In addition, there is a lack of addressing security needs of a system throughout the software development life cycle. Security is often pushed off to the maintenance stage. In 2019, WhiteHat security conducted a survey with 108 participants at a developer's event. The survey found that 43 percent of participants prioritized meeting release deadline over security, 70% had not received security certifications, and only half the participants surveyed had a security expert on their team. [28].

Although cyberattacks can be non-generic in nature, they still pose similar threats to software systems thereby an approach integrating the best cybersecurity implementation practices with the knowledge base of functional requirements can be powerful against cyberattacks. An example is a system requirement for users to be able to access specific parts of the system designated only for them. A developer will likely write a requirement specifying a user login and password feature. There are various attacks that attackers can use to retrieve usernames and passwords, e.g., phishing attacks, man in the middle attacks, brute force attacks. Having a strong password may possibly prevent or delay the consequences of brute force attacks but it does not prevent or delay the consequences of phishing or man in the middle attacks. One way to prevent the consequences of those attacks is two-factor authentication but having a requirement for two-factor authentication is not enough. That requirement needs to also have additional accompanying requirements that

explain features that securely implement two-factor authentication, e.g., alerting the account owner or administrator of a failed login attempts.

Semantic technologies are flexible in nature and can assist in data categorization and storage [29]. One of semantic technologies most diverse tools are ontologies, which have been used to create data collections, which are relationship maps of data and their respective categorizations. Ontologies are invaluable in that they not only store data collections, but they also highlight connections between them.

Machine learning has proven that computers can learn to identify patterns in data. The ability to identify patterns makes machine learning a powerful tool in identifying possible security requirements from descriptions of functional features of a system. The problem with machine learning is that it is able to identify patterns but needs a user to discern the meaning of the pattern. This is addressed by supervised machine learning models. Supervised machine learning models requires the data to have labels in order to train the model. The labeling of data is a tedious task if the data is not already pre-labeled. Ontologies can be used to assist in the labeling of data in order to train machine learning models to identify security requirements from descriptions of system features.

## 1.4 Research Questions

The following are the questions that drive the focus of this research.

### 1.4.1 How can the elicitation and analysis of functional features be leveraged to assist with the specification of cybersecurity requirements?

There are two sets of knowledge needed in order to elicit cybersecurity requirements: cybersecurity knowledge and knowledge of functional system requirements. Functional system requirements are

extracted from requirement eliciting activities. Cybersecurity requirements depend on functional requirements because they are extension of the quality of the functional requirement. The knowledge of what a system is supposed to do is stored in the functional requirements. It is difficult to protect something you do not know the details about. For example, in order to properly secure a building, the blueprints are needed to determine where to best place resources. The functional requirements are the blueprints of the system and store the information needed to determine where to best place resources, which in this case are cybersecurity requirements.

#### 1.4.2 How can the use of existing best practices of cybersecurity be leveraged to assist in the identification of cybersecurity requirements?

Semantic web technologies stores, categorizes, and links data together. Cybersecurity requirements require knowledge links between two domains: software engineering, and cybersecurity. Semantic web technologies are flexible enough to model these two growing and constantly evolving domains for inferences and requirements to be extracted. Semantic web technologies are a well-documented structured data representation and can be used to map cybersecurity best practices to common functional feature descriptions. The combination of semantic web technologies and machine learning can be used to leverage the identification of cybersecurity requirements based on best practices. This is discussed further in Chapter 2.

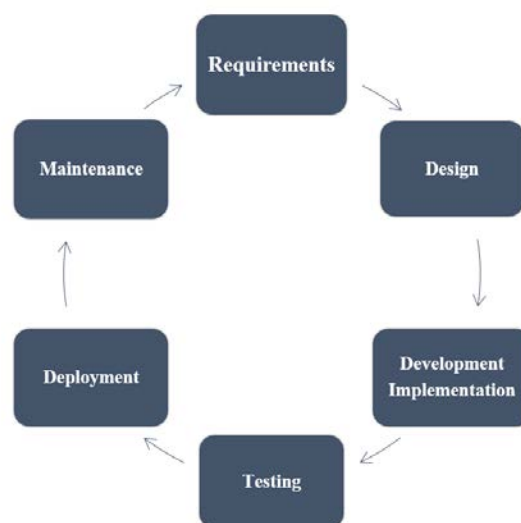
## 2 Background

This chapter discusses an introduction to the software development life cycle with a focus on the requirements gathering phase, a discussion of current security requirements elicitation techniques, an introduction to semantic technologies, and the foundations of machine learning.

### 2.1 Software Development Life Cycle

Software development life cycle models were developed after the software crisis in the late 1960s to the early 1970's. The software crisis was due to software development projects not being delivered on time, full of bugs, which did not meet stakeholder's requirements, and were difficult to maintain. The crisis was attributed to the nature of software systems. Software systems are intangible, easy to change, complex, and construction is human intensive. As a result, software processes were developed to provide guidance on what should be created, when they should be created, and how to evaluate those artifacts that are created.

The software development life cycle is an iterative process that consists of 6 stages: Requirements, Design, Development/Implementation, Testing, Deployment, and Maintenance.



*Figure 1: The software development life cycle*

There are two kinds of software development processes, namely prescriptive and agile. Prescriptive processes are orderly, while agile processes are flexible with quick delivery. Some well-known prescriptive processes are waterfall, incremental, spiral, unified process, and collaborative object modeling and architectural design method. Some well-known agile processes are extreme programming, lean, and scrum. There are also processes that are considered prescriptive and agile, e.g., feature driven. The earliest stage for all these processes is the requirements stage. In prescriptive processes a requirements specification document is produced. In agile processes user stories and acceptance test criterion are produced. All these approaches go through the same six design phases but in differing order. For example, waterfall starts with the planning phase and follows the cycle as shown in Figure 1 one time for the whole project while agile does the same process but multiple times for smaller parts of the project.

The requirements phase is the earliest stage of the software development lifecycle. The requirements phase determines project scope, identifies stakeholders, identifies resources, identifies timelines, and develops requirements. In this work the word requirements and specification are used interchangeably although they are different. Requirements are a description of what the stakeholder desires or needs, while specifications are precise descriptions of what the system ought to be. For example: A requirement might state the system should allow users to select the color and the specification will state the system shall allow the user to select one of the following colors from a drop-down menu: red, blue, black. The design phase is when the architecture for the project is determined, and a design for the system is defined. The development/implementation phase is when the system is created. The testing phase is when the system is verified. Verification is making sure the system meets the specifications determined during the requirements phase. Validation also takes place during the requirements phase and

design phase to ensure the requirements are understood and match the desire and needs of the stakeholders.

### 2.1.1 Writing Requirements

Requirements are a collection of statements that describe the needs and desires of the users. Requirements tell what the system should do and not how the system should do it. How the system should do is described in the design phase. Requirements are one of the top reasons software projects fail. Requirements engineering comprises of several activities: elicitation, documentation and definition, specification, prototyping, analysis, review/validation, and agreement/acceptance. There are two levels of requirements elicitation: high level and low level. High level elicitation attempts to understand the business purposes and justification for the project whereas low level elicitation attempts to gather the user's needs and desires [30].

In addition to the two levels requirements also fall into two category types: functional and non-functional. Functional requirements describe what the system is supposed to do. They are described as a verb often captured through the description of a use case that results in a product feature. Functional requirements are easy to capture as they are focused on user requirements and are used to verify the functionality of the software; they describe product features. Non-functional requirements describe system qualities (i.e., user expectation vs user requirements) and are described as quality attributes; they describe product properties. Non-functional requirements are difficult to capture as they focus on the user's expectation and verify the performance of the software. Some examples of non-functional requirements are availability, capacity, data integrity, environmental, interoperability, maintainability, manageability, recoverability, reliability, regulatory, scalability, security, serviceability, and usability [31].

There are many ways to group non-functional requirements. The two main ways to group them are by impact and aspect. Impact has three grouping categories: product requirements, process/organizational requirements, and external requirements [32]. Product requirements are requirements that are influenced by the expected product behaviors such as latency or data integrity. Process/organizational requirements are influenced by business requirements such as implementation needs and limitations. External requirements are influenced by forces outside the product and business organization such as government regulations.

These same requirements can also be grouped by their aspects: operational, revisional, and transitional [32]. Operational aspects are influenced by how the system is intended to be used. Revisional aspects are influenced by how the system is intended to be changed. Transitional aspects are influenced by how the system is meant to interact within its environment. The grouping of non-functional requirements is shown below in Figure 2 below.



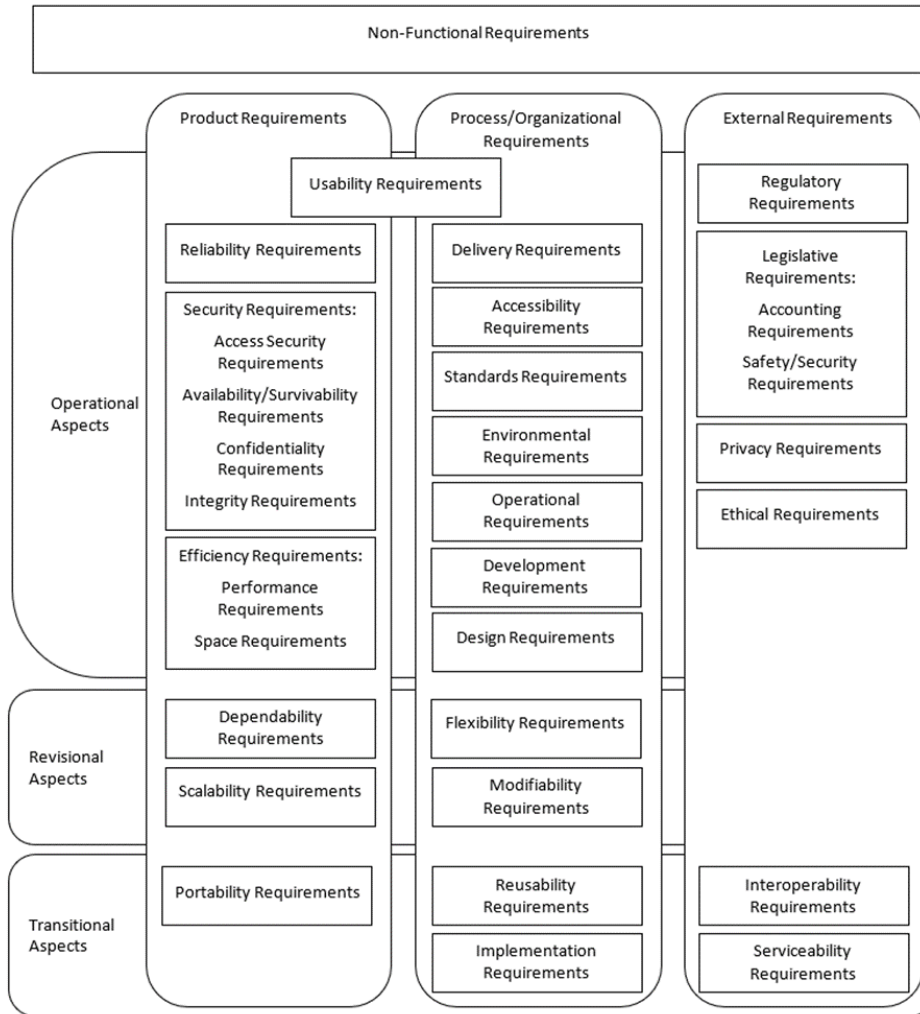


Figure 2: Non-functional requirements classification [32], [33]

It is important to get the requirements correct because they are one of the main sources of software errors but the relative cost to fix those errors are cheapest at this stage. The cost to fix software errors at the requirements stage is just 1 time the cost, at the design phase it is 5 times the cost, at the coding phase it is 10 times the cost, and at the testing phase it is 30 times the cost [23].

There are several characteristics of good requirements: they should be correct, unambiguous, complete, consistent, verifiable, and traceable [21]. The requirements should be correct thereby describing the user needs, it should not be an assumption of the user's needs. The

requirements should unambiguous because stakeholders, developers, and testers will all depend on the requirements as one source of truth therefore there should only be one interpretation of what the software will do. The requirements should be complete by representing all functionality, performance, design constraints, attributes, and external interfaces that the system shall have. The requirements should not conflict amongst each other or other documents, the requirements should be consistent. The requirements should be verifiable in order to check that the system does what it is supposed to do. Requirements should be able to be assessed whether or not the requirement has been met. Requirements should be traceable throughout the project to ensure they are being implemented and validated.

### 2.1.2 Requirements Gathering

Requirements gathering is an important part of understanding the essential parts of a system. The requirements document can be thought of as a contract between the stakeholders and the developers. A requirements specification document states the key objectives of the specifications, explains the environment the system will work in, and the design constraints. The process includes generating a list of requirements which can be functional or non-functional. These requirements are generated from various stakeholders such as customer, users, administrators, vendors, IT, staff, etc. Requirement gathering can be challenging because often stakeholders are not sure what they want because they are not aware of what is possible. Other challenges can include stakeholders not having the time and patience to participate in requirements gathering activities, stakeholders not agreeing on requirements, stakeholders not being the user of the system, and stakeholder not being the domain expert for the functions of the system.

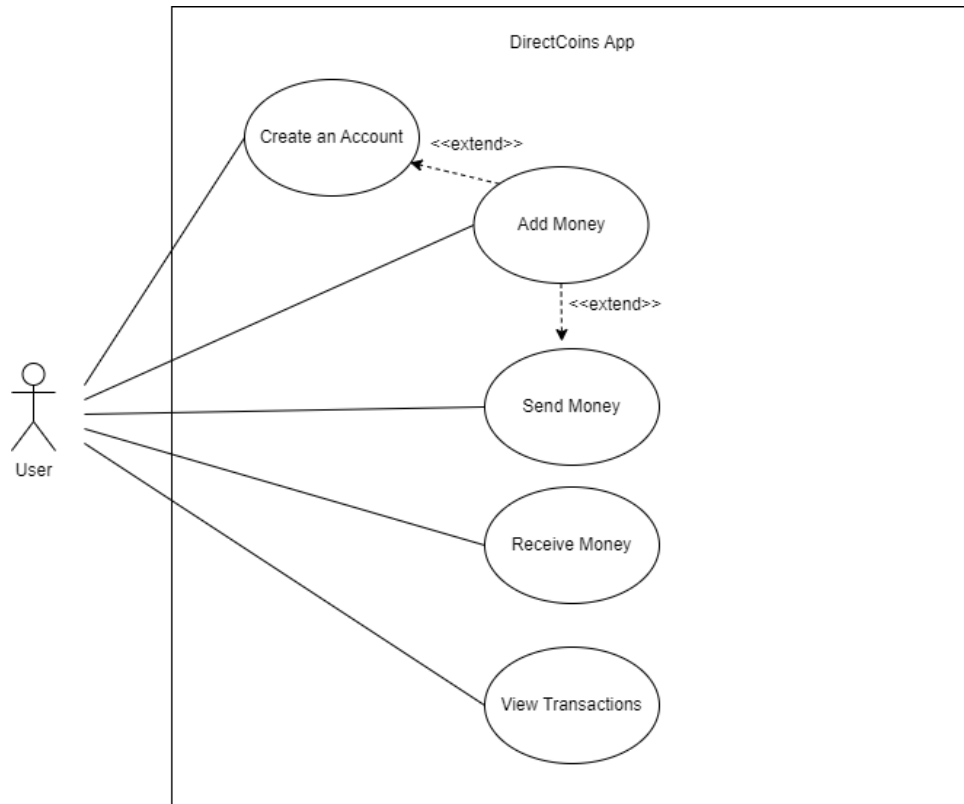
There are several techniques that can be used for requirement gathering such as interviews, questionnaires, user observations, workshops, brainstorming, role playing, use cases and scenarios,

and prototyping. Once high-level requirements, abstractions of the conceptual functions of the systems, are identified then lower-level requirements, application of functions, can be elicited through requirements analysis techniques to determine how the users and other systems will interact with the system. There are several requirements analysis techniques such as Business Process Modeling Notation (BPMN), data flow models, flow charts, Integrated Definition (IDEF) for function modeling, and Unified Modeling Language (UML). UML is well used in the development of software systems as it provides several diagrams for analyzing requirements. These diagrams are separated into two types of diagrams structure and dynamic behavior. Structure type diagrams model static parts of the system and how they relate to one another. Structure type diagrams are class, component, deployment, object, package, and composite structure. Dynamic behavior diagrams are diagrams that show change over time in the system. Dynamic type diagrams are use case, activity, state machine, sequence, communication, interaction, and timing.

### 2.1.3 Use Case models

Use Case models are a requirement gathering technique and analysis tool that describes the interaction between actors and the system. Use cases represent an abstract view of the system, its interaction with its environment and external entities as shown in Figure 3 below. Use case models also describes the system boundaries and the relationship between use case models whether through extension or inclusion. The keywords include and extend are used to signify use case inclusion (it invokes the other use case) and use case extension (the other use case adds additional steps). An actor is a user of the system which could be a human or another system. External entities are depicted as actors and can include human users, organizations, and other systems. Lines are then drawn to represent the interactions between the actor and the system. Use case models facilitate the elicitation of functional requirements by providing an abstraction of the main uses of

a software system and the actors that will interact with the system, enabling a way for requirement engineers to analyze and identify the functional needs. This assist in the analysis of the value the actor will get from interacting with the system. It helps document the information the actor and the system will exchanged, thus, highlighting requirements about the interfaces.



*Figure 3: Example use case model*

### 2.1.3.1 Scenarios

Scenarios are an extensions of use case models that describe the specific exchange of information between actors and a system as shown in Figure 4 below. Scenarios are written in steps and is where the in-depth analysis of a use case model occurs. The use case provides the general abstraction of the interaction between the actor and the system, but the scenario text provides the requirements concepts of what the system must do. The steps can include alternative flows which

describes different patterns of behavior for a system that could be taken while executing the main flow. Use cases can have one or more scenarios as artifacts and because they are written in simple text stakeholders can use the scenarios to validate the elicited behavior.

Use case models and scenarios can also be used to build other analysis models, create user interface prototypes, walkthrough design and implementation, and generate test cases.

**Use Case Name:** Send Money  
**Use Case Description:** The user wants to send money to another user.  
**Actors:** User  
**Preconditions:** The user is logged in to the system.  
**Trigger Conditions:** The user selects the send money button.  
**Flow of Events:**

1. The user searches for the receiving user they would like to send money to.
2. The system returns a list of users that best match their search entry.
3. The user selects the receiving user they would like to send money to.
4. The system prompts the user to enter an amount.
5. The system prompts the user to enter a reason for sending the funds.
6. The user enters the reason.
7. The user selects send money.
8. The system prompts the user to verify the transaction by confirming their password.
9. The user enters their password.
10. The system confirms the transaction and sends the funds to the receiving user's account.

**Alternatives:**

**Alt 1 step 4**

- 4.1 The amount selected is more than \$2,500.
- 4.2 The system returns the following error "You cannot send more than \$2,500 to any unique user per day."

**Alt 2 step 4**

- 4.1 The user enters an amount more than their available balance.
- 4.2 The system returns the following error prompt "Insufficient funds: Would you like to more add money?"
- 4.3 The user selects yes.
- 4.4 Add Money Use Case

*Figure 4: Example scenarios for the use case model shown in Figure 3 above.*

## 2.2 Security Requirement Elicitation Techniques

There have been previous efforts at capturing security requirements at the earliest stage of the development life cycle through two approaches, abuse cases and misuse cases, that attempt to enhance use case models and scenarios. In addition, there are standards for security implementations from well-known organizations such as: Open Web Application Security Project (OWASP), Cybersecurity & Infrastructure Security Agency (CISA), Carnegie Mellon University,

and MITRE. Researchers have also introduced security requirements frameworks such as: System Quality Requirements Engineering (SQUARE) and UMLsec. All security approaches and standards are inspired by the threat modeling process.

### 2.2.1 Threat Modeling

Threat modeling is an approach to assess and address the security risk of a system through the identification of vulnerabilities and their mitigations. Threat models structure the security aspects of the system through four common steps:

1. Identify – Identify the depth and scope of the analysis and the system assets to be analyzed.
2. Analyze – Analyze assets for vulnerabilities and the mitigations to the identified vulnerabilities.
3. Prioritize – Prioritize vulnerabilities to determine the order in which their mitigations will be implemented.
4. Validate – Trace asset vulnerabilities to ensure their mitigations are implemented.

There are many in depth and light approaches to threat modelling that are applied at different phases of the software development lifecycle.

Popular threat modeling approaches are:

- Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, elevation of Privilege (STRIDE), which was invented at Microsoft and is an approach that identifies threats by six categories which make up the names acronym. Stride is often applied to data flow diagrams.
- Process for Attack Simulation and Threat Analysis (PASTA) is a risk based seven step approach: define objectives, define the scope, decompose application, analyze

the threats, vulnerabilities and weakness analysis, analyze the attacks, and risk and impact analysis.

- Attack Trees are a diagrams that show identified assets and how they may be attacked.
- Abuse and Misuse cases are models that extend use case models to identify the way in which a systems can be taken advantage of. This topic is expanded on in sections 2.2.2 and 2.2.3.
- Surface Attack Analysis is a threat model that analyzes the system by looking at data paths into and out of the system. This topic is expanded on in section 2.2.4.

Section 2.2.6 below discusses security requirement frameworks which also follow the threat model analysis process.

### 2.2.2 Abuse Cases

Abuse cases are adaptations of use cases that define a sequence of actions between actors and the system that result in harm caused to the system or an actor. Abuse cases include a description of the range of security privileges that may be abused and a description of the harms that result from the abuse case. For example, imagine a system with a coupon code that is intended to be used once. If a user creates multiple accounts in order to reuse the code multiple times this would be considered an abuse case.

### 2.2.3 Misuse Cases

Misuse cases are also an adaption of use cases that specifies the unwanted behavior and interactions with the system. Misuse cases describe sequence of actions which when completed results in loss for the organization or some specific stakeholder [34]. In both abuse cases and misuse cases the analysis of security requirements requires the modeling of a malicious user

performing attacks on the system. Using the same coupon example used above in the abuse case section, if the system allows the same user to reuse that coupon code multiple time, this would be an example of misuse. An example misuse case is shown below in Figure 5 from research that expanded misuse diagrams into collaboration diagrams to add more context as to how the attack would take place against the system [35].

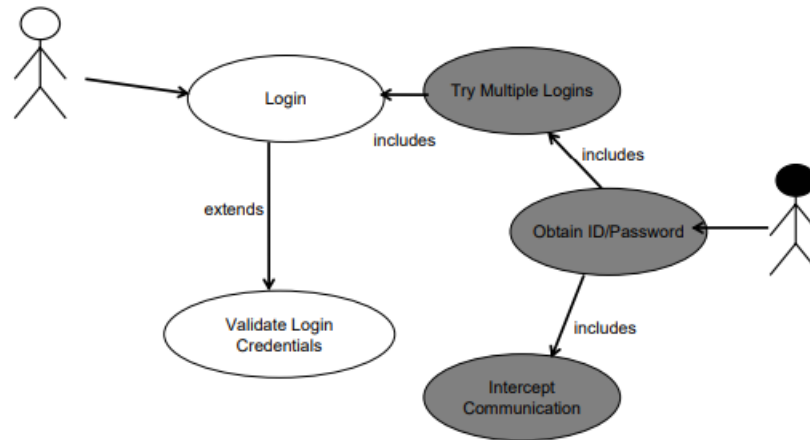


Figure 5: Example misuse case [35]

### 2.2.3.1 The Problem with Abuse and Misuse Cases

During the elicitation and analysis of requirements developers are constantly consulting with different stakeholders that are subject matter experts in the functionality for the part of the system they are developing. When developing abuse and misuse cases they do not require the input of a subject matter expert but instead rely on the imagination of the developer. Cybersecurity engineers are not consulted on the vulnerabilities the system could have. Cybersecurity engineers are often consulted after a system has been developed to do security assessments. Often the system owner is then responsible for finding a solution, usually third party, to address the vulnerabilities found. Abuse and Misuse case are accompanied by descriptions of the attack but not to the level of details that use case scenarios are written. This lack of detail limits the potential identifiable attack surface



of the system. Abuse and misuse case call for mitigations for the identified vulnerabilities but without input from security professionals the mitigation maybe inadequate to properly protect the system. In addition, abuse and misuse cases are an additional task for developers to conduct during the elicitation phase. It takes at least the same amount of effort as developing use case models.

#### 2.2.4 Attack Surface Analysis

Attack surface analysis is a technique that helps security personal determine the all the ways a system could be attacked externally; where an attacker can get access to the system and get data out. It helps identify security vulnerabilities, areas that require defense in depth, and when the attack surface has changed. The analysis considers all paths in and out of the system, the implementation that protects these paths, where data is stored, and the implementation that protects these data. The parsing of use case scenarios in this project is based on the attack surface analysis approach. Example entry and exit points are Application Programming Interfaced (API), files, user interfaces, databases, and email.

#### 2.2.5 Standards for Security Implementation

There are several cybersecurity standards that provide guidance for developing secure software such as OWASP - secure coding practices [36], CISA/Carnegie Mellon University CERT- secure coding [37], and MITRE - Common Weakness Enumeration (CWE) [38]. The OWASP-secure coding practices is a software security coding practices reference guide that can be used as a secure coding checklist that can be integrated into the development life cycle. Cert - Secure coding standard provides rules and recommendations that reflect the current thinking of the secure coding community. MITRE- common weakness enumeration is a list of known discovered software vulnerabilities. These standards and list can be used as a basis for eliciting cybersecurity requirements.

### 2.2.6 Security Requirement Frameworks

There are several well-known security requirement frameworks such as System Quality Requirements Engineering (SQUARE), Software Requirements Engineering Process (SREP), Secure Tropos, CLASP, CoRAS, and UMLsec. These frameworks lay out the process for eliciting security requirements. Some call for the advisement of security professionals in order to apply the framework. SQUARE is a framework that consist of 9 steps to elicit, categorize, and prioritizing security requirements for a system. SREP is a 9-step asset and risk driven framework that combines the common criteria (a framework for Information Technology (IT) security evaluation) with a requirements repository to elicit reusable security requirements and their associated threats. Secure Tropos is framework that focuses on security constraints, security dependencies, secure goals, secure tasks, and secure resources to elicit security requirements. Comprehensive Lightweight Application Security Process (CLASP) is a framework that focuses on roles and their associations. CoRAS is a framework that focuses on asset, threats, and vulnerabilities. Unified Modeling Language for security (UMLsec) is a framework based on UML models for the analysis of security features at the design phase.

## 2.3 Semantic Web Technologies

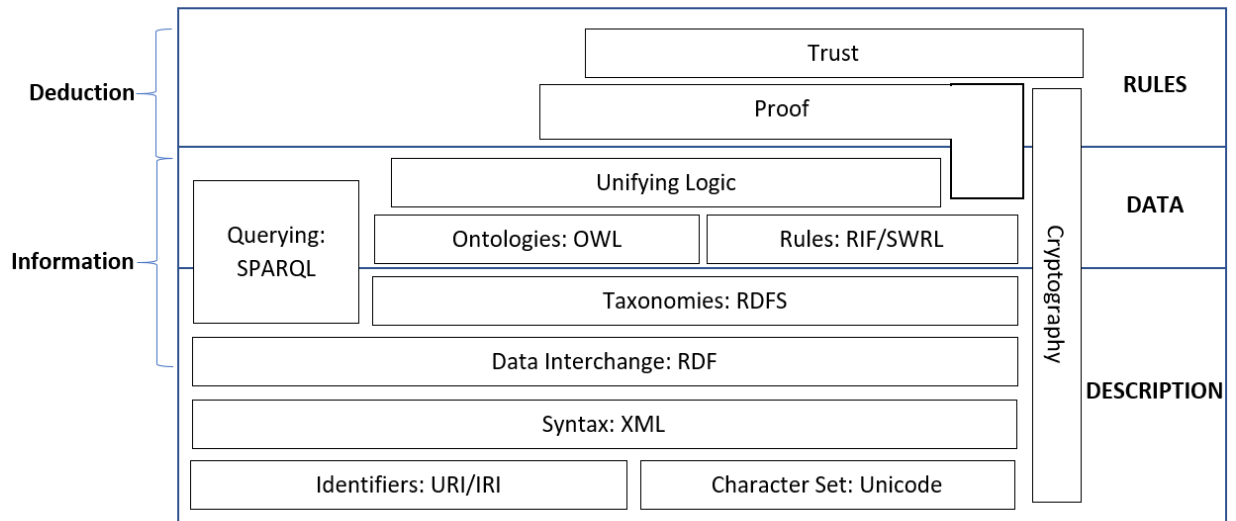


Figure 6: Semantic Web Stack [39], [40], [41]

### 2.3.1 What are Semantic Web Technologies?

Semantic means “relating to meaning in language or logic” [42]. Semantic data is data that is linked in a way that is meaningful to humans and understandable by computers. Semantic web technologies are the structure that store and use semantic data as shown in Figure 6 above. This stack is explained further in the coming paragraphs. Semantic web technologies have been used in web development but more recently has increasingly been used to power intelligent assistants. Semantic web technologies allow computers to understand the connection between pieces of data such that the computer is able to transform data into information to be consumed, thereby making it a resource. The semantic web technology architecture/stack consists of multiple parts: identifiers, character sets, syntax, data interchanges, taxonomies, ontologies, rules, querying abilities, unifying logic, cryptography, proof, trust, user interface, and applications which sit on top of this stack.

Identifiers used in semantic web technologies are referred to as Uniform Resource Identifier (URI)<sup>3</sup>. URIs are strings of characters that identify specific resources. These identifiers follow a set of syntax rules, often URI syntax also dictates a hierarchical naming scheme; a popular example is `http://` or `https://`. Identifiers allow interaction over a network by representing resources with a globally unique string. For example, in the World Wide Web (WWW), protocols are used to set the specific rules for defining URIs and exchanging resources. Two common URIs are Uniformed Resource Locators (URL) and Uniformed Resource Name (URN). Many are familiar with and understand the rules of URLs as they identify web addresses. A URL allows access to a named resource. A URN identifies a resource by name without identifying its location or how to access it. To exemplify the difference between URNs and URLs, consider a person's social security card and the same person's driver's license. A social security card is an example of a URN as it identifies an individual but does not identify the location of the individual or how to access the individual. A driver's license is an example of a URL as it identifies an individual and informs others as to how to find or access the individual.

Character sets are a collection of characters that are understandable by a computer. Semantic web technologies use Unicode which assigns a code and number to every character. Syntax is a set of rules that govern a language. In English, we consider grammar the syntax; in semantic web technologies, eXtensible Markup Language (XML) is the governing syntax.

Data interchanges are a set of frameworks that specify the governing rules for exchanging, gathering, storing, and merging data. In semantic web technologies, the Resource Description

---

<sup>3</sup> At times there are references to Internationalized Resource Identifier (IRI) instead of a URI. IRI were made to replace URIs to accommodate a larger character set. This allows for international languages. Every URI is an IRI.

Framework (RDF) is used. RDF is a set of metadata models that set the specifications for the WWW.

Taxonomies are a set of groupings. It is used for classification of data. In semantic web technologies, Resource Description Framework Schemas (RDFS) are used to define taxonomies. Ontologies are a set of grouping of groupings. The comparison of a taxonomy to an ontology can be best described as comparing a house (taxonomy) to a neighborhood (ontology), or a tree (taxonomy) to a forest (ontology). Taxonomies store data instances while ontologies store schemas. Both also imply hierarchies in data. Ontologies use Web Ontology Language (OWL), which is a web language used to represent relationships between entities sometimes referred to as the specification for ontologies.

Semantic web technologies also use a set of rules governed by Rule Interchange Format (RIF) and Semantic Web Rule Language (SWRL). Rules set a standard for exchanging rules between rule systems. SPARQL Protocols And RDF Query Language (SPARQL) is used to query knowledge bases stored within a semantic web technology stack.

There are a few more layers to the semantic web technology stack (unifying language, proof trust, cryptography, user interface, and applications) which describe ways of accessing and storing data. Some of these layers have not been fully developed and utilized, namely: the unifying language and proof trust. They are still being explored.

### 2.3.2 The Challenges of Semantic Technology

This section discusses two challenges of semantic web technology: representing modeling languages and scalability.

### *2.3.2.1 Representing Modeling Languages*

The representation of data storage, categorization, and collections is not trivial. Personal perception may be grounds for debate: the diversity of the web and its users makes it difficult to represent relationships in a way that fits everyone's point of view. Even when the modeling rules are established and followed, any representation is still based on the perception of the individual representing the modeling language.

### *2.3.2.2 Scalability*

Semantic web technologies are very scalable, but the size of the WWW is infinitely large with the amount of content that can be created. This is both a challenge and an advantage. It is difficult to efficiently scale semantic technologies accurately (i.e., making sure the data connections reflect the correct relationships). Scalability can be an advantage as it allows for the quick growth of data models.

### *2.3.3 Why Use Semantic Web Technologies?*

The semantic web is a complex, proven structure that requires technical language to create content. It has a well-established structure for storing and categorizing data. The purpose of the semantic web is to store, categorize, and create relationships between data that can quickly be shared.

### *2.3.4 With the improvements in other parts of Artificial Intelligence and Machine Learning is Semantic Web still relevant?*

Semantic web technologies fall under the artificial intelligence purview; semantic web technologies is artificial intelligence. With the improvement of other parts of artificial intelligence technologies semantic web technologies still have their place. Other parts of artificial intelligence technologies have improved greatly in identifying patterns in data and uses those patterns to

produce results. Semantic web technologies support the identification of patterns but that is a byproduct. Semantic web technologies stores known patterns. They can be leveraged to train supervised artificial technologies and to verify unsupervised artificial technologies.

### 2.3.5 What is an Ontology?

An Ontology is a data storage and categorization model that represents domains of knowledge, their properties, and the relationships between them. Ontologies find tremendous use in documenting and sharing explicit domain assumptions, understanding the structure of domain information, separating domain knowledge from operational knowledge, their ability to analyze domain knowledge, and their reuse of domain knowledge [43]. There are three defined levels of ontologies: upper or user level, middle level, and foundational level. Upper or user level ontologies are domain specific often harder to merge. Middle ontologies are domain oriented such as a music ontology or Suggested Upper Merged Ontology (SUMO) [44]. Foundational ontologies are domain independent or generic such as Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [45]. It is much easier to merge middle and foundational level ontologies.

#### 2.3.5.1 *What is the Difference between and Ontology and a Knowledge Model?*

A knowledge model instantiates an ontology. Ontologies are the general concepts of a domain while a knowledge model is the individual examples of that domain. Some ontologies do blur the line when including individuals in classes.

#### 2.3.5.2 *How to Create an Ontology?*

There is no best way to create an ontology though there are some approaches such as Simple Knowledge Organization System (SKOS) [46], Integrated Definition for Ontology Description Capture Method (IDEF5) [47], Methodology for Ontology Design based in Domain Analysis and

Layered Structure (MODDALS) [48], and Basic Formal Ontology (BFO) [49] that can be followed. Although there is no best way there are a few common steps in creating an ontology:

1. Define the scope of the ontology and questions to be answered.
2. Gather the information to populate the ontology.
3. Create an outline or draft of the ontology.
4. Formalize the ontology by adding additional properties.
5. Evaluate the ontology.
6. Share the ontology.

In step one it is important to understand the scope of the ontology and what information the ontology is to convey. This can be determined through the establishment of goals or competency questions. The next step is to gather information to create the ontology. Since the scope of the ontology is already defined it is important to consider ontologies that have already been developed. Already existing ontologies can replace the need to create an ontology or can serve as a base or extension of the ontology to be created. Now assuming that there are no available ontologies to accomplish the goals of the ontology to be created then you continue in the gathering of information and creating the skeleton of the ontology.

When creating an ontology, it is important to enumerate import terms. First, a list of concepts must be created without worrying about the overlapping ideas being represented, relationships between the concepts, properties of the concepts, or whether a certain concept is a class, subclass, or individual.

From this list of concepts then classes can be identified in the schema. Classes are often broad concepts or ideas that are defined by creating relationships to sub-concepts and ideas. For example, in cybersecurity, the Confidentiality, Integrity, and Availability (CIA) triad can be split



into three classes: confidentiality, integrity, and availability. These classes would become parent classes to subclasses that further specify the concepts inherent to the CIA triad. After the classes are defined, they are arranged in a taxonomic hierarchy. This starts to create the ontology skeleton. There are two kinds of classes primitive and defined. A primitive class has at least some conditions while a defined class has a specific criterion to adhere to.

Next, the allowed values for direct links needs to be established; these are referred to as slots. After the links, have been established, the relationships need to be filled. For example, when defining the slots for the CIA triad a direct link may be established from the subclass “authentication” to its parent classes, “access\_control”; this link would be given the “is” value. Therefore, authentication is access\_control. This creates a triple: entity, relationship, entity.

The next step is to create a hierarchy to start to formalize the ontology. This can be approached in three ways: top –down, bottom-up, or a combination of both. In a top-down approach one would begin with the general concept and work their way down to the subclasses of these concepts. In a bottom-up approach, a person would start with the sub classes and work their way up to the general class. In a combination approach one would create their ontology like a spider web from which one can pinch a general topic and show the hierarchy. Defining a hierarchy allows for inheritance between parent and child classes.

Formalizing the model means adding structure and properties to the ontology to in order for a computer to interrupt the ontology. The next step is to continue to define add properties to the slots, which are properties of the classes. There are four general object properties that can become slots: intrinsic, extrinsic, parts, and relationships. Intrinsic properties are properties that naturally belong to the concept; they are essential to defining the concept. A property that is intrinsic to a person’s identity would be their DNA. Extrinsic properties are properties that are not

essential to the concept and often are influenced from the outside of the concept. A property that could be argued is extrinsic to a person's identity is their first name. It is normally given to a person by their parents. It helps identify a person, but many people may share the same name, so it only helps narrow the search scope. Parts are things that belong to that concept, e.g., such as a heart is a part of a body. Relationships are how concepts relate to each other, e.g., a mother would have parent relationship to a child.

There are other properties that are not distinctly defined through slots but through the modeling that should also be considered such as transitive, symmetric, asymmetric, reflexive, and irreflexive. Transitive implies which entities precede another. For example, a grandmother precedes her daughter who precedes her granddaughter. This has a transitive property that the grandmother precedes the granddaughter. Symmetric properties imply which entities are equal. For example, synonyms. Asymmetric properties imply a one-way relationship. For example, a daughter cannot precede her mother. Reflexive properties imply when two entities are the same though described differently. For example, George Washington can be a founding father as well as the first president of the United States. Irreflexive properties imply a property cannot have a relationship with itself as it does with other entities [50]. For example, a person cannot be born before they are born.

Next, the facets of the slots need to be identified; this is similar to declaring a variable in a programming language. For example, slots can have the following value types: string, number, boolean, enumerated, and instance. Strings are a set of characters, numbers represent numerical values, booleans represents true or false values, and enumerated type represent identifiers that are constants in language, e.g., card suits: club, spades, diamonds, and hearts. An instance is a type used to define the relationships between concepts. In the case of an instance, the allowed classes,

an instant slot can link to are labelled as the range and the classes that an instance may attach to are called the domain of the slot. Slot cardinality must also be defined. Some slots may be single cardinality while others may be multiple cardinalities. To create instances of a class, a class must first be selected. Then an individual instance of that class must be created with the slot values filled with actual data. For example: assume you have a class “bottled\_water” it has the following slots properties and values:

Type: Still, Sparkling

Source: Spring, Distilled, Reverse\_Osmosis

Flavor: Unflavored, Lemon, Cucumber, Strawberry

PH: 7, 8, 8.5, 9, 9.5

When adding an instance to the class this creates an individual which is an instantiation of the class. One could add the Essentia brand bottled water which would be still, Reverse\_Osmosis, Unflavored, 9.5. There can also be inverse slots which allows for properties to flow both ways.

When defining classes and a class hierarchy, many guides will state that it is important to keep in mind the “is-a” or “kind-of” relationship between classes but that is to make sure all triple relationships make sense. One does not have to only use the “is-a” relationship between classes. Any word(s) that describes the relationship between two entities is acceptable. Often ontology builders start with predicates before moving to more complex relationship descriptions.

It is important to keep the scope and goals of the ontology in mind when developing an ontology. An ontology does not have to define every possible instance if it is outside the scope of the ontology. Other things to consider when building an ontology are naming conventions often camel case or underscore and whether to pluralize collections of objects. In addition to applying relationships, defining properties is also important to define when classes are disjoint. Disjoint classes are classes that do not share an instance with one another. Since ontologies are built to use

reasoners to discover unspecified relationships it is important to specify when a relationship does not exist if it is known. Instances of classes that are disjointed cannot be of the other class. An example of this is a student cannot be both absent and present in a class. The classes absent and present would be disjointed. All these tips help formalize the ontology in order to use reasoners and queries.

The next step is to evaluate the ontology. There are several ways to evaluate an ontology by checking for different evaluation metrics such as: accuracy, completeness, adaptability, clarity, computational efficiency, and consistency. This can be done through gold standard-base, corpus base, task-base, and criteria base. The gold standard base compares the created ontology to previously created ontologies. The corpus base compares the ontology to domain corpuses to see how much of the domain the ontology covered. The task-based approach measures how an ontology helps improve a certain task. Criteria based measures how an ontology follows a predefined criteria [51]. Another method for evaluating an ontology are competency questions which are determine during the scope definition and goals step of the ontology development. The ontology is queried to answer these questions.

The last step is to share the ontology. Ontologies can be shared through GitHub, Onothon, and Kbox to name a few. They can also be shared through a DOI or permanent link. It is important that shared ontologies are accompanied by the proper open-source license such as creative commons, and opensource to encourage the community to use them.

### 2.3.6 Use of Semantic Technologies in Gather Requirements

Semantic Technologies ontologies have been used for capturing requirements engineering knowledge. Ontologies for requirements engineering knowledge have captured the software

engineering book of knowledge, requirements specification document, requirements ontology, and application domain.

### 2.3.7 The Use of Semantic Technologies in the Cybersecurity Domain

The development of cybersecurity ontologies has been of interest to the MITRE, ARL, and DARPA [52], [53], [54]. The current development of ontologies has focused on cybersecurity situational awareness, capturing cybersecurity threat indicators, capturing incident responses, and integrating the knowledge base of cybersecurity threats. The target audience of these ontologies are usually cybersecurity analysts. Cybersecurity from a system perspective can be classified into two categories, either the design of an optimal system or the address of problems appearing from system operations [52]. There has been little attention in the development of ontologies to help address the design of optimal systems. Most ontologies focus on capturing knowledge of problems that appear from system operations.

## 2.4 Machine Learning

Machine Learning (ML) is a sub-field of Artificial Intelligence (AI). AI is the scientific study of the theory and development of computer intelligence to perform tasks that would normally require human interaction. Examples of this are visual recognition, speech recognition and understanding, and decision making. ML is the use of AI for a computer system to learn without explicitly being programmed with the learned knowledge. UnSupervised Machine Learning (USML) does not have labeled data therefore the algorithm is not trained to make a conclusion but to instead group data. USML instead only has inputs and no feedback system. These models depend on the probability distributions of the data or clustering if the data is not normalized in order to make conclusions. Supervised Machine Learning (SML) uses a feedback system to learn. SML models takes in inputs,

analyzes it based on what it has learned already from the training set, and produces an output; the accuracy of the output is rated and fed back into the model in order to update the model [55]. Human interaction is required to set up and train the model but once the model has met accuracy thresholds human interaction is no longer needed for the model to continue to perform. ML underperformance is often due to overfitting and underfitting. Overfitting a model is when a model is trained to account for the noise of the training data. This usually negatively impacts the performance of the model in the future because it believes the noise in the training data is part of the data details. On the other hand, is underfitting which usually occurs when data is limited, or the training data set does not account for new generalizations. Much like any other predictive analysis models it is important that the training set properly identifies the problem the model is to make decisions about; learning depends on the quality of the training data.

## 2.5.1 Approaches in Machine Learning

This section discusses the six common approaches in machine learning: supervised, unsupervised, semi-supervised, reinforcement, ensemble learning, and neural networks/deep learning [56].

### 2.5.1.1 Supervised Learning

Supervised learning is when a model is trained with a data set that labels the input with an expected output. The expected output is called the supervisory signal. This trains the model to make decisions based on inferences for new data. Supervised learning approaches are used in problems where data links are well understood. There are two categories of supervised learning: regression and classification. Regression is measuring data points against the mean of the data set and assigning a value to each data point. Example regression models are Linear Regression, Polynomial Regression, and Ridge Regression. Classification is the grouping of data points based

on characteristics. Example classification models are SVM, Decision Trees, Logistic Regression, Naïve Bayes, and K-NN.

#### 2.5.1.2 Unsupervised Learning

Unsupervised learning is when the model is trained with a data set that has the input but does not label the expected output. There are no indicators on whether or not the inferences are correct. Unsupervised learning approaches are used in problems where the data links are not well understood, or researcher are trying to discover new relationships. These methods are used to discover data structures without defining a structure. This approach is usually tackled by clustering data points to create categories. The three categories of unsupervised learning: Clustering, Associated Rule Learning, and Dimensionality Reduction. Clustering is the grouping of data based on similarities. Example of clustering algorithms are K-means, Birch, Mean Shift, and Gaussian Mixture Model. Associated rule learning maps data points based on their dependency of other data points. Examples of associated rule algorithms are Apriori, Eclat, and FP-growth. Dimensionality reductions is the transformation of data from a high dimension to a lower dimension such as 3-D to 2-D. Examples of dimensionality reduction algorithms are Singular Value Decomposition and Isomap Embedding,

#### 2.5.1.3 Semi-supervised Learning

Semi-supervised learning is the use of both approaches of supervised and unsupervised learning. It is used when some data links are understood, and others are not or the cost of labeling all known data links is too high. This approach has the benefit of being more accurate than unsupervised learning but without all the cost of supervised learning.

#### 2.5.1.4 Reinforcement Learning

Reinforcement learning works on a rewards system that maximizes the reward for agent actions when the action is favorable and limits the reward when the action is unfavorable thereby encouraging and discouraging actions. Therefore, two outcomes of reinforcement learning are positive reinforcement and negative reinforcement. This approach is used when the exact error of the data is unknown or when the data environment is dynamic or non-deterministic [55]. Supervisory learning approaches help establish data relationship links while reinforcement learning approaches establishes decision making. Reinforcement learning algorithms can be model free, or model based with agents that are on-policy and off-policy. Model free algorithms rely on trial and error while model based rely on the transition probability. An example of model base algorithm is Alphazero and model free is Deep Q Network (DQN). On-policy agents attempt to improve the policy used to make decisions while off-policy agents improve the policy not based on the decision maker. An example of off policy algorithm is Q-learning and on-policy is State-Action-Reward-State-Action (SARSA).

#### 2.5.1.5 Ensemble Learning

Ensemble learning establishes multiple hypothesis to solve the same problem. This is done by establishing multiple learners with different approaches to the problem and combines them to solve the problem. This approach is usually more effective as it helps boost weak learners. There are two approaches to ensemble learning algorithms sequential and parallel ensemble. There are three categories of ensemble learning: stacking, bagging, and boosting. Stacking is training a model to combine the predictions of several other learning algorithms. Examples of stacking algorithms are Canonical Stacking, Blending, and Super Ensemble. Bagging is a form of bootstrapping that allows all the learners to vote with equal weights on random samples selected for the training set.



Examples of bagging ensemble learning algorithms are Canonical Bagging, Random Forest, and Extra Trees. Boosting is incrementally training the learners. Each reiteration of training emphasizes the misclassification or the previous instance. Examples of boosting ensemble algorithms are AdaBoost, Gradient Boosting Machines, Stochastic Gradient Boosting.

#### 2.5.1.6 Neural Networks and Deep Learning

Neural networks and deep learning are models inspired by the brain that are designed to recognize patterns that in turn assist with clustering and classifying. These models are trained by looking at examples and making conclusions of those examples. Deep learning algorithms are stacked neural networks algorithms. Neural networks utilize nodes to represent relationships between classifications. There are five categories of neural networks and deep learning: Perceptrons, Convolution Neural Networks (CNN), Recurrent Neural Networks (RNN), Generative Adversarial Networks (GAN), and Autoencoders. Perceptrons are binary classification algorithms. An example perceptron algorithm is Multilayer Perceptrons (MLP). CNN (also the name of the algorithm) are MLP and all nodes are connected to one another. RNN recognize sequential characteristics and use the recognized pattern to predict the next conclusion. An example RNN algorithm is Long Short-Term Memory (LSTM) networks. These algorithms exhibit dynamic behavior of data sets therefore the time and dependency of relations of data point is important. Generative Adversarial Networks (also the name of the algorithm) pits two neural networks against each other to produce the best dataset output that most closely resembles the training set statistics. Autoencoders (also the name of the algorithm) uses dimensionality reduction to remove noise from datasets.

### 2.4.1 Natural Language Processing

Natural Language Processing (NLP) is the ability for a computer to analyze and understand human language in order to make decisions. NLP is a popular application of several machine learning algorithms to achieve information extraction, spam filters, sentiment analysis, auto predict, auto correct and speech recognition. NLP uses all of the techniques discussed in the previous section but in order to work with language the data needs to be transformed in a manner for the computer to do statistical analysis. This processing is referred to as vectoring. Vectorizing is mapping word to numbers to make predictions. There are several steps to preparing data for vectorizing: tokenization, stemming, lemmatization, part-of-speech tagging, named entity recognition, and chunking. Not all steps need to be performed it depends on the data and goals of using NLP. Tokenization is the breakdown of sentences into tokens often into individual words. During this process stop words can be identified and eliminated from the data. Stop words are commonly used words such as 'the', 'is', and 'and'. Eliminating stop words is helpful when trying to balance data. Stemming strips, the word to the root word by removing pre- and suf-fixes. Lemmatization groups words with liked meaning such as the present, past, and future tense of the word referring to just the root word. An example of lemmatization is of the word 'eating'. The root lemma word would be to 'eat'. 'Eaten' would also reduce to 'eat'. Parts of speech tagging would tag words as noun (n), verb (v), adjective (a), and adverb (r). Named entity recognition were named entities are classified into categories. Chunking is recombining tokens into large tokens that assist in predicting meaning. Depending on the type of data and the expected outcome of the NLP model all of these preprocessing techniques may or may not be used.

There are many well established vectorization algorithms that implore a combination of the preprocessing steps above and map the words to numbers: Bag of Words, Term Frequency-Inverse Document Frequency (TF-IDF), Word2Vec, Global Vectors (GloVe), and FastText.

Bag of Words tokenizes, creates a vocabulary, and they vectorizes. It splits sentences up into individualized words then it creates a vocabulary of unique words and finishes up by creating a sparse matrix with the frequency of each unique word. The columns of the matrix would represent the unique words and rows would represent each sentence that is processed. An example sparse matrix is show in below for the following two sentences “It is a lovely day today”, and “Yesterday was a lovely day”.

it	is	a	lovely	day	today	yesterday	was
1	1	1	1	1	1	0	0
0	0	1	1	1	0	1	1

*Table 2: Example sparse matrix for text vectorization*

TF-IDF attempts to determine how important a word is by giving less frequent words more weight. To accomplish this three equations, Equation 1: Term Frequency, Equation 2: Document Frequency, and Equation 3: Reciprocal of Document Frequency are calculated to determine the parts of Equation 4: TF-IDF.

$$TF = \frac{\text{Frequency of word}}{\text{Total number of words}} = TF(t, d) = \frac{f_d(t)}{\max_{w \in d} f_d(w)}$$

*Equation 1: Term Frequency*

$$DF = \frac{\text{Documents containing word } W}{\text{Total number of documents}}$$

*Equation 2: Document Frequency*

$$IDF = \log\left(\frac{\text{Total number of documents}}{\text{Documents containing word } W}\right) = idf(t, D) = \ln\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$$

*Equation 3: Reciprocal of Document Frequency*

$$TF - IDF = TF * IDF = TF(t, d) * IDF(t)$$

*Equation 4: TF-IDF*

Word2Vec uses neural networks to make the vectorization contextually aware by not ignoring the semantic of words. Word2Vec is often used when analogies are important.

GloVe expands on the basis of semantic importance of Word2Vec but doesn't only consider the relationship of words in the same sentence but across sentence. It has greater attention. GloVe is best used when analogies are important and works well on small datasets.

FastText is similar to Word2Vec and GloVe but instead of looking at relationships at the word-to-word level it looks at understanding how the combination of characters provides meaning of words. FastText is advantageous with datasets that contain unusual or very less frequently used words. Once the data is vectorized it can usually be used just as any other dataset.

### 2.5.2 Why Use Machine Learning?

ML helps identify unknown relationship links in data and supports computer system decision when a set of inputs are provided. This project will take a set of use case scenarios and output a set of cybersecurity requirements. This will not only require a storage method for data link relationships but also a need for automatically identifying and updating data link relationships as the human interaction with the system will be limited. ML can be used to speed up the patterns in linking functional requirement descriptions to the cybersecurity requirements as it can be used for

classifying requirements and pattern recognition between the requirements to help identify new relationships.

#### 2.4.2 How to Know which ML model to Use?

Choosing the right ML model starts with understanding the problem you are trying to solve, your data set, and the outcome that ML is expected to provide. Does your problem require the outcome to be text extraction, categorization, value prediction, recommendation, new pattern detection? It is important to understand your problem because it will drive the expected outcome. Once the problem is identified and the expected outcome is identified the next thing to consider is the type of data and features you will feed your model. Is it numbers, categories, text, pictures, audio, video? Is your data labeled? If it is labeled, you can easily use a supervised learning technique. If it is not labelled and the outcome does not have a fixed outcome, then an unsupervised learning model is probably a good fit. SciKit Learn [57] and Microsoft [58] both provide cheat sheets or diagrams to help choose the right estimator based on the dataset available and the expected outcome. The best way to choose a model is to understand your problem, the applications of the ML models, the advantages and disadvantages of the models, and the data that is available to train the models.

#### 2.4.3 How to evaluate Machine Learning Models?

Each type of machine learning algorithm has established metrics for determining performance. Since this research will use classification models this section will focus on how to evaluate classification models. The metrics for evaluating a classification model are accuracy, precision (positively predicted value), recall (sensitivity, true positive rate), F1 Score, specificity (selectivity, true negative rate), fall-out (false positive rate), miss rate (false negative rate), and Receiver-Operator Curve (ROC Curve)/ Area Under the Curve (AUC).

### 2.4.3.1 Confusion Matrix

In order to understand these metrics first it is important to understand a confusion matrix.

		Predicted Value	
		Positive	Negative
Real Value	Positive	TP	FN
	Negative	FP	TN

*Figure 7: Confusion Matrix*

A confusion matrix is a cross tab of predicted values and real values. TP stands for true positive which means that the real value is positive and the predicted value is also positive, FP stands for false positive which means the real value is negative but the predicted value is positive, this is also referred to as a Type I error, FN referred to as a Type II error stands for false negative which means the real value is positive and the model predicted value is negative, and TN which stand for true negative and means the real and predicted values are both negative.

These metrics are important in evaluating how well the model is performing but also in understanding how often the model is wrong when making a decision. This can be crucial depending on the domain in which the model is deployed. For example, in the medical field it is important to properly diagnose a patient. A TP and TN are great, but a FN can be very devastating for diseases that continue to worsen with time. An FP can also be devastating assuming there are no other ways to confirm the disease. The doctors may give a patient a treatment they do not need. Therefore, it is very important to consider the consequences of a model's mis-predictions.

### 2.4.3.2 Accuracy

The accuracy metric is the percentage of correct predictions of all the predictions made. The accuracy equation is shown in Equation 5 below. Accuracy is not a reliable metric when the data

set is unbalanced. A balanced dataset is a dataset that has an equal number of positives and negatives, for an equal distribution amongst data classes. If the data is heavily skewed to one side, it is considered unbalanced. This may cause the accuracy metric to give a false idea on the model's performance.

$$Accuracy = \frac{\text{Number of correct prediction}}{\text{Total number of predictions}} = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

*Equation 5: Accuracy Metric*

#### 2.4.3.3 Precision

The precision metric shown in Equation 6 below, also referred to as the positive predicted value, is the number of true positives predicted over the number of true and falsely predicted positives.

$$Precision = \frac{\text{Number of correctly identified positives}}{\text{Total number of positive predictions}} = \frac{TP}{(TP + FP)}$$

*Equation 6: Precision Metric*

#### 2.4.3.4 Recall

The recall metric, also referred to as the sensitivity or true positive rate, measures the model's sensitivity which is the ability of the model to determine all of the relevant instances in the dataset.

The equation is shown in Equation 7 below.

*Recall*

$$\begin{aligned} &= \frac{\text{Number of correctly identified positives}}{\text{The number of correctly identified positive and wrongly identified negatives}} \\ &= \frac{TP}{(TP + FN)} \end{aligned}$$

*Equation 7: Recall Metric*

#### 2.4.3.5 F1 Score

The F1 score shown in Equation 8 below, is one of the most reliable metrics for evaluating classification models. It determines if there is a good balance between recall and precision. Later in this chapter in the tradeoff section there is a discussion on why a good balance between metrics are important.

$$F1 = 2 \left[ \frac{(Recall * Precision)}{(Recall + Precision)} \right]$$

*Equation 8: F1 Score Metric*

#### 2.4.3.6 Specificity

The specificity metric shown in Equation 9 below, also referred to as selectivity or true negative rate, is the rate of true negative predictions that are correctly identified.

*Specificity*

$$\begin{aligned} &= \frac{\textit{The number of correctly identified negatives}}{\textit{The number of correctly identified negatives and wrongly identified positives}} \\ &= \frac{TN}{(TN + FP)} \end{aligned}$$

*Equation 9: Specificity Metric*

#### 2.4.3.7 Fall-Out

The fall-Out metric shown in Equation 10 below, also referred to as the false positive rate, is the likeness of the model to predict a positive value when there are no positive values.



*Fall – Out*

$$\begin{aligned} &= \frac{\textit{The number of wrongly identified positives}}{\textit{The number of correctly identified negaives and wrongly identified positives}} \\ &= \frac{FP}{(TN + FP)} \end{aligned}$$

*Equation 10: Fall-Out Metric*

#### 2.4.3.8 Miss Rate

The miss rate metric shown in Equation 11 below, also referred to as the false positive rate, is the number of positive true values incorrectly predicted as negative.

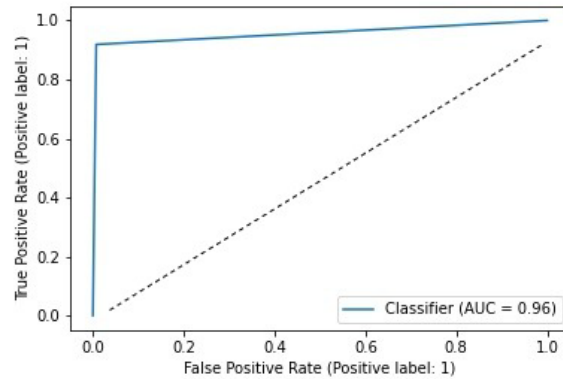
*Miss Rate*

$$\begin{aligned} &= \frac{\textit{The number of wrongly identified negatives}}{\textit{The number of correctly identified positves and wrongly identified negatives}} \\ &= \frac{FN}{(TP + FN)} \end{aligned}$$

*Equation 11: Miss Rate Metric*

#### 2.4.3.9 ROC/AUC

The Receiver-Operator Curve (ROC) and Area Under the Curve (AUC) is a chart of the relationship between sensitivity and fall-out. This metric is referred to when determining the model's performance. The ROC chart plots the true positive rates vs the false positive rate. There is a dashed line going from the bottom left corner where the axes meet to the top right corner. This line represents the 50% chance threshold of the model predicting correctly. The other line is AUC which has the best performance when far away from the dashed 50% line and hugging the upper left-hand corner of the chart.



*Figure 8: Example ROC curve*

#### 2.4.3.10 Metric Tradeoffs

There are several tradeoffs to consider when developing a model as having a perfect model is rarely achievable. It is often a sign of error when a model performs at 100%. This is a signal of overfitting. This is not ideal unless every single permutation of input and outcome were determined and used to train the model. The reason being is that the model is likely to perform poorly when presented with new datasets that contains combinations not seen previously. The F1 score determines the balance between precision and recall. There is a tradeoff between precision and recall when increasing precision recall decreases and vice versa. There is an optimal threshold for this trade off. This threshold is best determined by the problem and the deployment domain for the model. Usually, the threshold makes the model biased towards one of these two metrics. For high stake domains where the model is alerting the user to a problem it is preferable for the model to be biased towards recall. It is better to have more false positives than a false negatives. For low stake domains precision is preferred because it is more important for the model to be correct more often than wrong. As stated, this is not true in all examples provided it truly depends on the problem, the deployment domain, and the expected model outcome.

## 3 Approach

This chapter discusses the goals of this research, the proposed approach, the development of the approach, the technical decisions behind the approach, and how the approach will be used.

### 3.1 The Goals of this Research

Cybersecurity requirements are more challenging to elicit than other requirements because they are nonfunctional requirements that requires cybersecurity expertise and knowledge of the proposed system in order to elicit them.

The goals of this research are to generate cybersecurity requirements based on knowledge acquired from requirements elicitation and analysis activities, to provide cybersecurity specifications without requiring the specialized knowledge of a cybersecurity expert at the earliest stage of the development lifecycle, and to generate reusable cybersecurity requirements.

### 3.2 The Approach

This section discusses the approach followed to achieve the goals and ultimately answer the research questions.

#### 3.2.1 Scenario Driven Security Requirements Elicitation (SD-SRE)

The approach leverages use case scenarios and uses cybersecurity best practices and existing requirements to create a secure development ontology that then uses the ontology to label the use case scenarios in order to train a ML model to identify potential security requirements. This approach is named the Scenario Driven Security Requirements elicitation (SD-SRE). The SD-SRE utilizes a web portal, i.e., the SD-SRE portal, which developers can use to enter their use case scenarios and get security requirement suggestions for the system they are designing. The proposed system employs a ML model that is trained on use case scenarios that were labeled based on a

secure development ontology. The efficacy of the approach was validated through an experiment presented via the web SD-SRE portal that allowed new users to contribute use case scenarios and security requirements as a comparison repository.

### 3.2.2 The SD-SRE Process

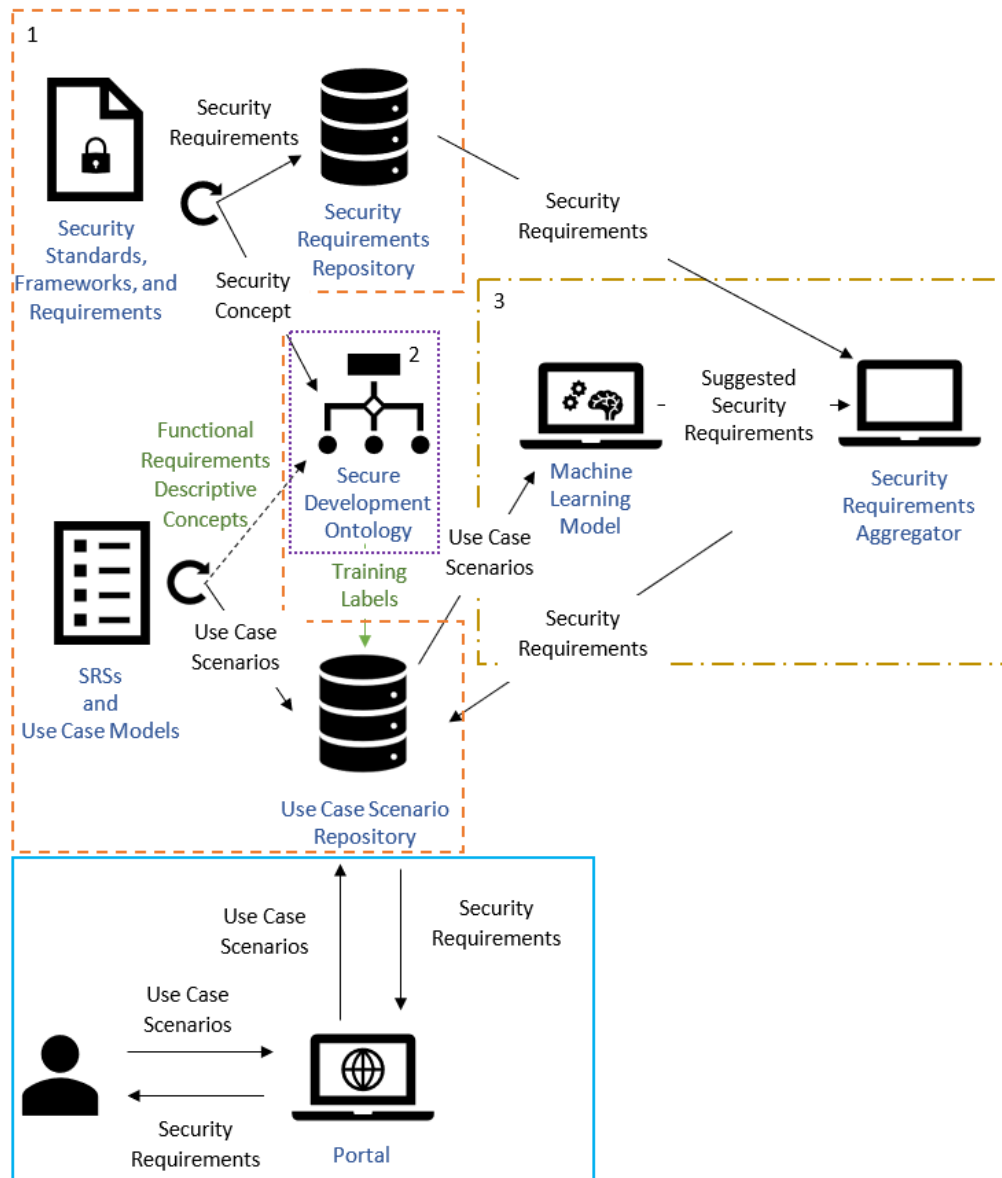


Figure 9: The SD-SRE process

These are the following phases that encompassed the SD-SRE development process:

1. Information gathering phase
2. Building the ontology
3. Determining the ML model
4. Validating the SD-SRE

Figure 9 describes the overview of the whole process discussed further in the steps listed above.

### 3.2.3 The Information Gathering Phase

There are several pieces of information that need to be gathered at the same time. One of the pieces of information that needs to be gathered continuously throughout the project are use case model scenarios and requirement specification documents. A repository is used to host the use case model scenarios in a format that could easily be digested by a vectorizer in order to run the scenarios through the machine learning model.

Another step taken is the analysis of already existing cybersecurity and requirements engineering semantic web projects to determine if they can be leveraged in the SD-SRE to generate cybersecurity requirements. There are several already existing cybersecurity and requirements related semantic web projects. The reuse of existing projects is an advantage of semantic technologies that can be leveraged here to build the new knowledge base. The currently existing projects that capture security knowledge for systems that are already deployed. They tackle security after the fact such as intrusion detection. For this research the ontology captures secure development knowledge for the building of optimal systems. There are also many projects that capture requirements engineering knowledge but many of them cover the theories of requirements engineering such as the Software Engineering Body of Knowledge (SWEBOK) and not

requirements themselves or they are more of a knowledge model than an ontology as they capture requirements for a particular system.

The next step is to analyze secure system implementation standards and existing security requirements techniques, frameworks, processes, and methodologies to map the domain and to gather cybersecurity requirements. The existing standards and frameworks provide a basis of cybersecurity expertise to generate best practice requirements. These initiatives are often used in development today to assist with generating security requirements and are a good base for a best practice secure development requirements knowledge domain. These steps are continuous through the development of this approach, labeled 1, and highlighted in Figure 9 above by the orange dashed line (- - - - -).

#### 3.2.4 Developing the Ontology

The first step to developing the ontology is to determine the scope of the ontology and competency questions. Competency questions were created based on the analysis of use case scenarios and security artifacts. Then cybersecurity development requirements knowledge base is mapped using semantic web technologies to make connections between the best practices discovered in the security artifacts to frequently used functional features discovered in the use case scenarios. The use case model scenarios were re-analyzed for missed functional features and to determine the word(s) used to best describe these features using the attack surface analysis approach (identifying parts of the system where data enters and leaves the system). These descriptions are also mapped in the ontology. The ontology is then tested with the competency questions created from use case scenarios. The ontology is available here: <https://github.com/JessicaSteinmann/SDO.git>. Figure 9 below shows a snapshots of the ontology for `permission_based_access_control` (PBAC) and `password_spraying` attack.

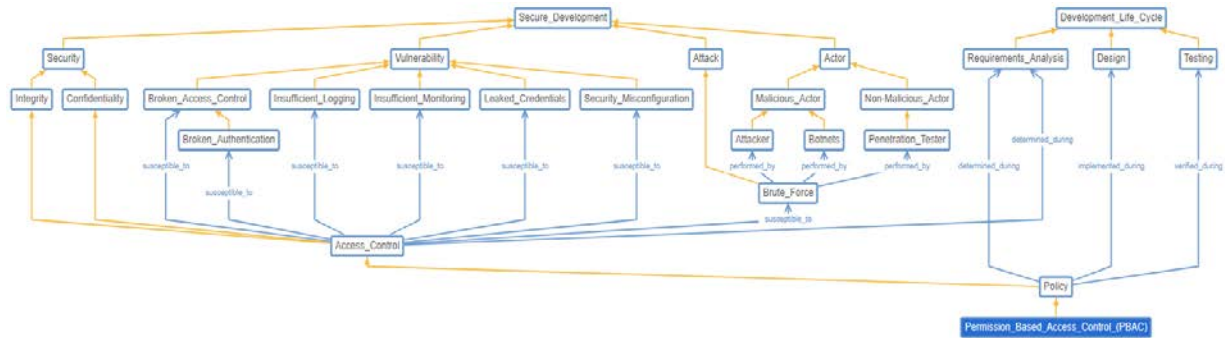


Figure 10: SDO snapshot of permission\_based\_access\_control\_(PBAC)

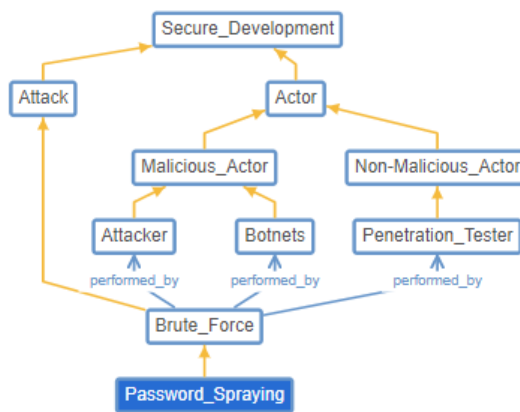


Figure 11: SDO snapshot of password\_spraying vulnerability

This step is labeled 2 and highlighted in the purple dotted line (.....) in Figure 9 above. Figure 12 below shows the ontology development process.

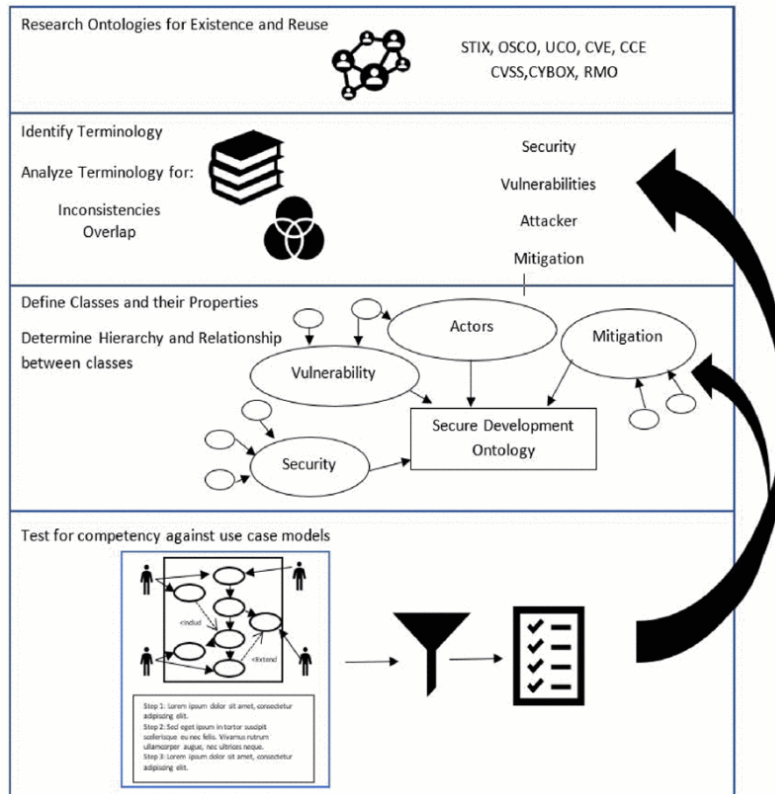


Figure 12: The ontology development process [59]

### 3.2.5 Determining the ML Model

After the secure development requirements domain knowledge is mapped. It is parsed for the terms that described functional features and their attached generic security topic. This information is used to automatically label the scenario steps in the use case repository.

A classification machine learning model is then trained on 70% of the labeled scenario steps to determine whether a scenario step triggers a generic security topic. The system is tested with the remaining 30% of the scenarios. The test train split was increased as the model performance increased. The final model is based on a 90/10 split. If a topic is triggered the system labels it a yes in the repository. Table 3 below shows the model iterations for the train test spit.



Model Iteration	Train (% of labeled data)	Test (% of labeled data)
1,2,3,4	70	30
5	80	20
6	90	10

Table 3: Train test split for model testing

Then the system aggregator deduplicates the requirements by project and pulls the requirements from the cybersecurity requirements repository. This process is labeled 3 and highlight in Figure 9 above by the gold dash dot line (— · — · —). Figure 13 below show the data flow of the above process. The aggregator returns the suggested security requirements to the SD-SRE portal.

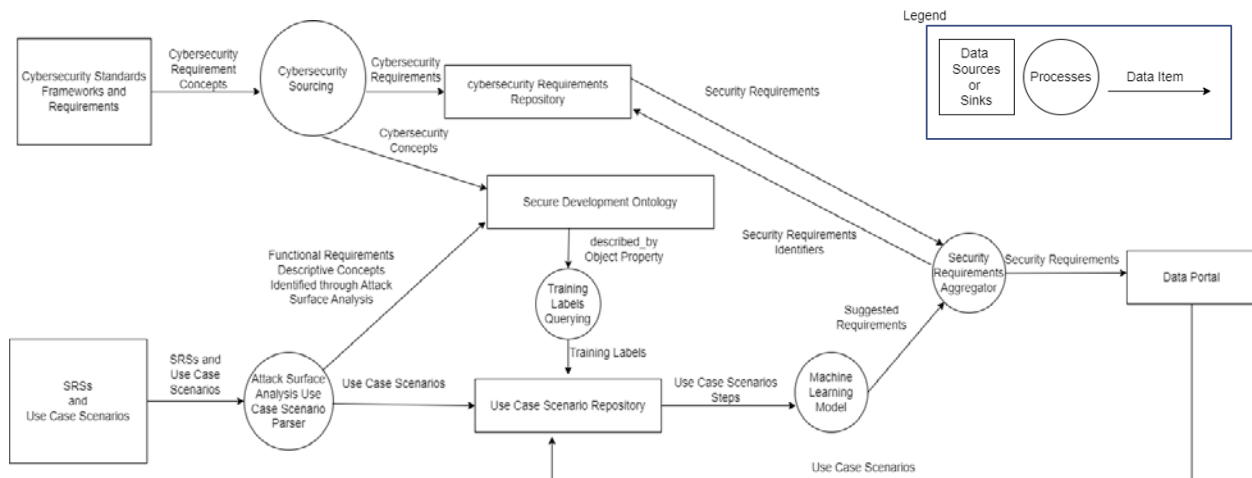


Figure 13: Data flow diagram

### 3.2.6 The Aggregator

The aggregator performs two functions gathers liked topics and deduplicates the requirements. The aggregator is required because the security requirements are grouped into 11 distinct topics but some of the topics overlap. For example: email collection would also employ requirements from input validation and database, login would also employ requirements from authentication,

authorization, logging, input validation, and encryption. The aggregator needs to pull those requirements and return them along with the email collection or login requirements in case they weren't flagged by other scenario steps. The aggregator will also need to deduplicate the requirements because the model is run on each scenario step therefore a requirement may be duplicated and would need to be unduplicated for the end user.

Figure 14 below shows an example of how the aggregator works. The first table shows the requirements flagged by the model. As shown, there are duplicated requirements and missing requirements. In the middle table the aggregators pulls the missing requirements and groups the like requirements. There are missing requirements because requirements such as authorization has a place in the login process, but it also has a place in accessing databases and other system resources. The authorization requirements were not looped into the login requirements as they have unique requirements that can apply to situation other than login. Therefore, if the system doesn't flag authorization requirements but flags login requirements then the system would pull the authorization requirements also. In the third table the aggregator deduplicates the requirements and these are the requirements presented to the end user. This is done to avoid having duplicated requirements in multiple topics.

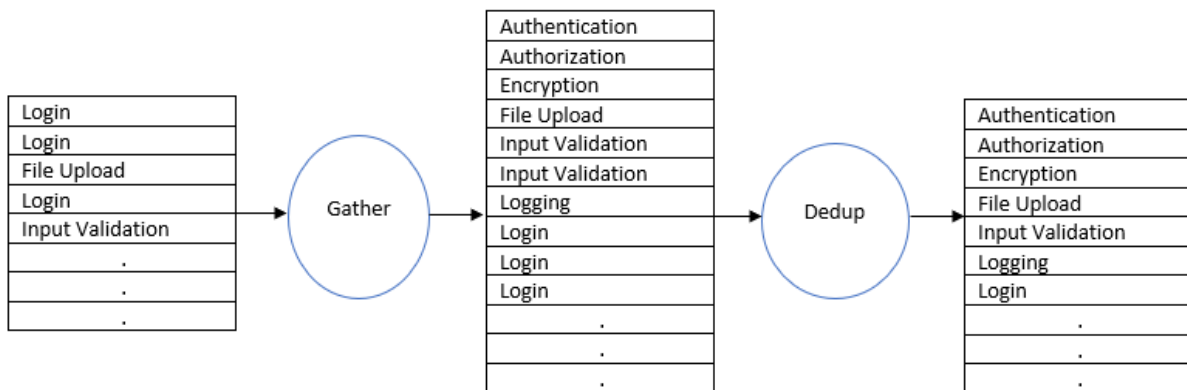


Figure 14: Requirements aggregator

### 3.2.7 Validating the SD-SRE

In order to validate the SD-SRE a portal was created that captures use case scenarios and requirements/specifications in a way that could automatically be parsed into the ML model. Students from a requirements engineering course were asked to participate in a study. The students are presented with a system description and asked to generate the use case scenarios with a focus on security and security requirements/specifications for the system. The use case scenarios are run through the model to suggest the security requirements for the system. The system suggested requirements were compared to the student's requirements. The details of this experiment are further discussed and shown in section 4.4 below. This step is highlighted in Figure 9 above by the solid light blue line (\_\_\_\_\_).

## 3.3 Rationale for Technical Decisions

Knowledge of the proposed system is determined during the requirements elicitation and analysis phase. There are several requirements engineering activities that take place during this phase, (e.g., interviews, use case modeling, scenario development, and prototyping) in order to determine the system's functional requirements. Use case models and scenarios are one of the more complete activities for the elicitation and analysis of requirements as it details the expected uses of the system, the actors that interact with the system, and the system's environment and boundaries. Use case models are further expanded with scenarios that provide more insight into the exact functions of the system. The scenarios explain how the interaction with the system will occur. They describe actions actors, and the system will take. This approach is inspired by attack surface analysis approach to determine system vulnerabilities. Use case scenarios are one of the earliest activities describing the system that has the complete description of the intended interaction between the

actor and the system and therefore are good for analyzing the many ways an attacker could access the system and its data.

Semantic technologies and machine learning can be used to elicit cybersecurity requirements by storing, categorizing, and linking knowledge domains that can be utilized to elicit cybersecurity requirements. The knowledge of systems gathered during the requirements elicitation phase can be leveraged to generate new cybersecurity requirements as new functional features are described. Semantic technologies in combination with machine learning are best used as it easily allows relations between actors, functional features, threats, vulnerabilities, and cybersecurity requirements to be documented. Semantic technologies supports security experts and developers experienced with querying ontologies to ask questions about different functional features and their security requirements while the machine learning model allows less experienced developers to enter their scenarios and get security suggestions. The ontology also allows for quick adaption of security concepts as no longer features become standard practice (such as security questions or SHA-1) and for the addition of new security concepts and functional features.

### 3.4 How Will the SD-SRE Approach be Used?

It is envisioned that developers will use the requirements elicitation SD-SRE portal to conduct elicitation activities and to document their requirements. Within the SD-SRE portal developers can run the security recommender to analyze their use case scenarios to get security requirements. This is the first step for developers in determining the security requirements of their systems. The model can continue to improve as new systems are added, new security recommendations are proposed, and with developer feedback.

## 4 Scenario Driven Security Requirements Elicitation (SD-SRE)

This section details the SD-SRE and elaborates on the sections above detailed in Figure 9. It discusses the overall data collection to support this research, example competency questions and outcomes of the ontology, the performance of the machine learning model, and the validation of the SD-SRE.

### 4.1 Gathering Information

This section discusses the resources gathered for this research. This research required the gathering of use cases and security frameworks, standards, and requirements. Figure 15 below is the highlighted portion of the SD-SRE (shown fully in Figure 9) discussed in this section.

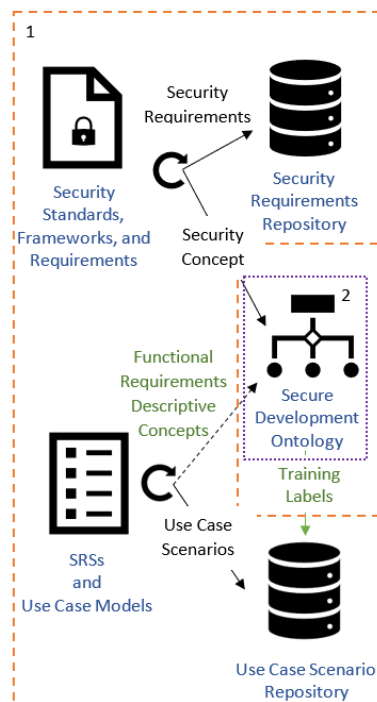


Figure 15: Information gathering for SD-SRE

#### 4.1.1 Use Case Scenarios

Use case scenarios were gathered from requirements engineering courses and fellow students. *Table 4* below shows the breakdown of how many resources were gathered and used. A total of 77 resources were gathered. These resources comprised of requirements specification documents, use case scenarios, and customer interview documentations.

Resources Gathered	Resources Used	Use Case Scenarios	Number of Unique Projects
77	61	1183	8

*Table 4: Use case scenario resources*

Of the 77 resources gathered 61 were usable as they had at least one-use case with accompanying scenarios. The 16 resources not used did not have use case scenarios in the document. The 61 resources represented 8 unique projects which had 1183 use case scenarios. These scenarios were used to train the machine learning models.

#### 4.1.2 Security Concepts and Requirements

The security frameworks, standards and requirements were used to determine generic security requirements and to create the secure development ontology. *Table 5* below shows how many sources were reviewed and how many were used.

Source	Number of Sources Reviewed	Number of Sources Used
CWE	933	3
NIST	9	2
IEEE	6	0
OWASP	102	27
Other	26	1
Total	1076	33

*Table 5: Security concept and requirements resources*

A total of 1076 security related sources were reviewed but only 33 had ideas and concepts related to the requirements stage. The 1076 sources comprises of 933 weaknesses from the MITRE’s Common Weakness Enumeration (CWE) [38]. 9 sources from the National Institute of Standards and Technology (NIST) [60]. 6 sources from Institute of Electrical and Electronics Engineers (IEEE). 102 sources from OWASP [61]. 26 sources from various other security related information distributors.

This resulted in 154 security requirements. The requirements groupings and topics are shown in Table 6 below. All currently suggestable security requirements are listed in the appendix in Table 41.

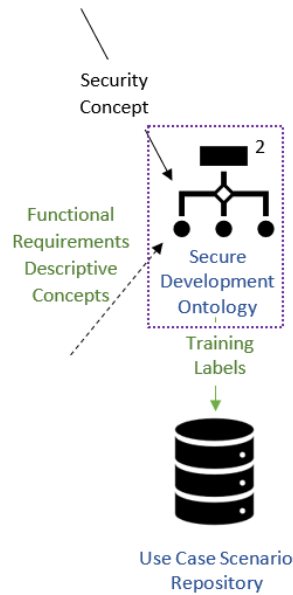
Topic	Number of Requirements
Login	56
Sensitive Information	6
Authentication	7
Authorization	10
Email Collection	7
Input Validation	10
Encryption	18
Random Number	1
Database	16
File Upload	13
Logging	10
Total	154

*Table 6: Security requirements*

## 4.2 Secure Development Ontology

The Secure Development Ontology (SDO) models the secure development of common features, security principles, attackers, vulnerabilities, and mitigations. The SDO has 858 axioms, 178 classes, 29 object properties, 3 data properties, and 159 individuals to describe how entities relate to each other. The ontology serves two purposes: allows users familiar with ontologies the ability to query the ontology for secure development concepts and provides the words that describe a path

in and out of the system used to train the machine learning models. The ontology was provided to the public through GitHub and Ontohub. Figure 16 below is the highlighted portion of the SD-SRE (shown fully in Figure 9) discussed in this section. The ontology comprises of security concepts, functional feature description concepts, and training label terms.



*Figure 16: Ontology use in the SD-SRE*

The most important classes of the ontology are actor, attack, mitigation, security, vulnerability, development life cycle, and secure development. This actor class identifies users that interact with the system and classifies them as malicious or non-malicious users. This class contains 2 subclasses: Malicious Actor and Non-Malicious Actor. The attack class identifies attacks that can occur to a system. This class contains 16 subclasses. The mitigation class identifies mitigations to attacks and vulnerabilities in systems. This class contains 10 sub-classes. The security class identifies the CIA triad. This class contains 3 sub-classes: confidentiality, integrity, and availability. The vulnerabilities class identifies vulnerabilities that can exist in a system. This



class contains 14 sub-classes. The development life cycle stage class identifies the life cycle stage for the consideration of the security concept. This class contains 5 sub-classes. The secure development class identifies common development features and their most secure implementations. This class contains 8 sub-classes.

#### 4.2.1 Security Concepts Competency Questions

An example modeling of an attack from the attack class is shown in Figure 11 below. The example shows the use of rainbow tables to crack hashes to perform a brute force attack.

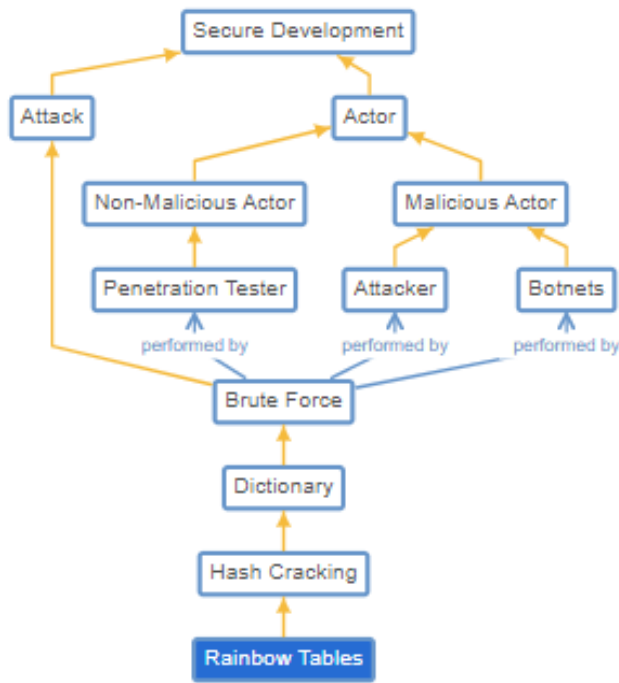


Figure 12: Hash cracking attack modeled in the SDO

The SDO has 29 object properties as shown in Figure 17 below.

- owl:topObjectProperty
- avoid\_use
- based\_on
- comes\_before
- considered
- considered\_during
- described\_by
- determined
- determined\_during
- extends
- identifies
- implemented
- implemented\_after
- implemented\_during
- implements
- is\_a
- mitigates
- performed\_by
- requires
- should\_allow
- should\_be
- should\_have
- should\_not\_be
- should\_use
- stored\_in
- susceptible\_to
- triggers
- verified
- verified\_by
- verified\_during

Figure 17: SDO object properties

The described\_by object property was queried to obtain the labels for the machine learning models. The individuals that describe access control are shown in Figure 18 below. These words were used to label the login machine learning model.

Access Control				
account	handle	log into	new user	user information
accounts	handles	login	new users	username
admin	input username	login	password	username has
administrator	inputs a password	log-in	registers	username is
change password	inputs password	logs in	registration	wrong password
clicks login	inputs username	logs into	sign into	
credentials	log in	logs the user in	signs	
enters password	log in to	new user	signs into	

Figure 18: Access\_control described\_by

The ontology was tested with competency questions. Here we will detail the competency questions surrounding access control as it is the most expansive and complete topic in the ontology.

Concept competency questions for access control:

1. During which phase of the development life cycle is access control determined?
2. What part of the CIA triad does access control support?
3. What vulnerabilities is access control susceptible to?
4. What are the mitigations for the vulnerabilities of access control?

Figure 19 below shows part of the SDO pertaining to access control. Questions 1-3 can be answered with a query that is directly connected to access control as they are direct connections.

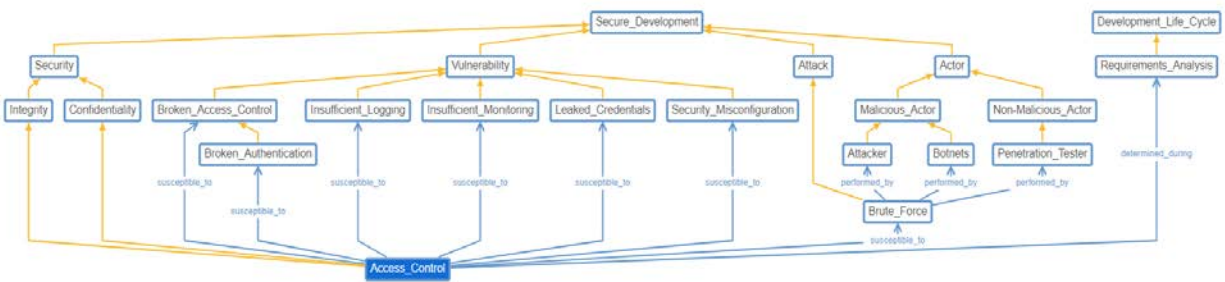


Figure 19: Access\_control Class in the SDO

Table 7 below shows the competency questions and their answer from the ontology.

Competency Question	Object Property	Class
During which phase of the development life cycle is access control determined?	<i>determined_during</i>	Requirements_Analysis
What part of the CIA triad does access control support?	<i>isa</i>	Integrity, Confidentiality

What vulnerabilities is access control susceptible to?	<i>susceptible_to</i>	Broken_Access_Control, Insufficient_Logging, Insufficient_Monitoring, Leaked_Credentials, Security_Misconfiguration, Brute_Force
What are the mitigations for the vulnerabilities of access control?	<i>mitigation</i>	Block_Known_Breached_Credentials Delays Encryption Hash-Based_Message_Authentication_Codes Hashing IP_Address_Restrictions Logging Monitoring Mutlifactor_Authentication Notifications Risk_Based_Restrictions Secure_Storage Soft_Lockout

Table 7: Access\_control class of the SDO

Question 4 is more complicated to query as the mitigations are at least 2 hops from access\_control. Figure 20 below shows an example of one of the mitigations of access control, multifactor authentication, which is a sub-class of authentication and should occur before authorization. This shows an example of how mitigations are not directly connected to access control.

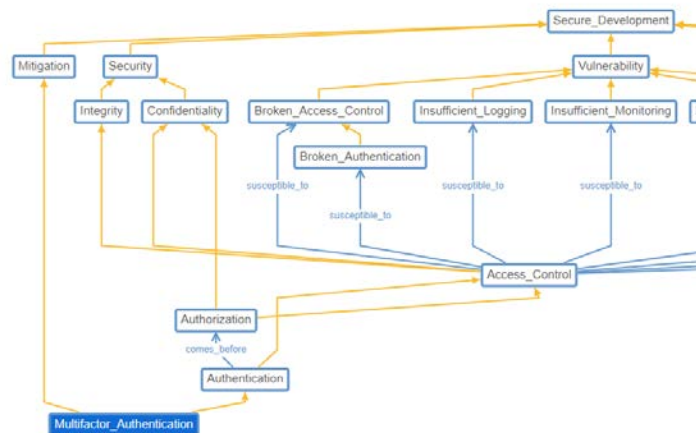


Figure 20: Multifactor\_authentication mitigation example for access control

#### 4.2.2 Use Case Scenarios Parsing

Below are a few use case scenario steps and how they were parsed to identify security and functional feature descriptions. The scenarios were parsed using the attack surface analysis threat modeling method. The scenario steps describe a use case for creating a new user profile.

Step 1: The user enters the username they would like to use.

Step 2: The system returns whether the username is available.

Step 3: The user uploads their profile picture.

Step 4: The user enters the password they would like to use.

Step 5: The user clicks submit.

Step 6: The system returns a verification that a new user profile was created.

From these scenario steps the following keywords were parsed and labeled to identify data paths, actors, and data exchanges. The results of this are shown below in Table 8.

Words/ phrases that identify data path into and out of the system:	Words that identify the actor:	Words and phrases that identify data exchanged terms:
enters (in)	user	username
uploads (in)	system	password
submit (in)		picture
returns (out)		

*Table 8: Parsed used case scenarios classifications*

From the scenario steps above 3 word/phrases were identified to mean that data enters the system. 1 word/phrase was identified to mean that data enters the system. 2 actors were identified. 3 data exchange terms were identified. From the attack surface analysis perspective these scenarios steps would trigger login requirements due to the data exchange of username and password which describes access control, input validation due to the data path of enters and uploads, and file upload due to data path uploads and data exchange of a picture. Attack surface analysis was used to identify and link functional features to security requirements. From parsing the use case scenarios

in the repository 159 unique word/phrases that identified a data path, and 37 unique actors. These 159 unique words were attached to security concepts in the ontology with the described\_by relationship queried to train the machine learning models.

Creating a username and password as well as using that information to access a system is very common in most systems today, but as research continues to show, compromised credentials continues to be the number one attack vector. From the example scenario steps above the approach suggested 4 security requirements for creating a username of which 3 directly apply and 1 should be considered, 24 security requirements for creating and storing a password, and 13 security requirements for input validation. Table 5 below shows the suggested security requirements for creating the UserID/Username.

Requirement Topics	Requirement Sub-Topics	Requirements	Direct Application?
Login	UserID/Username	The system shall only allow case insensitive user IDs.	Yes
Login	UserID/Username	The system shall only allow unique user IDs/Username.	Yes
Login	UserID/Username	For high security systems the system shall assign a secret username that is not based on user's public data.	Consideration
Login	UserID/Username	The system shall not allow sensitive accounts such as system administrators to log in from the front end of the system.	Yes

*Table 9: 4 security requirements for creating a username*

There was a total of 141 requirements identified for the 6 scenario steps of which 32 are direct applications and 109 should be considered. Example requirements that don't directly apply to the 6 scenario steps but should be considered are requirements pertaining to forgotten passwords, changing passwords, login alerts, session management, etc. These suggested requirements tackle common vulnerabilities such as broken access control, using weak hashing

algorithms, broken authentication, insufficient logging and monitoring, security misconfigurations, neglecting input validation, and leaked credentials.

### 4.3 Machine Learning Model

This ML problem is a multi-label problem. In classification models there are two type of problems multi-class which are problems that classify predictions into one of multiple labels vs binary labels and multi-label problems which means something can have multiple labels. For example, a binary class would be classified as 'yes, or 'no' while a multi-class would classify something such as color 'red', 'blue', 'black'. For multi-class problems the thing being classified can only be of one of the multiple classes. Contrary in multi-label problems the thing being classified can have multiple classes. For example: A woman can be a mother, a daughter, a sister, and a grandmother all at the same time. Each of those labels is a class. They can all be true. The problem in this research is a multi-label problem. A use case scenario can trigger login requirements which would trigger authentication, logging, session management, input validation, and encryption.

There were many models considered: Linear SVC, Navies Bayes, Multi-Layer Perceptron classifier. The linear SVC performed better than Navies Bayes and just as well as Multi-Layer Perceptron therefore it was chosen.

Liner SVC has the following hyper parameters: penalty, loss, dual, tol, c, mutli\_class, fit intercept, intercept\_scaling, class\_weight, verbose, random\_state, max\_iter. penalty is the normalization used to penalize the model's prediction. This parameter keeps the model from overfitting the data. The choices are l1 and l2. The default is l2 and this was used for all models. Loss can be 'hinge' or 'squared hinge'. The default is squared hinge and was used for all models. The loss function is often referred to as the cost or error function it represents the cost of a value. tol is the tolerance for stopping criteria which means it tells the model when to stop searching

when the model performance is no longer improving. `c` is the regularization parameter which is the control for penalizing the model's flexibility; `fitting_multi_class` determine the strategy to use for data with more than two classes. This is not the case for this research therefore the default 'ovr' was selected. `fit_intercept` determines where to calculate the intercept if the data is not already centered. `intercept_scaling` is weight attached to the features when fitting the intercept in order to offset penalty. `class_weight` was adjusted to 'balanced' for all models in order to adjust the predictions weights to the inverse proportionality of the frequencies. `verbose` enables a verbose output. `random_state` sets a seed to rerun the same model. `Max_iter` set the maximum number of iterations to run assuming the `tol` is not reached [62].

There are 11 distinct labels identified for this research. The labels group like requirements. The 11 labels are: login, sensitive information, authentication, authorization, email collection, input validation, encryption, random number generation, database, file upload, and logging. Therefore, when a scenario step is processed through the model it is taken through 11 binary class classification models to pull the requirements. Each model started with an 70/30 train/test split then the final model was on a 90/10 split. All the models had unbalanced data therefore accuracy will look very high but should not be relied upon. In this domain it is better for the model to falsely label a scenario step as a false positive (Type I error) vs a false negative (Type II error) because it is best to alert to all the possible security requirements a developer should consider vs not alerting them at all. It is less consequential to suggest a requirement that doesn't pertain to a system than to miss a requirement. The next sections detail the performance of each model. Figure 21 below is the highlighted portion of the SD-SRE (shown fully in Figure 9) discussed in this section. The ML models digest use case scenario steps and suggest the security requirements that should be



pulled from the security requirements repository. The requirements are then aggregated to provide complete and unduplicated security requirements to the user.

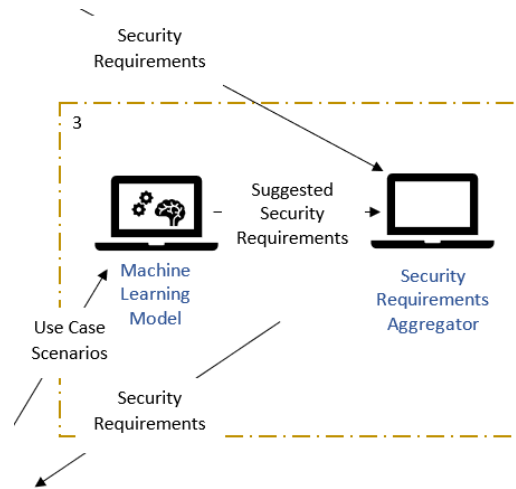


Figure 21: Machine leaning model in the SD-SRE

#### 4.3.1 Login Requirements

The login requirements model predicts whether the system requires security requirements that properly implements a login feature (access control). The data is unbalanced with scenario steps 7623 labeled ‘No’ and 566 labeled ‘Yes’. This model uses 37 terms [login, credentials, registers, username, password, sign in, etc.] to label the model. These terms were also showed in Figure 18 above.

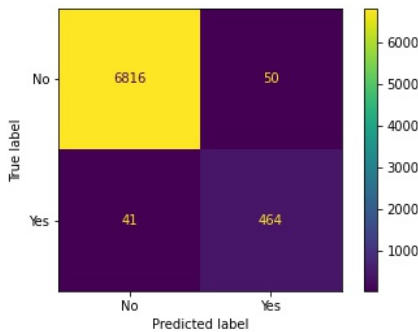


Figure 22: Login requirements model confusion matrix

Figure 22 shows the confusion matrix which shows a type I error of 50 miss classified scenario steps as being a login trigger and 41 as a type II error which is mean the scenario steps did trigger a login requirement but were not properly identified. 6816 scenarios were properly identified as not triggering a login requirement while 464 did properly trigger a login requirement. This is not the full picture the roc curve, precision, recall, and f1-score should also be examined. The accuracy can be examined but should not be emphasized as the data is unbalanced and it will lead to false implications as to how well the model performed.

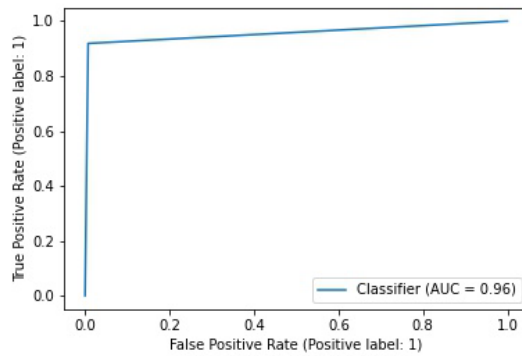


Figure 23: Login requirements model ROC and AUC curve

Figure 23 above is the ROC and AUC curve which plots the true positive rate against the false positive rate. The more curve hugs the upper left-hand corner of the chart the better the performance of the model, but this can also be a false indicator as this can also show that a model is overfitted. The AUC is .96 for this model. It performs very well on the training data.

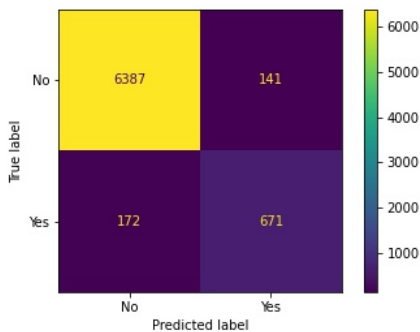
	precision	recall	f1-score	support
No	0.99	0.99	0.99	6866
Yes	0.90	0.92	0.91	505
accuracy			0.99	7371
macro avg	0.95	0.96	0.95	7371
weighted avg	0.99	0.99	0.99	7371

Table 10: Login requirements model performance

Table 10 above shows the precision score, recall, the f1-score and the accuracy. The accuracy is .99 but this is a false indication of model performance as the data is mostly ‘No’ vs ‘Yes’ on triggering login requirements. Precision is the measure of quality of the model and recall is the measure of quantity. The higher the precision the more likely it is for the model to return relevant predictions; the percentage of the predictions that are relevant. The higher the recall the more likely it is for the model to return most of the relevant predictions; the percentage of relevant predictions properly classified. The F1 score displays the balance between precision and recall. This model has a precision of .99 ‘No’ and .90 ‘Yes’, recall of .99 ‘No’ and .92 ‘Yes’, f1 score of .99 ‘No’ and .91 ‘Yes’.

#### 4.3.2 Sensitive Information Requirements

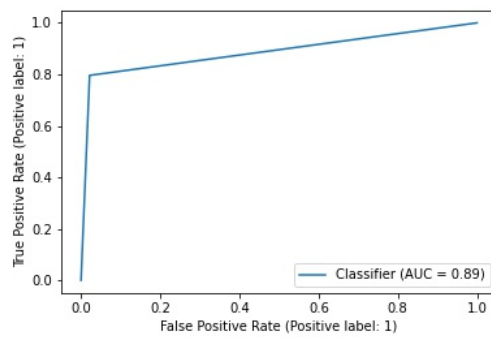
The sensitive information requirements model predicts whether the system requires security requirements that pertain to protecting sensitive data. The data is unbalanced with scenario steps 7247 labeled ‘No’ and 942 labeled ‘Yes’. This model uses 35 terms [social security number, keys, encrypt, name, national id, admin, etc.] to label the model.



*Figure 24: Sensitive information requirements model confusion matrix*

Figure 24 above shows the confusion matrix for the sensitive information label which shows a type I error of 141 miss classified scenario steps and 172 as a type II error which is mean

the scenario steps did trigger a sensitive information requirement but were not properly identified. 6387 scenarios were properly identified as not triggering a sensitive information requirement while 671 did properly trigger a sensitive information requirement. This is not the full picture the roc curve, precision, recall, and f1-score should also be examined. The accuracy can be examined but should not be emphasized as the data is unbalanced and it will lead to false implications as to how well the model performed.



*Figure 25: Sensitive information requirements model ROC and AUC Curve*

Figure 25 above is the ROC and AUC curve which plots the true positive rate against the false positive rate. The more curve hugs the upper left-hand corner of the chart the better the performance of the model, but this can also be a false indicator as this can also show that a model is overfitted. The AUC for this model is .89 which is a moderately well performance.

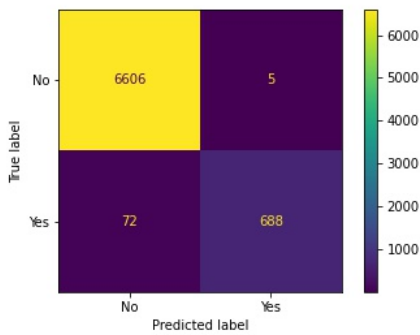
	precision	recall	f1-score	support
No	0.97	0.98	0.98	6528
Yes	0.83	0.80	0.81	843
accuracy			0.96	7371
macro avg	0.90	0.89	0.89	7371
weighted avg	0.96	0.96	0.96	7371

*Table 11: Sensitive information requirements model performance*

Table 11 above shows the precision score, recall, the f1-score and the accuracy. The accuracy is .96 but this is a false indication of model performance as the data is mostly ‘No’ vs ‘Yes’ on triggering sensitive information requirements. This model has a precision of .97 ‘No’ and .83 ‘Yes’, recall of .98 ‘No’ and .80 ‘Yes’, f1 score of .98 ‘No’ and .81 ‘Yes’.

### 4.3.3 Authentication Requirements

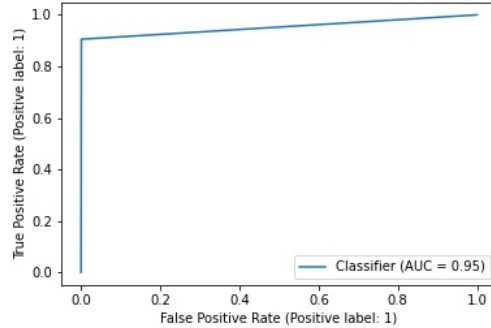
The authentication requirements model predicts whether the system requires security requirements that properly implements a login feature. The data is unbalanced with scenario steps 7342 labeled ‘No’ and 847 labeled ‘Yes’. This model uses 13 terms [access, allows, displays, etc.] to label the model.



*Figure 26: Authentication requirements model confusion matrix*

Figure 26 above shows the confusion matrix for the authentication label which shows a type I error of 5 miss classified scenario steps and 72 as a type II error which is mean the scenario steps did trigger an authentication requirement but were not properly identified. 6606 scenarios were properly identified as not triggering an authentication requirement while 688 did properly trigger an authentication requirement. This is not the full picture the roc curve, precision, recall, and f1-score should also be examined. The accuracy can be examined but should not be

emphasized as the data is unbalanced and it will lead to false implications as to how well the model performed.



*Figure 27: Authentication requirements model ROC and AUC Curve*

Figure 27 above is the ROC and AUC curve which plots the true positive rate against the false positive rate. The more curve hugs the upper left-hand corner of the chart the better the performance of the model, but this can also be a false indicator as this can also show that a model is overfitted. The AUC for this model is .95 which is a very good performance.

	precision	recall	f1-score	support
No	0.99	1.00	0.99	6611
Yes	0.99	0.91	0.95	760
accuracy			0.99	7371
macro avg	0.99	0.95	0.97	7371
weighted avg	0.99	0.99	0.99	7371

*Table 12: Authentication requirements model performance*

Table 12 above shows the precision score, recall, the f1-score and the accuracy. The accuracy is .99 but this is a false indication of model performance as the data is mostly ‘No’ vs

'Yes' on triggering authentication requirements. This model has a precision of .99 'No' and .99 'Yes', recall of 1.00 'No' and .91 'Yes', f1 score of .99 'No' and .95 'Yes'.

#### 4.3.4 Authorization Requirements

The authorization requirements model predicts whether the system requires security requirements that properly implements a login feature. The data is unbalanced with scenario steps 7334 labeled 'No' and 855 labeled 'Yes'. This model uses 15 terms [access, allow, displays, third party, tool, etc.] to label the model.

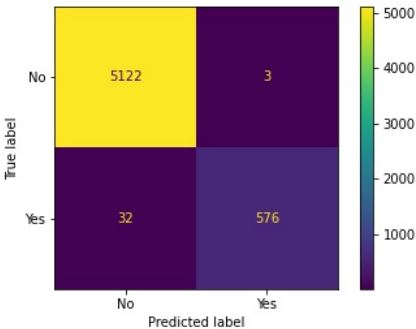
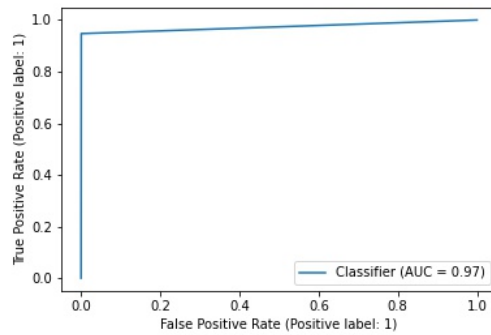


Figure 28: Authorization requirements model confusion matrix

Figure 28 above shows the confusion matrix for the authorization label which shows a type I error of 3 miss classified scenario steps and 32 as a type II error which is mean the scenario steps did trigger an authorization requirement but were not properly identified. 5122 scenarios were properly identified as not triggering an authorization requirement while 572 did properly trigger an authentication requirement. This is not the full picture the roc curve, precision, recall, and f1-score should also be examined. The accuracy can be examined but should not be emphasized as the data is unbalanced and it will lead to false implications as to how well the model performed.



*Figure 29: Authorization requirements model ROC and AUC Curve*

Figure 29 above is the ROC and AUC curve which plots the true positive rate against the false positive rate. The more curve hugs the upper left-hand corner of the chart the better the performance of the model, but this can also be a false indicator as this can also show that a model is overfitted. The AUC for this model is .97 which is a very good performance.

	precision	recall	f1-score	support
No	0.99	1.00	1.00	5125
Yes	0.99	0.95	0.97	608
accuracy			0.99	5733
macro avg	0.99	0.97	0.98	5733
weighted avg	0.99	0.99	0.99	5733

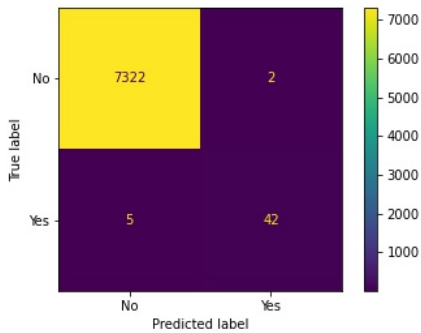
*Table 13: Authorization requirements model performance*

Table 13 above shows the precision score, recall, the f1-score and the accuracy. The accuracy is .99 but this is a false indication of model performance as the data is mostly ‘No’ vs ‘Yes’ on triggering authorization requirements. This model has a precision of .99 ‘No’ and .99 ‘Yes’, recall of 1.00 ‘No’ and .95 ‘Yes’, f1 score of 1.00 ‘No’ and .97 ‘Yes’.



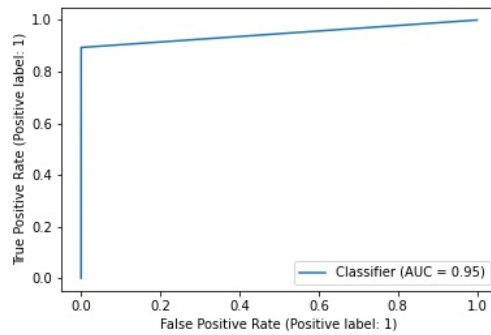
### 4.3.5 Email Collection Requirements

The email collection requirements model predicts whether the system requires security requirements that properly implements a login feature. The data is unbalanced with scenario steps 8137 labeled 'No' and 52 labeled 'Yes'. This model uses 3 terms [email, email address, emails] to label the model.



*Figure 30: Email collection requirements model confusion matrix*

Figure 30 above shows the confusion matrix for the email collection label which shows a type I error of 2 miss classified scenario steps and 5 as a type II error which is mean the scenario steps did trigger an email collection requirement but were not properly identified. 7322 scenarios were properly identified as not triggering an email collection requirement while 42 did properly trigger an email collection requirement. This is not the full picture the roc curve, precision, recall, and f1-score should also be examined. The accuracy can be examined but should not be emphasized as the data is unbalanced and it will lead to false implications as to how well the model performed.



*Figure 31: Email collection requirements model ROC and AUC Curve*

Figure 31 above is the ROC and AUC curve which plots the true positive rate against the false positive rate. The more curve hugs the upper left-hand corner of the chart the better the performance of the model, but this can also be a false indicator as this can also show that a model is overfitted. The AUC for this model is .95 which is a very good performance.

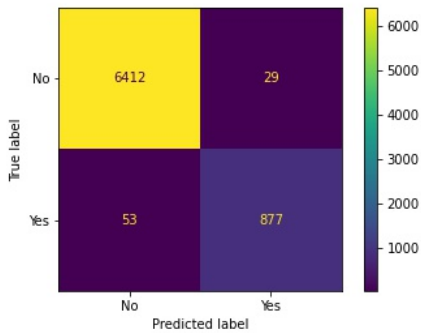
	precision	recall	f1-score	support
No	1.00	1.00	1.00	7324
Yes	0.95	0.89	0.92	47
accuracy			1.00	7371
macro avg	0.98	0.95	0.96	7371
weighted avg	1.00	1.00	1.00	7371

*Table 14: Email collection requirements model performance*

Table 14 above shows the precision score, recall, the f1-score and the accuracy. The accuracy is 1.00 but this is a false indication of model performance as the data is mostly ‘No’ vs ‘Yes’ on triggering email collection requirements. This model has a precision of 1.00 ‘No’ and .95 ‘Yes’, recall of 1.00 ‘No’ and .89 ‘Yes’, f1 score of 1.00 ‘No’ and .99 ‘Yes’.

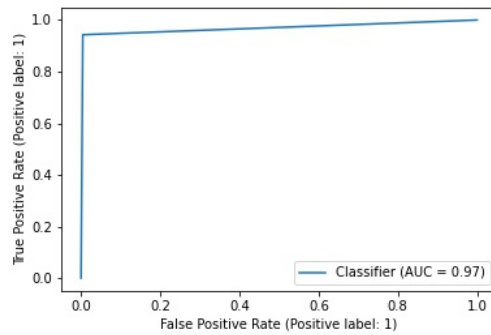
### 4.3.6 Input Validation Requirements

The input validation requirements model predicts whether the system requires security requirements that properly implements a login feature. The data is unbalanced with scenario steps 7154 labeled 'No' and 1035 labeled 'Yes'. This model uses 14 terms [input, enter, enters, edit, edits, text box, etc.] to label the model.



*Figure 32: Input validation requirements model confusion matrix*

Figure 32 above shows the confusion matrix for the input validation label which shows a type I error of 29 miss classified scenario steps and 53 as a type II error which is mean the scenario steps did trigger an input validation requirement but were not properly identified. 6412 scenarios were properly identified as not triggering an input validation requirement while 877 did properly trigger an input validation requirement. This is not the full picture the roc curve, precision, recall, and f1-score should also be examined. The accuracy can be examined but should not be emphasized as the data is unbalanced and it will lead to false implications as to how well the model performed.



*Figure 33: Input validation requirements model ROC and AUC Curve*

Figure 33 above is the ROC and AUC curve which plots the true positive rate against the false positive rate. The more curve hugs the upper left-hand corner of the chart the better the performance of the model, but this can also be a false indicator as this can also show that a model is overfitted. The AUC for this model is .97 which is a very good performance.

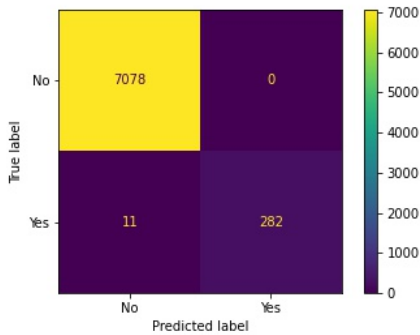
	precision	recall	f1-score	support
No	0.99	1.00	0.99	6441
Yes	0.97	0.94	0.96	930
accuracy			0.99	7371
macro avg	0.98	0.97	0.97	7371
weighted avg	0.99	0.99	0.99	7371

*Table 15:Input validation requirements model performance*

Table 15 above shows the precision score, recall, the f1-score and the accuracy. The accuracy is .96 but this is a false indication of model performance as the data is mostly ‘No’ vs ‘Yes’ on triggering input validation requirements. This model has a precision of .99 ‘No’ and .97 ‘Yes’, recall of 1.0 ‘No’ and .94 ‘Yes’, f1 score of .99 ‘No’ and .96 ‘Yes’.

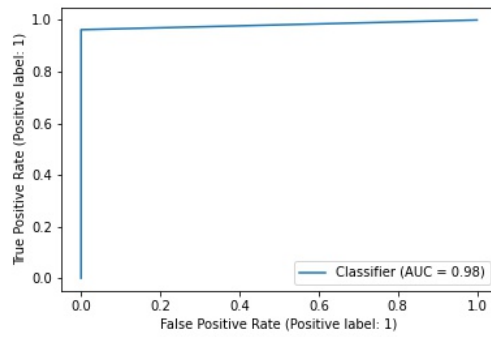
### 4.3.7 Encryption Requirements

The encryption requirements model predicts whether the system requires security requirements that properly implements a login feature. The data is unbalanced with scenario steps 7865 labeled 'No' and 324 labeled 'Yes'. This model uses 9 terms [password, credentials, encrypt, database, etc.] to label the model.



*Figure 34: Encryption requirements model confusion matrix*

Figure 34 above shows the confusion matrix for the encryption label which shows a type I error of 0 miss classified scenario steps and 11 as a type II error which is mean the scenario steps did trigger an encryption requirement but were not properly identified. 7078 scenarios were properly identified as not triggering an encryption requirement while 282 did properly trigger an encryption requirement. This is not the full picture the roc curve, precision, recall, and f1-score should also be examined. The accuracy can be examined but should not be emphasized as the data is unbalanced and it will lead to false implications as to how well the model performed. This model is likely to miss positive identifications based on their being more type II errors vs Type I errors, but not more likely than the other models as the number of Type II errors is low.



*Figure 35: Encryption requirements model ROC and AUC Curve*

Figure 35 above is the ROC and AUC curve which plots the true positive rate against the false positive rate. The more curve hugs the upper left-hand corner of the chart the better the performance of the model, but this can also be a false indicator as this can also show that a model is overfitted. The AUC for this model is .98 which is a very good performance.

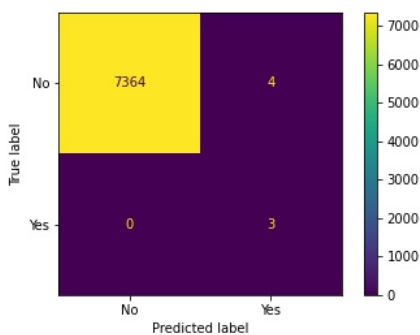
	precision	recall	f1-score	support
No	1.00	1.00	1.00	7078
Yes	1.00	0.96	0.98	293
accuracy			1.00	7371
macro avg	1.00	0.98	0.99	7371
weighted avg	1.00	1.00	1.00	7371

*Table 16: Encryption requirements model performance*

Table 16 above shows the precision score, recall, the f1-score and the accuracy. The accuracy is 1.00 but this is a false indication of model performance as the data is mostly ‘No’ vs ‘Yes’ on triggering encryption requirements. This model has a precision of 1.00 ‘No’ and 1.00 ‘Yes’, recall of 1.00 ‘No’ and .96 ‘Yes’, f1 score of 1.00 ‘No’ and .98 ‘Yes’.

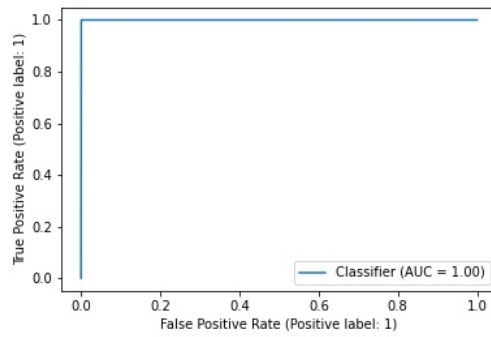
#### 4.3.8 Random Number Generator Requirements

The random number generator requirements model predicts whether the system requires security requirements that properly implements a login feature. The data is unbalanced with scenario steps 8185 labeled 'No' and 4 labeled 'Yes'. This model uses 7 terms [random, random number, randomize, randomly, etc.] to label the model. This model is worrisome as there were only 4 samples in the data set that trigger random numbers.



*Figure 36: Random number requirements model confusion matrix*

Figure 36 above shows the confusion matrix for the random number generator label which shows a type I error of 4 miss classified scenario steps and 0 as a type II error which is mean the scenario steps did trigger a random number requirement but were not properly identified. 7364 scenarios were properly identified as not triggering a random number requirement while 3 did properly trigger a random number requirement. This is not the full picture the roc curve, precision, recall, and f1-score should also be examined. The accuracy can be examined but should not be emphasized as the data is unbalanced and it will lead to false implications as to how well the model performed.



*Figure 37: Random number requirements model ROC and AUC Curve*

Figure 37 above is the ROC and AUC curve which plots the true positive rate against the false positive rate. The more curve hugs the upper left-hand corner of the chart the better the performance of the model, but this can also be a false indicator as this can also show that a model is overfitted. The AUC for this model is 1.0 which is a very well performing model, but this is unfortunately not a good indicator for this model as there were only 4 samples in the dataset.

	precision	recall	f1-score	support
No	1.00	1.00	1.00	7368
Yes	0.43	1.00	0.60	3
accuracy			1.00	7371
macro avg	0.71	1.00	0.80	7371
weighted avg	1.00	1.00	1.00	7371

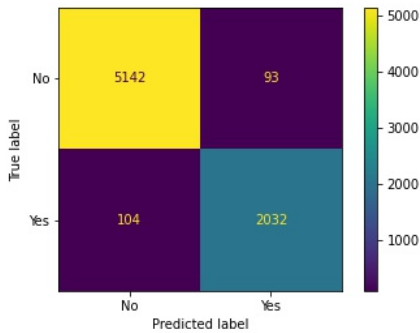
*Table 17: Random number requirements model performance*

Table 17 above shows the precision score, recall, the f1-score and the accuracy. The accuracy is 1.00 but this is a false indication of model performance as the data is mostly ‘No’ vs ‘Yes’ on triggering random number requirements. This model has a precision of 1.00 ‘No’ and .43 ‘Yes’, recall of 1.00 ‘No’ and 1.00 ‘Yes’, f1 score of 1.00 ‘No’ and .60 ‘Yes’.



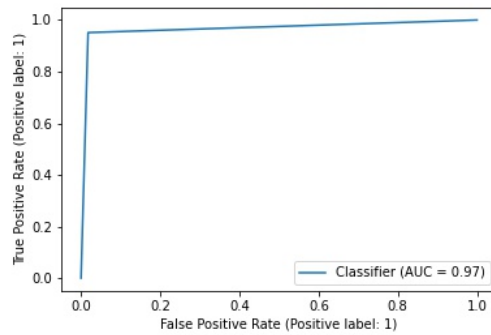
### 4.3.9 Database Requirements

The login requirements model predicts whether the system requires security requirements that properly implements a login feature. The data is unbalanced with scenario steps 5834 labeled ‘No’ and 2355 labeled ‘Yes’. This model uses 27 terms [database, store, save, selects, etc.] to label the model.



*Figure 38: Database requirements model confusion matrix*

Figure 38 above shows the confusion matrix for the database label which shows a type I error of 93 miss classified scenario steps and 104 as a type II error which is mean the scenario steps did trigger a database requirement but were not properly identified. 5142 scenarios were properly identified as not triggering a database requirement while 2032 did properly trigger a database requirement. This is not the full picture the roc curve, precision, recall, and f1-score should also be examined. The accuracy can be examined but should not be emphasized as the data is unbalanced and it will lead to false implications as to how well the model performed.



*Figure 39: Database requirements model ROC and AUC Curve*

Figure 39 above is the ROC and AUC curve which plots the true positive rate against the false positive rate. The more curve hugs the upper left-hand corner of the chart the better the performance of the model, but this can also be a false indicator as this can also show that a model is overfitted. The AUC for this model is .97 which is a very good performance.

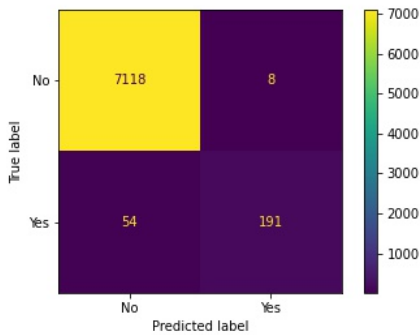
	precision	recall	f1-score	support
No	0.98	0.98	0.98	5235
Yes	0.96	0.95	0.95	2136
accuracy			0.97	7371
macro avg	0.97	0.97	0.97	7371
weighted avg	0.97	0.97	0.97	7371

*Table 18: Database requirements model performance*

Table 18 above shows the precision score, recall, the f1-score and the accuracy. The accuracy is .97 but this is a false indication of model performance as the data is mostly ‘No’ vs ‘Yes’ on triggering database requirements. This model has a precision of .98 ‘No’ and .96 ‘Yes’, recall of .98 ‘No’ and .95 ‘Yes’, f1 score of .98 ‘No’ and .95 ‘Yes’.

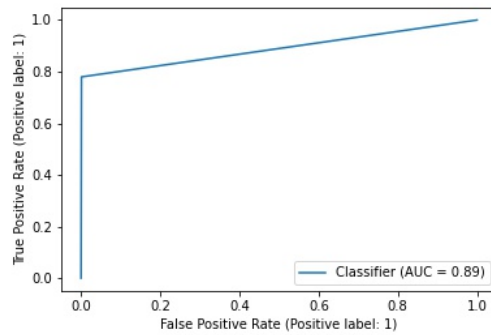
#### 4.3.10 File Upload Requirements

The file upload requirements model predicts whether the system requires security requirements that properly implements a file upload feature. The data is unbalanced with scenario steps 7915 labeled 'No' and 274 labeled 'Yes'. This model uses 23 terms [upload, image, pdf, video, song, etc.] to label the model.



*Figure 40: File upload requirements model confusion matrix*

Figure 40 above shows the confusion matrix for the file upload label which shows a type I error of 8 miss classified scenario steps and 54 as a type II error which is mean the scenario steps did trigger a file upload requirement but were not properly identified. 7118 scenarios were properly identified as not triggering a file upload requirement while 191 did properly trigger a file upload requirement. This is not the full picture the roc curve, precision, recall, and f1-score should also be examined. The accuracy can be examined but should not be emphasized as the data is unbalanced and it will lead to false implications as to how well the model performed.



*Figure 41: File upload requirements model ROC and AUC Curve*

Figure 41 above is the ROC and AUC curve which plots the true positive rate against the false positive rate. The more curve hugs the upper left-hand corner of the chart the better the performance of the model, but this can also be a false indicator as this can also show that a model is overfitted. The AUC for this model is .89 which is a moderately good performance.

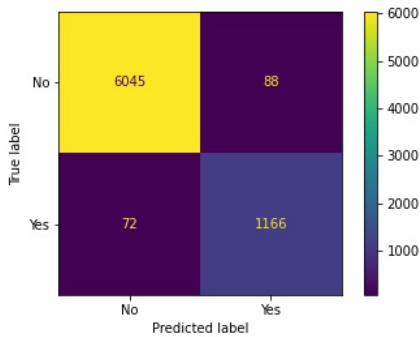
	precision	recall	f1-score	support
No	0.99	1.00	1.00	7126
Yes	0.96	0.78	0.86	245
accuracy			0.99	7371
macro avg	0.98	0.89	0.93	7371
weighted avg	0.99	0.99	0.99	7371

*Table 19: File upload requirements model performance*

Table 19 above shows the precision score, recall, the f1-score and the accuracy. The accuracy is .99 but this is a false indication of model performance as the data is mostly ‘No’ vs ‘Yes’ on triggering file upload requirements. This model has a precision of .99 ‘No’ and .96 ‘Yes’, recall of 1.00 ‘No’ and .78 ‘Yes’, f1 score of 1.00 ‘No’ and .86 ‘Yes’.

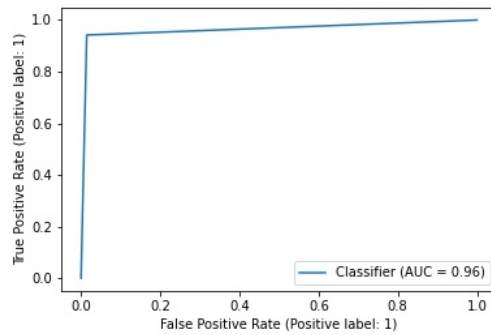
### 4.3.11 Logging Requirements

The logging requirements model predicts whether the system requires security requirements that properly implements a logging feature. The data is unbalanced with scenario steps 6798 labeled 'No' and 1391 labeled 'Yes'. This model uses 16 terms [error, errors, click, invalid, notify, notifies, etc.] to label the model.



*Figure 42: Logging requirements model confusion matrix*

Figure 42 above shows the confusion matrix for the logging label which shows a type I error of 88 miss classified scenario steps and 72 as a type II error which is mean the scenario steps did trigger a logging requirement but were not properly identified. 6045 scenarios were properly identified as not triggering a logging requirement while 1166 did properly trigger a logging requirement. This is not the full picture the roc curve, precision, recall, and f1-score should also be examined. The accuracy can be examined but should not be emphasized as the data is unbalanced and it will lead to false implications as to how well the model performed.



*Figure 43: Logging requirements model ROC and AUC Curve*

Figure 43 above is the ROC and AUC curve which plots the true positive rate against the false positive rate. The more curve hugs the upper left-hand corner of the chart the better the performance of the model, but this can also be a false indicator as this can also show that a model is overfitted. The AUC for this model is .96 which is a moderately good performance.

	precision	recall	f1-score	support
No	0.99	0.99	0.99	6133
Yes	0.93	0.94	0.94	1238
accuracy			0.98	7371
macro avg	0.96	0.96	0.96	7371
weighted avg	0.98	0.98	0.98	7371

*Table 20: Logging requirements model performance*

Table 20 above shows the precision score, recall, the f1-score and the accuracy. The accuracy is .98 but this is a false indication of model performance as the data is mostly ‘No’ vs ‘Yes’ on triggering logging requirements. This model has a precision of .99 ‘No’ and .93 ‘Yes’, recall of .99 ‘No’ and .94 ‘Yes’, f1 score of .99 ‘No’ and .94 ‘Yes’.

#### 4.3.12 Weakness

This section describes the weaknesses of this approach.

##### *4.3.12.1 Weaknesses of the model*

There are a few weaknesses of the model and the Ontology. The use case scenarios used in this research is student data. These are relatively new developers learning how to work with these models. The scope of the use cases used were limited to the project's students were assigned. A lot of the use cases came from past requirements engineering courses, so the students were assigned the same project for the semester. This is both an advantage and disadvantage. It is an advantage to see if the model is assigning the same set of requirements to the same project. There are cases in which the model did not and on further inspection it is because the students did not describe that feature in the use case scenarios. It is a disadvantage as there wasn't much variety in project domain scope. There are many more label identifying terms left to be discovered that were not represented in the dataset. These labels in the future could be missed as the model has never encountered them before. This is a concern, but this approach is made to evolve with the times. In addition, the multi-layer approach to this problem by analyzing the scenarios at the individual steps helps duplicate the likeliness to trigger a security requirement.

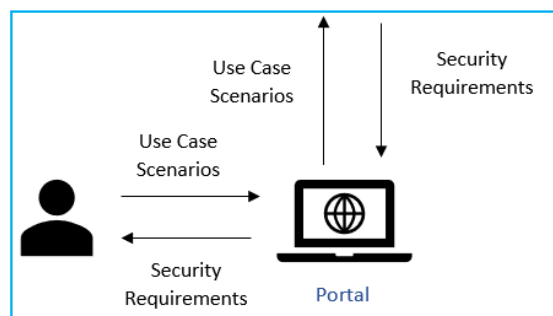
##### *4.3.12.2 Weaknesses of the approach*

This approach is weak in addressing the security of Internet of Things (IOT) devices. The security requirements for this project can be applied to IOT devices but with the limited computing power of some of these devices make implement unfeasible. This approach is also limited on physical security requirements as many of the security recommendations for physical security were mostly referenced at the design level.

## 4.4 Validation

The efficacy of this approach was tested with volunteer students from a hybrid systems and software engineering graduate requirements engineering course. The students were presented with a fictitious product description for the DirectCoins app as shown in Figure 45 below. The DirectCoins app describe a money transfer mobile and web app. There are six key features of the app: create an account, send money, receive money, add money, cash out money, and view transactions.

Students were asked to provide the use case scenarios with a focus on security and the security requirements for this system. Figure 44 below is the highlighted portion of the SD-SRE (shown fully in Figure 9) discussed in this section.



*Figure 44: Portal interface of the SD-SRE for validation*

Students were split into two groups. One group of students used their own approach to develop the use case scenarios and security requirements while the other group used SD-SRE portal. In addition both groups of students were asked to track the amount of time they worked on the activity. The use case scenarios were directly entered into the security development SD-SRE portal by the students who were assigned the portal. The SD-SRE portal is designed to provide the security requirements back to the user but this was disabled for this activity in order to compare the security requirements produced by the students and the requirements produced by the system.



Pictures of the SD-SRE portal are included in the appendix. The students were given five days to complete the use cases and requirements. All the use cases were then run through the security requirements recommender for comparison. The quantity and quality of the requirements were compared between the systems recommendations and those provided by the students.

Assigned Team Name: \_\_\_\_\_

Portal URL:

The DirectCoins App is a free mobile and web-based application that allows users to send money to friends and to pay merchants quickly and securely. A user must be able to do all functions of the app through both their mobile devices and a web browser.

Users will need the ability to do the following on the app:

1. **Create an Account.** Users will be able to create custom handles which will serve as their Username. In order to create an account users will need to provide their legal name, phone number, email address, mailing address, date of birth, city of birth, country of birth, national id number (you can assume social security number for this project), a picture of the front and back of a government issued ID ( state ID, passport, Driver's license) , and upload a pdf file of a check for direct deposit. Users will also have to select their account type: general user or merchant. Direct deposit set up will be handled manually within 5 business days. The client already has a solution for direct deposit if they can collect the requested information. If the documents are not correct the client will contact the user to collect the correct documents.
2. **Send Money.** Users will be able to send money to friends and merchants by searching for their handle or scanning their QR code. The transaction must capture from whom the money coming, who it is going to, how much, and why. A user cannot cancel a send money transaction once submitted. There is a daily total limit of how much money can be sent: \$9,999 dollars a day and no more than \$2,500 to any unique user a day. A user can not send more money than they currently have in their account.
3. **Receive Money.** The user will see from whom the payment came, how much, and the reason why the money was sent.
4. **Add Money.** Users will be able to add money to their account by linking their bank account.
5. **Cash Out Money.** Users will be able to cash out through direct deposit which is established within 5 business days of account creation.
6. **View Transactions.** Users need the ability to see all their transactions for the past 36 months. The user should be able to filter via: month, year, transaction type (sent, pending, received), amount, and users.

Security is of great importance to the client considering the type of information and assets involved with this product. Please focus on security.

*Figure 45: DirectCoins app*

#### 4.4.1 Student Educational Backgrounds

A total of 19 students volunteered to participate in the study. Nine were assigned the portal while ten were not. Of the nine assigned the portal six entered data into the portal. Of the six, five had use case scenario steps and requirements and four were complete enough to describe the key functions of the DirectCoins app. Of the ten not assigned the portal six returned the activity with five being complete. All students had Science, Technology, Engineering, and Math (STEM) undergraduate majors. Their graduate degrees were a majority Software Engineering, Systems Engineering, and Unmanned Systems Engineering. Table 21 and Table 22 below shows the breakdown of student majors for all student who agreed volunteered for this study.

Undergraduate Major	# of students who submitted the activity	Total # of students	Graduate Major	# of students who submitted the activity	Total # of students
Computer Science	3	4	Human Factors	1	1
Mechanical Engineering	1	1	Software Engineering	3	4
Meteorology	0	1	Systems Engineering	0	2
Psychology	1	1	Unmanned and Autonomous Systems Engineering	2	2
Spaceflight Operations	0	1			
Unmanned Aircraft Systems Science	1	1			
Total	6	9	Total	6	9

*Table 21: Volunteer students educational background of portal users*

Undergraduate Major	# of students who submitted the activity	Total # of students	Graduate Major	# of students who submitted the activity	Total # of students
Aerospace Engineering	2	2	Electrical Engineering and Computer Science	1	1
Computer Engineering	0	1	Software Engineering	3	4
Human Factors	0	1	Systems Engineering	2	4
Physics	0	1	Unmanned and Autonomous Systems Engineering	0	1
Software Engineering	2	3			
Systems Engineering	1	1			
Unmanned Aircraft Systems	1	1			
Total	6	10	Total	6	10

*Table 22: Volunteer students' educational background of non-portal users*

#### 4.4.2 Student Results

Six of the nine students assigned the portal submitted something in the portal of which one was unusable as it was just a project description and no use case scenarios. Of the five remaining three were complete and two had completed use cases but not enough to describe the whole system. The one unusable was removed from the analysis the partial use cases were kept for analysis. Table 23 below shows the use case, requirements, and time it took for students using the portal.

Project Submissions	Number of Student Submitted Scenarios	Number of Scenario Steps	Time Students Took to Complete Scenarios	Number of Student Submitted Requirements	Time Students Took to Complete Requirements
Portal1	7	44	70	25	64
Portal2	6	24	42	10	32
Portal3	7	15	23	13	19
Portal4	6	41	98	21	36
Portal5	7	33	42	19	21
Total	33	157	275	88	172
Mean	6.6	31.4	55	17.6	34.4
Median	7	33	42	19	32
Mode	7	N/A	42	N/A	N/A

*Table 23: Portal students submissions*

Six of the ten students not assigned the portal submitted something. one was unusable for suggesting security requirements as it was just a use case scenarios description and no steps. Of the five remaining four were complete and one had completed use cases but not enough to describe the whole system. The one unusable project was removed from the analysis for generating security requirements, but the student security requirements were kept. The other partial use cases were kept for analysis.

Project Submissions	Number of Student Submitted Scenarios	Number of Scenario Steps	Time Students Took to Complete Scenarios	Number of Student Submitted Requirements	Time Students Took to Complete Requirements
NoPortal1	14	186	260	32	30
NoPortal2	2	66	59	0*	0*
NoPortal3	6	62	83	7	15
NoPortal4	6	90	40	13	30
NoPortal5	11	115	140	20	30
NoPortal6	9* <sup>4</sup>	0*	35*	22	60
Total	39	519	582	94	165
Mean	7.8	103.8	116.4	18.8	33
Median	6	90	83	20	30
Mode	6	N/A	N/A	N/A	30

*Table 24: Student submissions not assigned the portal*

<sup>4</sup> \* means not used in the analysis due to lack of data.

Table 24 above shows the use case, requirements, and time it took for students not using the portal.

Table 25 below shows the statistics for the whole validation set. There were a total of 10 projects used for this study.

	Number of Student Submitted Scenarios	Number of Scenario Steps	Time Students Took to Complete Scenarios	Number of Student Submitted Requirements	Time Students Took to Complete Requirements
Total	72	676	857	168	337
Minimum	2	15	23	7	15
Maximum	14	186	260	25	64
Mean	7.2	67.6	85.7	16.9	33.7
Median	6.5	53	64.5	19	30
Mode	6	N/A	42	13,19	30

*Table 25: All student submission- the full validation set*

#### 4.4.3 Results of the Suggested Requirements

This next section shows the results by requirements topic. All of the possible suggestable requirements are in the appendix in Table 41. The number of steps classified as true are taken from seeing if the model returned a yes or a no for the suggested requirement topic for the step. This is compared to the labeling of the scenario steps as done to train the model by using the describe\_by terms in the ontology. The scenario steps are labeled after the suggested security requirements were obtained.

Table 26 below shows the results of parsing the projects through the SD-SRE for the login suggested requirements.

Project submissions	Login	# of steps classified as true	# of steps expected to be classified as true	Student login requirements	# of directly related requirements	# of requirements
Portal1	Yes	4	12	8	0	56
Portal2	Yes	4	2	2	1	56
Portal3	Yes	2	2	3	0	56
Portal4	Yes	18	18	4	1	56
Portal5	Yes	8	23	5	1	56
NoPortal1	Yes	86	32	11	0	56
NoPortal2	Yes	44	29	N/A	1	56
NoPortal3	Yes	12	15	3	0	56
NoPortal4	Yes	1	12	6	0	56
NoPortal5	Yes	30	36	5	1	56
NoPortal6	N/A	N/A	N/A	9	N/A	N/A

*Table 26: Validation results for login requirements*

While analyzing the scenario step for this particular topic it is discovered that descriptions entered by NoPortal4 barely acknowledge a login was needed. The use cases referred to the user/username as handle as stated in the project description and this is was used by some of the student submissions. This was not a case encountered in the test and train data prior to this validation however it was identified in the ontology for the describe\_by link in access control.

Scenario Name	Step Type	Step Description
Create custom handles	Flow	User inputs a custom handle
Create custom handles	Flow	If the handle is taken, step 2a is run
Create custom handles	Flow	The handle is saved
Create custom handles	Flow	End of scenario
Create custom handles	Alternate Flow	The user is given an error message indicating for them to try again. Return to step 1
Create account	Flow	The user fills out the form, providing legal name, phone number, email address, mailing address, date of birth, city of birth, country of birth, national id number.
Create account	Flow	The user uploads documents into the form for pictures of the government id and check.
Create account	Flow	The user selects account type, either user or merchant.
Create account	Flow	The user submits the form

Create account	Flow	The form is reviewed by the client
Create account	Flow	If a form problem is detected, 6a is run.
Create account	Flow	The form is processed, and the account is created.
Create account	Flow	End of scenario
Create account	Alternate Flow	The client contacts the user to collect the correct documents. Return to step 2

*Table 27: Portal1 Login related Use Cases*

Portal1 had no directly relatable login security requirements because the provided scenario step descriptions did not have a directly relatable security requirement. The scenario steps are displayed above in Table 27. A relatable requirement would be “The user enters their password”, or the “The user enters their username”. Instead Portal1 used the term “handle” to identify the user. This would relate to 30 login password related requirements. Portal1 did trigger the login requirement because it identified different account types between user and merchant. This model can be improved training the machine learning model on the “handle” example as a substitute for username. The word handle was captured in the ontology for the describe\_by for access control as shown in Figure 18 above.

Table 28 shows an example of Portal3 scenario steps that were tagged for triggering a login requirement vs those that were not. As shown below four scenario steps were tagged but only two are directly related to login.

Use Case Name	Alternate Step	Scenario Step Description	Portal	Login_prediction
Create Account	Flow	Allows user to register with DirectCoins	Yes	Yes
Create Account	Alternate Flow	Enter personal information	Yes	Yes
Create Account	Alternate Flow	Enter payment type (credit/wire transfer)	Yes	Yes
Verify User	Flow	DirectCoins App verifies the user logging in to their account	Yes	Yes

*Table 28: Example of Portal3 login Requirement Analysis*

Table 29 below shows the results of parsing the projects through the SD-SRE for the sensitive information suggested requirements.

Project submissions	Sensitive information	# of steps classified as true	# of steps expected to be classified as true	Student sensitive information requirement	# of directly related requirements	# of requirements
Portal1	Yes	1	7	0	0	6
Portal2	Yes	5	2	0	0	6
Portal3	No	0	0	0	0	6
Portal4	No	0	0	0	0	6
Portal5	Yes	10	9	0	0	6
NoPortal1	Yes	9	2	0	0	6
NoPortal2	Yes	66	49	N/A	N/A	6
NoPortal3	Yes	17	13	0	0	6
NoPortal4	Yes	2	8	0	0	6
NoPortal5	Yes	11	10	1	0	6
NoPortal6	N/A	N/A	N/A	0	N/A	N/A

*Table 29: Validation results for sensitive information requirements*

Portal3 and Portal4 did not have any information sensitivity indicators in its use cases. Portal3 simply stated the “Enter personal information”. Portal4 states the “The systems prompts user for ID number” in reference to collecting social security/national ID. Although Portal3 and Portal4 did not trigger any sensitive information requirements the system returns it them any how because the system ping them for login requirements which are extended by sensitive information requirements. This model can be improved by allowing users to tag data sensitivity as they would tag the sensitivity of actors; the sensitivity of data an actor has when interacting with the system.

Table 30 below shows the results of parsing the projects through the SD-SRE for the authentication suggested requirements.



Project submissions	Authentication	# of steps classified as true	# of steps expected to be classified as true	Student authentication requirements	# of directly related requirements	# of requirements
Portal1	No	0	0	4	0	7
Portal2	No	0	0	1	0	7
Portal3	No	0	0	1	0	7
Portal4	No	0	0	4	0	7
Portal5	No	0	0	3	0	7
NoPortal1	No	0	0	6	0	7
NoPortal2	No	0	0	N/A	N/A	7
NoPortal3	No	0	0	1	0	7
NoPortal4	No	0	0	3	0	7
NoPortal5	No	0	0	4	0	7
NoPortal6	N/A	N/A	N/A	3	N/A	N/A

*Table 30: Validation results for authentication requirements*

There were no direct authentication requirement triggers in the use case scenario description, but the requirements were triggered due to the pining of login, and database requirements.

Table 31 below shows the results of parsing the projects through the SD-SRE for the authorization suggested requirements.

Project submissions	Authorization	# of steps classified as true	# of steps expected to be classified as true	Student authorization requirements	# of directly related requirements	# of requirements
Portal1	No	0	0	0	0	10
Portal2	Yes	5	3	3	0	10
Portal3	No	0	0	3	0	10
Portal4	No	0	0	4	0	10
Portal5	No	0	0	4	0	10
NoPortal1	No	0	0	0	0	10
NoPortal2	No	0	0	N/A	N/A	10
NoPortal3	Yes	12	7	1	0	10
NoPortal4	No	0	0	2	0	10
NoPortal5	No	0	0	3	0	10
NoPortal6	N/A	N/A	N/A	3	N/A	N/A

*Table 31: Validation results for authorization requirements*

There were a few direct authorization requirement triggers in the use case scenario description, but the requirements were also triggered due to the pinning of login, and database requirements.

Table 32 below shows the results of parsing the projects through the SD-SRE for the email collection suggested requirements.

Project submissions	Email Collection	# of steps classified as true	# of steps expected to be classified as true	Student email collection requirements	# of directly related requirements	# of requirements
Portal1	No	0	1	1	0	0
Portal2	No	0	0	0	0	0
Portal3	No	0	0	0	0	0
Portal4	Yes	5	5	0	4	7
Portal5	No	0	2	0	0	0
NoPortal1	Yes	3	3	0	4	7
NoPortal2	Yes	5	5	N/A	N/A	7
NoPortal3	No	0	0	0	0	0
NoPortal4	Yes	2	2	0	4	7
NoPortal5	No	0	0	0	0	0
NoPortal6	N/A	N/A	N/A	0	N/A	N/A

*Table 32: Validation results for email collection requirements*

The email collection did not perform as well as desired. The requirements were missed for two systems that had email use case scenario descriptions. These descriptions were low with the most being five for one project. There were no other requirements that would also trigger this requirement. These set of requirements are specific to email collection and would not be applicable in the overall input validation set of requirements. This model can be improved by getting more data with email descriptions.

Table 33 below shows the results of parsing the projects through the SD-SRE for the input validation suggested requirements.

Project submissions	Input validation	# of steps classified as true	# of steps expected to be classified as true	Student input validation Requirements	# of directly related requirements	# of requirements
Portal1	Yes	3	5	0	0	10
Portal2	Yes	5	4	0	0	10
Portal3	No	0	2	0	0	10
Portal4	Yes	8	8	0	0	10
Portal5	Yes	7	7	0	0	10
NoPortal1	Yes	26	16	0	0	10
NoPortal2	Yes	22	26	N/A	N/A	10
NoPortal3	Yes	6	10	0	0	10
NoPortal4	Yes	7	9	1	0	10
NoPortal5	Yes	3	11	0	0	10
NoPortal6	N/A	N/A	N/A	0	N/A	N/A

*Table 33: Validation results for input validation requirements*

Most of the projects except Portal3 triggered the input validation requirements. Portal3 should have done so based on two steps but did not. It was triggered however for Portal3 due to login requirements. If Portal3 had triggered email collection and file upload requirements those would have also triggered the input validation requirements but that was not the case for Portal3 as it did not have requirements that triggered either. Portal1 had a lot of false negatives because the use case identifies when a user does not take the action of entering data which was not expected. NoPortal6 uses the word prompt and create for identifying users entering data which was not accounted for in the test and train set. This model can be improved by looking for when users are prompted and when users are creating.

Table 34 below shows the results of parsing the projects through the SD-SRE for the encryption suggested requirements.

Project submissions	Encryption	# of steps classified as true	# of steps expected to be classified as true	Student encryption requirements	# of directly related requirements	# of requirements
Portal1	No	0	1	0	0	18
Portal2	No	0	0	1	0	18
Portal3	No	0	0	0	0	18
Portal4	No	0	0	0	0	18
Portal5	Yes	5	7	0	0	18
NoPortal1	No	0	0	2	0	18
NoPortal2	No	0	0	N/A	N/A	18
NoPortal3	Yes	2	3	0	0	18
NoPortal4	No	0	2	0	0	18
NoPortal5	No	0	0	1	0	18
NoPortal6	N/A	N/A	N/A	0	N/A	N/A

*Table 34: Validation results for encryption requirements*

Encryption is automatically triggered by login, sensitive information, and database.

Although it did not have many steps trigger it, it was still pulled for all projects.

The random number requirement table is not added as no scenario steps required a random number directly. The random number scenario was return for all projects that ping login, sensitive information, encryption, or database as they all would likely require a random key or token.

Table 35 below shows the results of parsing the projects through the SD-SRE for the database suggested requirements.

Project submissions	Database	# of steps classified as true	# of steps expected to be classified as true	Student database Requirements	# of directly related requirements	# of requirements
Portal1	Yes	3	10	0	0	16
Portal2	Yes	5	9	1	0	16
Portal3	Yes	2	4	0	0	16
Portal4	Yes	9	13	0	0	16
Portal5	Yes	6	14	0	0	16
NoPortal1	Yes	31	37	0	0	16

NoPortal2	Yes	23	19	N/A	N/A	16
NoPortal3	Yes	6	11	1	0	16
NoPortal4	Yes	9	15	0	0	16
NoPortal5	Yes	2	17	0	0	16
NoPortal6	N/A	N/A	N/A	0	N/A	N/A

*Table 35: Validation results for database requirements*

Database is triggered for a lot a steps. Any step that suggest making a choice, entering data, or displaying data should trigger a database step in addition to login, input validation, email collection, file upload, and logging requirements.

Table 36 below shows the results of parsing the projects through the SD-SRE for the file upload suggested requirements.

Project submissions	File upload	# of steps classified as true	# of steps expected to be classified as true	Student file upload Requirements	# of directly related requirements	# of requirements
Portal1	No	0	1	0	0	0
Portal2	No	0	0	0	0	0
Portal3	No	0	0	0	0	0
Portal4	No	0	2	0	0	0
Portal5	No	0	2	0	0	0
NoPortal1	Yes	25	4	1	1	13
NoPortal2	No	0	3	N/A	N/A	0
NoPortal3	No	0	1	0	0	0
NoPortal4	Yes	26	10	0	1	13
NoPortal5	Yes	2	4	1	1	13
NoPortal6	N/A	N/A	N/A	0	N/A	N/A

*Table 36: Validation results for file upload requirements*

Portal1 only had one step to upload files. Portal2, Portal3 did not describe the need to upload files in the scenarios. NoPortal1 describes the user not uploading files which triggered a lot more steps than expected. Even the labels assumed that it should be labeled to trigger that step. It was left as is although not an expected description.

Project submissions	Logging	# of steps classified as true	# of steps expected to be classified as true	Student logging Requirements	# of directly related requirements	# of requirements
Portal1	Yes	9	21	0		10
Portal2	Yes	11	16	0		10
Portal3	No	0	4	0		10
Portal4	Yes	12	32	0		10
Portal5	Yes	10	26	1		10
NoPortal1	Yes	52	34	0		10
NoPortal2	Yes	13	15	N/A	N/A	10
NoPortal3	Yes	33	28	0		10
NoPortal4	Yes	24	31	0		10
NoPortal5	Yes	22	25	5		10
NoPortal6	N/A	N/A	N/A	0	N/A	N/A

*Table 37: Validation results for logging requirements*

Table 37 above shows the results of parsing the projects through the SD-SRE for the logging suggested requirements.

Portal3 did not trigger a logging requirement but it is triggered if any other requirements topic is triggered. Table 38 below shows the number of projects that were triggered just by a standalone scenario from a project rather than because it extends another set of requirements.

Topic	Number of projects that triggered based on Use Case Scenarios	Number of projects that triggered additional requirement due to extending another requirement	Number of projects that should have had requirements trigger but didn't
Login	10	0	0
Sensitive Information	8	2	0
Authentication	0	10	0
Authorization	2	8	0
Email Collection	4	0	2
Input Validation	9	1	1
Encryption	2	8	2
Random Number	0	10	0
Database	10	10	0
File Upload	3	0	5
Logging	9	10	1

*Table 38: Number of projects that return a requirements grouping.*

#### 4.4.4 Comparison between Using the Portal or Not Using the Portal

Students not using the portal provided better quality use case scenarios. Based on feedback, although some students liked the portal layout, they prefer to see everything on the same page as you would for an SRS or while documenting use case models. Students who did not use the portal also submitted their use case model while the portal did not collect the use case model which may have discouraged students from drawing it prior to writing scenarios. Some students who did not use the portal stated that they would step away from the model and come back to it when they had an idea while this was not described by students using the portal. The portal was a learning curve as it was the first-time students were using it. The portal however had its pros it provided consistency for capturing the scenarios and making sure all of the fields were provided to be filled. The students who did not use the portal did not have consistency across their submissions. For example, one student did not have any preconditions for any of their use case.

#### 4.4.5 Comparison Between Recommended Security Requirements and Student Security Requirements

Overall the SD-SRE performed better than the student requirements which was expected. The students used were not security experts but instead early career system developers. On average the SD-SRE produced more than 140 requirements while students suggested only 18. The quality of the requirements related to security are more complete in the SD-SRE than what students produced.

Project Submissions	# of student suggested Requirements	# of SD-SRE Suggested Requirements
Portal1	25	134
Portal2	10	134
Portal3	13	134
Portal4	21	141
Portal5	19	134
NoPortal1	32	154
NoPortal2	0	141
NoPortal3	7	134
NoPortal4	13	154
NoPortal5	20	147
NoPortal6	22	0

*Table 39: Comparison of number of requirements student suggested vs the SD-SRE*

## 4.5 Research Questions Results

This section summarizes how the SD-SRE approach addresses the research questions.

### 4.5.1 How can the elicitation and analysis of functional features be leveraged to assist with the specification of cybersecurity requirements?

SD-SRE combines semantic web technologies and machine learning to analyze use case models which capture the functional features for systems to be developed. An ontology, the secure development ontology, was created to capture best practice security requirements of common functional features, vulnerabilities of these common functional requirements, mitigations of those vulnerabilities, and the common use case scenarios step descriptions of those functional features. 1183 use case scenarios were analyzed to determine the functional feature descriptions that are documented in the ontology. In creating the ontology 154 requirements were documented and stored in a repository. The security requirements were grouped into 11 topics that match functional features. The ontology was queried for the step descriptions and this was used to quickly label the



use case scenarios. The labeled data was used to train 11 machine learning models, one for every topic. This was then incorporated into a web portal where users can enter use case models and get suggested security requirements.

The efficacy of the SD-SRE was tested by having students create use case scenarios for a new project the system had not previously been trained on. The students were also asked to produce security requirements which were compared to the systems security requirements.

#### 4.5.2 How can the use of existing best practices of cybersecurity be leveraged to assist in the identification of cybersecurity requirements?

The use of existing best practices of cybersecurity were used to create a secure development ontology and a security requirements repository for generic frequently implemented functional features. The ontology captures best practices by documenting security requirements that were captured by reviewing 1076 security development sources. This ontology was leveraged to train a machine learning model that digest use case models and returns the security requirement recommendations from the security requirements repository. ML was used to make the matching of security requirements to functional requirements automatic.

The SD-SRE will continue to improve as more use case scenarios are added and analyzed to continue to improve the ML models. The ontology will continue to evolve as new security recommendations are made. The ontology also documents security requirements that use to be common practice but have since been considered insecure, for example the use of security questions as an authenticator or the use of SHA-1. As more hash algorithms are found to have collisions or currently adopted practices are abandoned such as abandoning passwords for multifactor authenticators the ontology will continue to be updated as well as the security requirements.



## 5 Related Work

This chapter describes related works in the development and application of cybersecurity ontologies, the use of machine learning in cybersecurity requirements elicitation, the use of ontologies with machine learning, the use of ontologies with machine learning in cybersecurity and is followed by a comparison to the proposed work.

### 5.1 Development of Cybersecurity Ontologies

The MITRE Corporation produced a trade study on developing an ontology for the cybersecurity domain using the middle out approach [52]. The authors chose malware as the cybersecurity topic to develop a method to be reused iteratively in the evolution of their cybersecurity ontology. This work took into consideration the perspective of the ontology user, an analysis of the data source, and reused existing ontologies that had security related concepts. The authors verified their ontology through the development of competency questions that they used to create use cases. The two goals of this work were to document a process for developing cybersecurity ontologies, and to catalog useful resources for the cybersecurity domain.

A similar project presented a framework for building an ontology for cybersecurity focused on situational awareness [53]. The research proposed adopting a semantic model of cybersecurity to overcome the limitations of situational awareness of analysts due to the complex interaction of human and machine in a widespread communication network. The authors reviewed multiple ontologies to extract the foundations for an ontology of secure operations in cyberspace. The Ontologies of Secure Cyber Operations (OSCO) was developed by incorporating three ontologies; the CRATELO ontology, which is an ontology used as a base reference, the DOLCE\_SPRAY, and SECCO ontologies, which defined additional security concepts.

The approach was used to develop parts of the OSCO ontology, which was verified using two use cases, the secure retrieval of a file and the detection of an intrusion. One of the goals of this research was to build an ontology that could reduce the number of alarms by identifying false alarms to assist analyst in using resources more effectively.

Another paper discussed the efforts of researchers to combine several general world and security related ontologies together to make the Unified Cybersecurity Ontology (UCO) [63]. The authors developed this ontology to provide a common understanding of cybersecurity domain and standards, map existing ontologies, map industry knowledge, and develop foundational use cases to verify the UCO ontology. The development of the UCO ontology focused on supporting information integration and situational awareness. The authors incorporated ontologies to expand the diversity of use cases that the UCO ontology could handle. UCO is based on the Intrusion Detection System that describes cybersecurity related events. The use cases developed to verify the ontology were based on situation and threat understanding by correlating contextual observations (STUCCO) [64], an ontology that extracted entities from the national vulnerability database. Using the UCO ontology, four example results were presented: identification of PDF reader vulnerabilities, identification of vulnerabilities in other products, suggestion of alternate software without the vulnerability the user is trying to avoid, and the assessment of changing a product vendor based on product vulnerability counts.

These papers define approaches for building cybersecurity ontologies through the reuse of existing ontologies or combining previously defined ontologies. The development of cybersecurity ontologies can be described as either a combination of security ontologies or a combination of domain (i.e., human factors) and security ontologies. These ontologies were developed from the perspective of addressing cybersecurity problems in system operations.

## 5.2 Applications of Cybersecurity Ontologies

Researchers have used ontologies to predict cybersecurity threats. One example of this is a system that was developed to use cybersecurity ontologies to analyze web conversations scraped off the internet to predict when hackers are planning attacks and to assess the viability of the attack [65].

Another use of cybersecurity ontologies was for solving the problem of determining security breaches [54]. In this instance the system scans data across enterprises and links them in a semantic graph that uses ontologies to identify breaches and suggest responses. This paper describes establishing an ontology with the dual purpose of providing a data map and the ability to provide an automatic translation. The authors state, “The main weakness of these ontologies is that they focus more on objects rather than events.” The authors argue that capturing events in addition to objects is necessary to construct a timeline of events that can increase situational awareness.

The aforementioned approaches all target solving problems that arise from system operations. Contrary to this, the approach proposed in this research aims to establish an ontology to solve the first category of cybersecurity type of problems, developing optimal systems, by using an ontology to assist in the elicitation of cybersecurity requirements at the earliest stages of software development i.e., the requirements gathering stage. Because software developers are not necessarily security experts, there are security issues that continuously appear in newer systems. One possible reason is that developers may not be aware of the existence of the issue and the solution, or they might have missed the security requirement. Cybersecurity developments are reactive to discovered threats but many of these threats have simple proven solutions, but those solutions are slowly implemented and are often after the system has been deployed. Therefore, the ontology developed in this research is based on secure development. The Secure Development

Ontology (SDO) maps security attacks, mitigation, and the implementation to mitigate the threat [59].

### 5.3 The Use of Machine Learning in Requirement Elicitation

The use of machine learning in requirement elicitation has been used to prioritize elicited requirements for implementation, to elicit requirements based on inductive learning approaches, to help select which requirement elicitation techniques are best used for a proposed project, and automatic requirements elicitation in agile processes [66], [67], [68], [69].

Machine learning in the requirements gathering phase of the development lifecycle has been used to prioritize requirements. Determining which requirements to focus on first is a strategic process that software developers take on after requirements are gathered. It is tedious and time consuming because like requirements need to be grouped in addition to system critical requirements. A novel framework was developed that interweaves human and machine activities to properly prioritize requirements. The approach is similar to Analytic Hierarchy Process (AHP) in that it is based on pairwise preferences but differs in that it allows prioritization over a large set of requirements [70]. Later the same authors explored the scalability issues of requirements prioritizations with machine learning techniques. AHP is impractical with requirement sets greater than 20. The method instead relied on a case-based ranking by combining human preference elicitation and automatic preference approximation [71]. Case-based ranking considers the stakeholders preferences with requirements ordering approximations by taking in the human effort input and encoding the domain knowledge with a partial order requirements attributes [66].

A hybrid machine learning model was created to determine the best requirements elicitation techniques. A multi regression model was built based on the strengths and weaknesses of each technique to determine how each techniques attributes rank alone and how each attribute would

affect one another. The authors stated that most activities are performed due to familiarity or history of using a technique regardless of whether or not it is the best technique to elicit requirements for a specific problem. Then the authors used an Artificial Neural Network (ANN) to select elicitation techniques given the proposed project [68].

An empirical study was conducted that used machine learning to predict project effort. Effort was defined as cost in this study. The authors looked at projects from the bottom-up perspective in two different organizations. They were able to apply a neural network approach to produce predictions for different parts of project with about 90% accuracy. The authors concluded that this method had potential with the ability to scale the data [72].

Another machine learning approach was designed to identify expert stakeholders in the requirements gathering process. The machine learning technique analyzed stakeholder contributions, extracted domain specific topics, and made profiles of stakeholder's interest. These profiles are then filtered and chosen based a given requirements elicitation topic [73].

Methodologies have been created to classify the quality of requirements using machine learning. The methodology is a learning-based machine learning approach based on metrics that represent attributes of requirements that experts would consider good or bad qualities of requirements. The classifier was trained with requirements that were pre-classified then tested against a new set of requirements [74].

Another method used ML to automatically gather functional requirements from agile processes [69]. The method is based on task adaptive leaning by justification trees algorithm. This method supports many of the agile processes that extreme programming requires. The authors used a knowledge base that applied some of the extreme programming techniques which allowed for

adjustments and expansion. The authors were able to produce a set of requirements that could serve as final requirements.

A semi supervised learning approach was used to identify non-functional requirements in textual specifications. The authors used pre-classified non-functional requirements to train the model. The classification was based on categorization of text properties and feedback from users. The approach was 70% accurate in the detection non-functional requirements [75]. ML is applied in various ways in the requirements elicitation domain such as classifying the best elicitation techniques, prioritizing requirements for implementation, to determining project effort, identifying stakeholders, determining quality of requirements. Two previous works stand out when compared to the proposed approach in this research: gathering functional requirements from agile processes and classifying non-functional requirements from a set of requirements. Classifying non-functional requirements is of interest to extend this work in the future because the current methodology does not consider if the security requirement is already present it assumes that it is not. There are times security requirements may already be present and having the system identify them may be of interest in the future. Gathering non-functional requirements from agile processes is the closest to this approach but this approach focuses on suggesting security requirements.

#### **5.4 The Use of Machine Learning in Cybersecurity Requirement Elicitation**

In 2016 researchers mined Software Requirement Specifications (SRS) for security requirements and developed a classification model. The security requirements were broken down into four classifications: authentication-authorization, access control, cryptography- encryption, and data integrity. From the collected mined data, the authors classified the requirements then tested this against security requirements projects [67].



The authors work is similar to the proposed approach, but the authors are not suggesting new security requirements whereas this approach is suggesting new security requirements not classifying already written requirements.

## 5.5 The Use of Ontologies with Machine Learning

Researchers explored the multiple ways ontologies have been used with machine learning. In one study the researchers used artificial intelligence methods to determine if two life science ontologies were the same or similar. The authors also use unsupervised machine learning techniques to embed new information in ontologies. Then the authors explored methods that use ontologies to constrain machine learning problems [76].

A survey paper looked at 15 published in 2018 and 2019 in the medical field that incorporated the use of ontologies, ML, and a hybrid of ML and ontologies. The hybrid approaches focused on vectorizing ontologies to be consumed by ML algorithms, tries to predict the context node given a node, and the embedding of ontologies to classify patients [77]. Another survey paper looked at how ML is being applied to the semantic web in order to find missing links and enrich ontologies [78].

These works incorporate the use of ontologies with machine learning but many of the applications are for training a model on the ontology in order to improve the ontology by finding missing links and entities not as data labels themselves.

## 5.6 The Use of Ontologies with Machine Learning in Cybersecurity

Researcher leveraged ontologies and ML techniques for malware analysis. The authors created an ontology of 4570 Android apps with their associated features then used ML to identify malware features and flag whether or not the app may have malware [79].

Another implementation of ontologies with ML was the creation of an automatic attack detection ontology that trained a deep learning algorithm to identify cyber-attacks specifically intrusion detection from system logs [71]. Other researchers trained a ML model on UCO to detect early cyber security attacks [75].

Other researchers created an ontology of security requirements to train a ML algorithm on how to identify and classify security requirements from general requirements in requirements specifications [80]. One of the authors has previous work on identifying security requirements based on linguistic analysis and machine learning. In that work the author used linguistic rules to train the model and did not create an ontology of security requirements [81].

Researchers used an ontology to validate the results of a supply chain treat analysis and prediction ML model [82]. Other authors used ML to extract entities from cybersecurity domain corpuses to build an ontology and a cybersecurity knowledge base [83]. Similarly, other researchers developed a security related ontology to train an NLP ML model to find entities and relations from cybersecurity related documents [84].

These works focus on security for implemented systems, the classification of security requirements from general requirements, and ML to implement ontologies in the cybersecurity domain. In contrast the proposed approach uses an ontology to label use case scenarios to suggest cybersecurity requirements that should be considered in the implementation of the functional features described within the scenarios.

## 5.7 Comparison to Approach

The cybersecurity related ontologies were developed from the perspective of addressing cybersecurity problems in system operations. While the ontology proposed in this work is from the perspective of building optimal systems.

ML is applied in various ways in the requirements elicitation domain such as classifying the best elicitation techniques, prioritizing requirements for implantation, to determining project effort, identifying stakeholders, determining quality of requirements, identification of non-functional requirements, gathering requirements from elicitation techniques, and classifying requirements. These works differ from the proposed approach as the proposed approach is suggesting security requirements from functional requirements elicitation activities.

## 6 Results and Conclusion

This section discusses the results of the SD-SRE validation, the benefits of the SD-SRE and the future work to be done to continue to improve the SD-SRE.

### 6.1 SD-SRE Results

The SD-SRE performed better in both quantity and quality of requirements when compared to those who did not use the SD-SRE. The SD-SRE on average suggested 140 security requirements while students on average suggested 18 with an average of 8 being security related. If the aggregators, which allows for piggybacking requirements, was not part of the system the SD-SRE would on average return 107 requirements.

The students on average had 2-3 security related requirements the SD-SRE did not capture directly. These requirements were very specific to the system and will not be added to the SD-SRE. The requirements discussed limiting the amount of money that a user could send.

Project Submissions	# of student suggested requirements	# of security related student suggested requirements	# of security related student suggested requirements not capture by SD-SRE	# of SD-SRE suggested requirements without Aggregator	# of SD-SRE Suggested Requirements
Portal1	25	4	3	98	134
Portal2	10	7	2	108	134
Portal3	13	4	3	72	134
Portal4	21	7	2	99	141
Portal5	19	6	2	116	134
NoPortal1	32	26	1	118	154
NoPortal2	0	0	0	105	141
NoPortal3	7	3	1	126	134
NoPortal4	13	6	0	118	154
NoPortal5	20	10	0	111	147
NoPortal6	22	3	3	0	0

*Table 40: SD-SRE validation results*

## 6.2 Results of Best Practices

The students are novice system developers and none of the students indicated having a specialty in cybersecurity. The lack of cybersecurity expertise and training is a motivator for this project as it is a representation of a majority of industry developers. Below we see a comparison of security requirements produced by the students compared to similar requirements produced by the SD-SRE.

Noportal1 had 26 security related requirements but of those 26, three requirements are no longer recommended security requirements as they are considered unsecure due to the rise of social media. Security questions are not secure because they typically suggest security questions ask about information that can easily be found on the internet due to the nature of social media, which encourages users to overshare. Examples such as “what was your elementary school teachers name” can easily be found through various digital yearbook website or “what is the name of your dog” can likely be found by looking at someone’s social media. The three requirements suggested by Noportal1 in regard to security questions are the following:

“The system shall prompt the user to create security questions during account creation.”

“Users shall have 3 to 5 security questions at all times.”

“The system shall prompt the user to update security questions every 4 months.”

The SD-SRE returns this requirements which recommends not using security requirements:

“The system shall not use security questions for account recovery.”

Another example of security requirements suggested that did not follow best practices focused on password creation. NoPortal1 also had security requirements about password length and character requirements as shown here:

“Passwords shall be 10 or more characters.”

“Passwords shall include 1 or more numbers.”

“Passwords shall include 1 or more special characters.”

“Passwords shall contain no spaces.”

NoPortal4 also had similar requirements:

“A valid password shall be between 10 and 20 characters long.”

“A valid password shall include at least one lower case letter, one upper case letter, one number, and one special symbol.”

The SD-SRE returns these requirements in relation to password selection:

“The system shall ensure the password is not blank.”

“The system shall require passwords with a minimum of 12 characters.”

“The system shall require passwords with a maximum of 128 (some still recommend 64...depends on hashing implementation) characters limitation. Note that this may be adjusted in the future as hashing algorithms have more capacity.”

“The system shall allow usage of all characters in passwords including Unicode, whitespaces, and emojis.”

“The system shall not limit or require the number or type of characters a user may use in the password field (i.e. no password composition rule).”

Both Noportal1 and Noportal4 are suggesting password composition rules which are no longer best practice and the SD-SRE returns a requirement stating to not implement a composition rule. NoPortal4 suggest the length of the password be at least 10 characters but no longer than 20 characters. Best practices for password length suggest a minimum of 12 characters and a maximum of 128 which is limited due to the hashing algorithm to be used.

Portal4 did capture a best practice security requirement about using a strength meter which is also suggested by the SD-SRE.

Portal4 suggested strength meter requirement:

“The system shall include a strength meter when the user is choosing a password.”

SD-SRE suggested strength meter requirement:

“The system shall use a password strength meter and present it to the users.”

The security requirements suggested by the students are likely due to features they’ve experienced themselves such as password composition rules. The SD-SRE suggest the best practices for these requirements based on current security standards.

### 6.3 The Benefits of the SD-SRE

This research could be an effective way to implement cybersecurity requirements into the earliest stage of the software development life cycle because this approach will allow developers to elicit cybersecurity requirements throughout the requirements gathering stage.

It is envisioned that the SD-SRE provides a way to elicit cybersecurity requirements that can be reused. This approach will allow requirements engineers to focus on, what use cases are best used for, functional requirements through the analysis of requirement engineering techniques. The SD-SRE can minimize the added layer of activities of current security requirement approaches such as abuse and misuse cases allowing developers to focus on the elicitation of functional requirements. Abuse/misuse cases add a layer to use case modeling for security by having developer describe the many ways that a malicious user could abuse/misuse a system. This requires the developer to have cybersecurity skills in order to properly determine the multitude of ways a system can be targeted and to write effective requirements to mitigate those instances. In contrast to this, the SD-SRE does not require the requirements engineer to consider the many ways a system

is vulnerable. It also does not require the requirements engineer to come up with requirements to mitigate those vulnerabilities. The SD-SRE will use the knowledge of the system stored in requirements gathering activities to assist with the elicitation and analysis of cybersecurity requirements.

The SD-SRE can be a seamless manner to discover cybersecurity requirements, in a lightweight way as it can easily handle changes to requirements without the extensive steps proposed by already existing frameworks. The SD-SRE allows researchers to capture more data to continue to enhance the approach. The capture of functional system features could be reanalyzed by researchers with cybersecurity domain knowledge to see if new security related requirements should be added to the domain knowledge map and security requirements repository. As existing standards and frameworks are updated this approach can easily be updated to reflect those new improvements.

Using the SD-SRE requirements engineers can rapidly, non-invasively, identify security related requirement at the earliest stages of the development lifecycle. Requirements engineers are already performing requirement elicitation and analysis activities which are often used to later develop other models and test cases. These techniques can now be used to analyze the security of a system. This is beneficial because the security requirement will be integrated in the system, through design and architecture. Instead of later once the system is deployed. Additionally, this can facilitate the sharing of cybersecurity requirements as requirements engineering activities can also be reused. Furthermore, using requirements engineering activities, software engineers can develop testing plans of the system. The SD-SRE can then in the future be expanded to also support the validation and verification of the system security.



## 6.4 Future Work

This research proposes an approach to elicit security requirements at the earliest stage of the development lifecycle. This work can continue to expand as new requirements and vulnerabilities are discovered. For example, as hashing algorithms are cracked or common practices such as security questions are shown to not be secure. These new discoveries will need to be added and adjusted in the ontology. As the SD-SRE portal continues to populate with new scenarios new description of features will be discovered and the models will need to be trained on those new descriptions. The approach can be improved by categorizing requirements based on the severity of vulnerability the requirement is mitigating. The use case parser can also be improved by implementing a more in-depth threat model. The model can be expanded to suggest security requirements from functional requirements and to digest user stories in addition to use case scenarios. The model can also be adjusted to identify already existing security requirements to not duplicate the requirements in the suggestions and to identify incomplete requirements. If the SD-SRE portal reaches tens of thousands of scenarios the model can be adjusted to a semi supervised model in order to limit the interaction needed when new descriptions are added.

This approach can also be expanded to the design and implementation phase. Design phase models can be analyzed to train a model to identify security design suggestions based on existing security frameworks. This would address the current weakness in the model for physical systems as many of the security suggestions for the physical devices rely on design. There are many security standards for secure coding based on programming languages. A plug in can be designed for popular Integrated Development Environments (IDE) or code editors to identify security flaws in code and implementation decisions. For example, preventing tab nabbing and clickjacking in HTML, tokenside jacking in JSON, properly implementing Docker security, and preventing unsafe

javascript by using the “eval” function. Although not full proof all these examples can be mitigated through certain practices.

The SD-SRE portal can be expanded to allow developers to interact with other developers on the suggested requirements whether through a comments section or a forum feature. There can be deliberate features for developers to provide feedback or suggestions on the suggested requirements. The SD-SRE portal can also host security development related new, trainings, and seminars on security requirements elicitation, design, and implementation. The SD-SRE portal can link to additional resources for suggested requirements, design, or code suggestions.

## 7 References

- [1] Ponemon Institute, "2017 Cost of cybercrime study, insights on the security investments that make A difference," [Online]. Available: [https://www.accenture.com/\\_acnmedia/PDF-62/Accenture-2017CostCybercrime-US-FINAL.pdf#zoom=50](https://www.accenture.com/_acnmedia/PDF-62/Accenture-2017CostCybercrime-US-FINAL.pdf#zoom=50). [Accessed 15 October 2022].
- [2] Ponemon, Accenture, "The cost of cybercrime, ninth annual cost of cybercrime study unlocking the value of improved cybersecurity protection," [Online]. Available: [https://www.accenture.com/\\_acnmedia/pdf-96/accenture-2019-cost-of-cybercrime-study-final.pdf](https://www.accenture.com/_acnmedia/pdf-96/accenture-2019-cost-of-cybercrime-study-final.pdf). [Accessed 15 October 2022].
- [3] Federal Bureau of Investigation, "Internet crime report 2021," Internet Crime Complaint Center, [Online]. Available: [https://www.ic3.gov/Media/PDF/AnnualReport/2021\\_IC3Report.pdf](https://www.ic3.gov/Media/PDF/AnnualReport/2021_IC3Report.pdf). [Accessed 15 October 2022].
- [4] Z. M. Smith, E. Lostri and J. A. Lewis, "The hidden cost of cybercrime," McAfee Foundation, [Online]. Available: <https://www.mcafee.com/enterprise/en-us/assets/reports/rp-hidden-costs-of-cybercrime.pdf>. [Accessed 15 October 2022].
- [5] S. Morgan, "Cybercrime to cost the world \$10.5 trillion annually by 2025," Cybercrime Magazine, 13 November 2020. [Online]. Available: <https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/>. [Accessed 15 October 2022].

- [6] IBM Security, "Cost of a data breach report 2022," IBM, [Online]. Available: <https://www.ibm.com/downloads/cas/3R8N1DZJ>. [Accessed 15 October 2022].
- [7] Accenture, "State of cybersecurity resilience 2021," [Online]. Available: [https://www.accenture.com/\\_acnmedia/PDF-165/Accenture-State-Of-Cybersecurity-2021.pdf](https://www.accenture.com/_acnmedia/PDF-165/Accenture-State-Of-Cybersecurity-2021.pdf). [Accessed 15 October 2022].
- [8] RISKIQ, "Evil internet minute 2021," [Online]. Available: <https://safe.riskiq.com/rs/455-NHF-420/images/Evil-Internet-Minute-RiskIQ-Infographic-2021.pdf>. [Accessed 15 October 2022].
- [9] A. Dellinger, "Understanding the first american financial data leak: how did it happen and what does it mean?," *Fornes*, 26 May 2019. [Online]. Available: <https://www.forbes.com/sites/ajdellinger/2019/05/26/understanding-the-first-american-financial-data-leak-how-did-it-happen-and-what-does-it-mean/?sh=2828dfdc567f>. [Accessed 15 October 2022].
- [10] M. Hill and D. Swinhoe, "The 15 biggest data breaches of the 21st century," *CSO*, [Online]. Available: <https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html>. [Accessed 15 October 2022].
- [11] D. R. L. Sveinsson, "Top 10 data reaches so far in 2022," *ERMProtect Cybersecurity Solutions*, [Online]. Available: <https://ermprotect.com/blog/top-10-data-breaches-so-far-in-2022/>. [Accessed 15 October 2022].
- [12] A. Greenberg, "A remote-start app exposed thousands of cars to hackers," *WIRED*, 10 August 2019. [Online]. Available: <https://www.wired.com/story/mycar-remote-start-vulnerabilities/>. [Accessed 15 October 2022].

- [13] "Here are the latest, most damaging things in the DNC's leaked emails," The Washington Post, 25 July 2016. [Online]. Available: <https://www.washingtonpost.com/news/the-fix/wp/2016/07/24/here-are-the-latest-most-damaging-things-in-the-dncs-leaked-emails/>. [Accessed 2022 October 2022].
- [14] G. Oladipo, "Cyberattacks force over a dozen US airport websites offline," TheGuardian, 10 October 2022. [Online]. Available: <https://www.theguardian.com/us-news/2022/oct/10/cyberattacks-disrupt-us-airport-websites>. [Accessed 25 October 2022].
- [15] InfoSec, "A history of anonymous," InfoSec, 2011 October 11. [Online]. Available: <https://resources.infosecinstitute.com/topic/a-history-of-anonymous/>. [Accessed 15 October 2022].
- [16] TrendMicro, "Hacktivism 101: a brief history and timeline of notable incidents," TrendMicro, 17 August 2015. [Online]. Available: <https://www.trendmicro.com/vinfo/pl/security/news/cyber-attacks/hacktivism-101-a-brief-history-of-notable-incidents>. [Accessed 15 October 2022].
- [17] S. Langlois, "Hacktivism 101: A brief history and timeline of notable incidents," MarketWatch, 2 November 2020. [Online]. Available: <https://www.marketwatch.com/story/founder-of-hacker-group-anonymous-reveals-his-ultimate-endgame-11604336926>. [Accessed 15 October 2022].
- [18] J. Abbate and W. Aspray, Recoding gender: women's changing participation in computing, Cambridge: MIT Press, 2012.
- [19] C. Connley, "Google, Apple and 12 other companies that no longer require employees to have a college degree," CNBC, 2018.

- [20] J. Baker, "2018's Software engineering talent shortage- It's quality, not just quantity," Hackernoon, 2017.
- [21] K. Wiegers and J. Beatty, "Software requirements," Pearson Education, 2013.
- [22] J. Whittaker, "What is software testing? And why is it so hard?," vol. 17, no. 1, pp. 70-79, 2000.
- [23] J. Dooley, Software development and professional practice, Apress, 2011.
- [24] P. Hope, G. McGraw and A. Annie, "Misuse and abuse Cases: getting past the positive," *IEEE Security and Privacy*, vol. 2, no. 03- May-June, pp. 90-92, 2004.
- [25] D. Mellado, C. Blanco, L. Sánchez and E. Fernández-Medina, "A systematic review of security requirements engineering," *Computer Standards and Interfaces*, vol. 32, pp. 153-165, 2010.
- [26] "14 Annual CSI computer crime and security survey," Computer Security Institute, 2009.
- [27] A. Razzaq, Z. Anwar, F. H. Ahmad, K. Latif and F. Munir, "Ontology for attack detection: an intelligent approach to web application security," *Computers & Security*, vol. 45, pp. 126-146, 2014.
- [28] E. Sheridan, "While most developers worry about security, many development teams lack a dedicated security expert," DEVOPSDigest, 2020. [Online]. Available: <https://www.devopsdigest.com/while-most-developers-worry-about-security-many-development-teams-lack-a-dedicated-security-expert>. [Accessed 15 October 2022].
- [29] P. A. Laplante, "A requirements engineering for software and systems," *Taylor and Francis Group LLC.*, 2009.
- [30] B. Nuseibeh and S. Easterbrook, Requirements engineering, Academic Press, 2003.

- [31] U. Eriksson, "Why is the difference between functional and Non-functional requirements important?," ReQTest, 5 4 2012. [Online]. Available: <https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/>. [Accessed 15 3 2021].
- [32] Requirements Quest, "Nonfunctional requirements," Requirements Quest, 2018. [Online]. Available: <https://requirementsquest.com/nonfunctional-requirement-examples/>. [Accessed 5 3 2021].
- [33] Tech Talk, "Requirement analysis in software design," Tech Talk, 06 2 2015. [Online]. Available: <https://tech-talk.org/2015/02/06/requirement-analysis-in-software-design/>. [Accessed 5 3 2021].
- [34] S. G. Yoo, H. P. Vaca and J. Kim, "Enhanced misuse cases for prioritization of security requirements," in *Proceedings of the 9th International Conference on Information Management and Engineering*, Barcelona, 2017.
- [35] Z. Dwaikat and F. Parisi-Presicce, "From misuse cases to collaboration diagrams," in *3rd International Workshop on Critical Systems Development with UML*, 2004.
- [36] OWASP, "OWASP secure coding practices - quick reference guide," OWASP, 2010.
- [37] CERT, "Secure coding standard," CERT, 2018.
- [38] MITRE, "Common weakness enumeration (CWE)," MITRE, 2022.
- [39] T. Berners-Lee, *Semantic web*, Washington DC: W3C, 2000.
- [40] S. Faubel, "Semantic web stack," 2007.
- [41] G. Tumarello, *The semantic web*, Milano: W3C, 2005.
- [42] "Semantic definition," Oxford Dictionary, [Online]. Available: <https://en.oxforddictionaries.com/definition/semantic>.

- [43] N. F. Noy and D. L. McGuinness, *Ontology development 101: a guide to creating your, Stanford university*.
- [44] IEEE, "Suggested upper merged ontology (SUMO)," IEEE, [Online]. Available: <https://www.ontologyportal.org/>. [Accessed 15 October 2022].
- [45] ISTC-CNR Laboratory for Applied Ontology, "DOLCE : descriptive ontology for linguistic and cognitive engineering," ISTC-CNR Laboratory for Applied Ontology, [Online]. Available: <http://www.loa.istc.cnr.it/dolce/overview.html>. [Accessed 15 October 2022].
- [46] W3C, "SKOS Simple Knowledge Organization System," 18 August 2009. [Online]. Available: <https://www.w3.org/TR/skos-reference/>. [Accessed 15 October 2022].
- [47] IDEF, "Integrated DEFinition Methods (IDEF)," IDEF, [Online]. Available: <https://www.idef.com/idef5-ontology-description-capture-method/>. [Accessed 15 October 2022].
- [48] J. Cuenca, F. Larrinaga and E. Curry, "MODDALS methodology for designing layered ontology structures," *Applied Ontology*, vol. 15, no. 2, pp. 185-217, 2020.
- [49] "Basic Formal Ontology," BFO, [Online]. Available: <https://basic-formal-ontology.org/>. [Accessed 15 October 2022].
- [50] T. Chungoora, "Practical knowledge modelling: ontology development 101," Knowledge Graph Consultant. [Online]. [Accessed 15 October 2022].
- [51] J. Raad and C. Cruz, "A survey on ontology evaluation methods," in *International Conference on Knowledge Engineering and Ontology Development, part of the 7th International*, Lisbonne, 2015.



- [52] L. Obrst, P. Chase and R. Markeloff, "Developing an ontology of the cyber security domain," in *Seventh International Conference on Semantic Technologies for Intelligence, Defense, and Security*, 2012.
- [53] A. Oltramari, L. F. Cranor, R. J. Walls and P. McDaniel, "Building an ontology of cyber security," in *STIDS 2014 (9th International Conference on Semantic Technology for Intelligence, Defense, and Security)*, 2014.
- [54] M. B. Salem and C. Wacek, "Enabling new technologies for cyber security defense with the ICAS cyber security ontology," in *Semantic Technology for Intelligence, Defense, and Security*, 2015.
- [55] G. Bonaccorso, *Machine learning algorithms : popular algorithms for data science and machine learning*, Packt Publishing Ltd, 2018.
- [56] O. Trekhelb, "Homemade machine learning in python," [Online]. Available: <https://medium.com/datadriveninvestor/homemade-machine-learning-in-python-ed77c4d6e25b>.
- [57] Scikit Learn, "Choosing the right estimator," [Online]. Available: [https://scikit-learn.org/stable/tutorial/machine\\_learning\\_map/index.html](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html). [Accessed 15 October 2022].
- [58] Microsoft, "Machine learning algorithm cheatsheet," [Online]. Available: <https://learn.microsoft.com/en-us/azure/machine-learning/algorithm-cheat-sheet>. [Accessed 15 October 2022].
- [59] J. Steinmann and O. Ochoa, "Supporting security requirements engineering through the development of the secure development ontology," in *2022 IEEE 16th International Conference on Semantic Computing (ICSC)*, 2022.

- [60] NIST, "Information Technology Cybersecurity," NIST, [Online]. Available: <https://www.nist.gov/cybersecurity>. [Accessed 15 October 2022].
- [61] OWASP, "OWASP," [Online]. Available: <https://owasp.org/>. [Accessed 15 October 2022].
- [62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: machine learning in python," *Journal of Machine Learning Research*, pp. 2825-2830, 2011.
- [63] Z. Syed, A. Padia, T. Finin, L. Mathews and A. Joshi, "UCO: a unified cybersecurity ontology," in *AAAI Workshop on Artificial Intelligence for Cyber Security*, 2016.
- [64] M. Lannacone, S. Bohn, G. Nakamura, J. Gerth, K. Huffer, R. Bridges, E. Ferragut and J. Goodall, "Developing an ontology for cyber security knowledge graphs," in *Cyber and Information Security Research Conference*, 2015.
- [65] T. M. Georgescu and I. Smeureanu, "Using ontologies in cybersecurity field," *Informatica Economică*, vol. 21, no. 3/2017, pp. 5-15, 2017.
- [66] A. Perini, A. Susi and P. Avesani, "A machine learning approach to software requirements prioritization," *International Journal of Scientific Engineering and Technology Research*, vol. 3, no. 32, pp. 6409-6416, 2014.
- [67] R. Jindal , R. Malhotra and A. Jain, "Automated classification of security requirements," in *2016 International Conference on Advances in Computing, Communications and Informatics*, Jaipur, 2016.

- [68] N. R. Darwish, A. A. Mohamed and A. S. Abdelghany, "A hybrid machine learning model for selecting suitable requirements elicitation techniques," *International Journal of Computer Science and Information Security*, vol. 14, no. 6, pp. 380-391, 2016.
- [69] . A. (. Ronit, "Automatic requirements elicitation in agile processes," in *IEEE International Conference on Software - Science, Technology and, Engineering*, 101-109, 2005.
- [70] P. Avesani, C. Bazzanella, A. Perini and A. Susi, "Supporting the requirements prioritization process," in *SEKE 2004: International Conference on Software Engineering and Knowledge Engineering*, Povo-Trento, 2004.
- [71] P. Avesani, C. Bazzanella, A. Perini and A. Susi, "Facing scalability issues in requirements prioritization with machine learning techniques," in *13th IEEE International Conference on Requirements Engineering*, Paris, 2005.
- [72] G. D. Boetticher, "Using machine learning to predict project effort: empirical," San Diego, 2001.
- [73] C. Castro-Herrera and J. Cleland-Huang, "A machine learning approach for identifying expert stakeholders," in *2009 Second International Workshop on Managing Requirements Knowledge*, Atlanta, 2010.
- [74] E. Parra, C. Dimou, J. Llorens, V. Moreno and A. Fraga, "A methodology for the classification of quality of requirements using machine learning techniques," *Information and Software Technology*, vol. 67, pp. 180-195, 2015.
- [75] A. Casamayor, D. Godoy and M. Campo, "Identification of non-functional requirements in textual specifications: a semi-supervised learning approach," in *National Council for Scientific and Technical Research*, 2009.

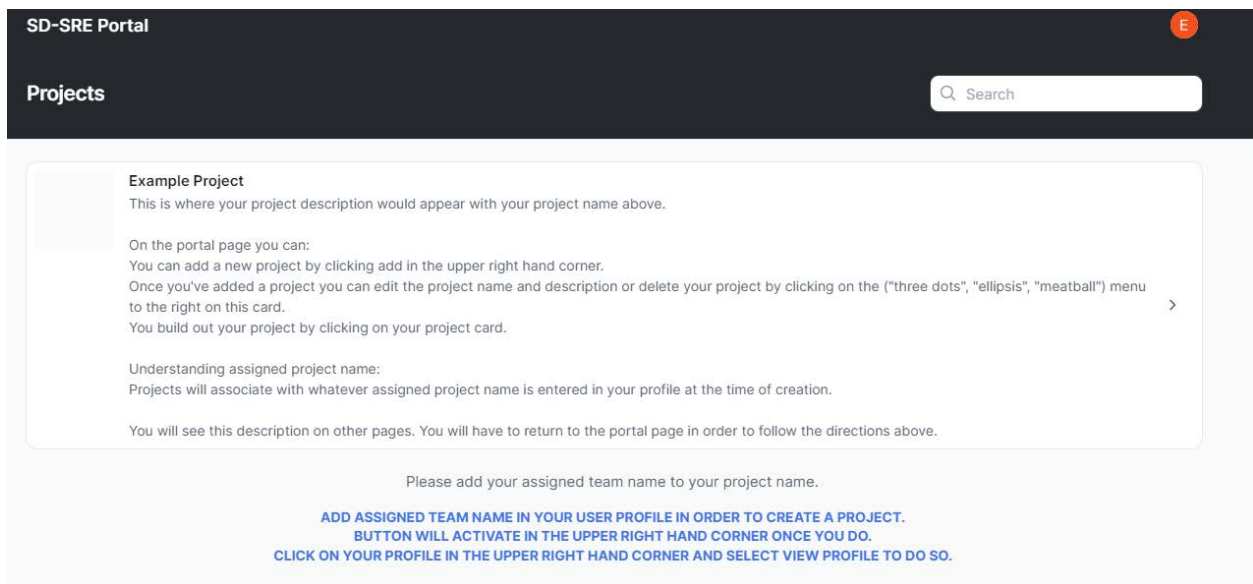
- [76] M. Kulmanov, F. Z. Smaili, X. Gao and R. Hoehndorf, "Semantic similarity and machine learning with ontologies," *Briefings in Bioinformatics*, vol. 22, no. 4, 2021.
- [77] P. N. Robinson and M. A. Haendel, "ontologies, knowledge representation, and machine learning for translational research: recent contributions," *Yearb Med Inform*, pp. 159-162, 2020.
- [78] C. d'Amato, "Machine learning for the semantic web: lessons learnt and next research directions," *Semantic Web*, vol. 11, no. 1, pp. 195-203, 2020.
- [79] L. C. Navarro, A. K. Navarro, G. André, A. Rocha and R. Dahab, "Leveraging ontologies and machine-learning techniques for malware analysis into Android permissions ecosystems," *Computers and Security*, vol. 78, pp. 429-453, 2018.
- [80] T. Li and Z. Chen, "An ontology-based learning approach for automatically classifying security requirements," *Journal of Systems and Software*, vol. 165, p. 110566, 2020.
- [81] T. Li, "Identifying security requirements based on linguistic analysis and machine learning," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, 2017.
- [82] A. Yeboah-Ofori, H. Mouratidis, U. Ismai and S. Islam, "Cyber supply chain threat analysis and prediction using machine learning and ontology," in *IFIP International Conference on Artificial Intelligence Applications and Innovations: AIAI 2021: Artificial Intelligence Applications and Innovations*, 2021.
- [83] Y. Jia, Y. Qi, H. Shang, R. Jiang and A. Li, "A practical approach to constructing a knowledge graph for cybersecurity," *Engineering*, vol. 4, no. 1, pp. 53-60, 2018.
- [84] T.-M. Georgescu, "Natural language processing model for automatic analysis of cybersecurity-related documents," *Symmetry*, vol. 12, no. 3, 2020.

- [85] NIST, "SP 800- 207 zero trust architecture," August 2020. [Online]. Available: <https://csrc.nist.gov/publications/detail/sp/800-207/final>. [Accessed 15 October 2022].
- [86] H. Zheng, Y. Wang, C. Han, F. Le, R. He and J. Lu, "Learning and applying ontology for machine learning in cyber attack detection," in *018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018.
- [87] S. N. Narayanan, A. Ganesan, K. Joshi, T. Oates, A. Joshi and T. Finin, "Early detection of cybersecurity threats using collaborative cognition," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, 2018.

## 8 Appendix

### 8.1 The SD-SRE Portal

The requirements SD-SRE portal welcome page is shown below in Figure 46 with the example project that explains how the SD-SRE portal works.



*Figure 46: SD-SRE Portal welcome page*

The add project ability is shown in Figure 47 below.

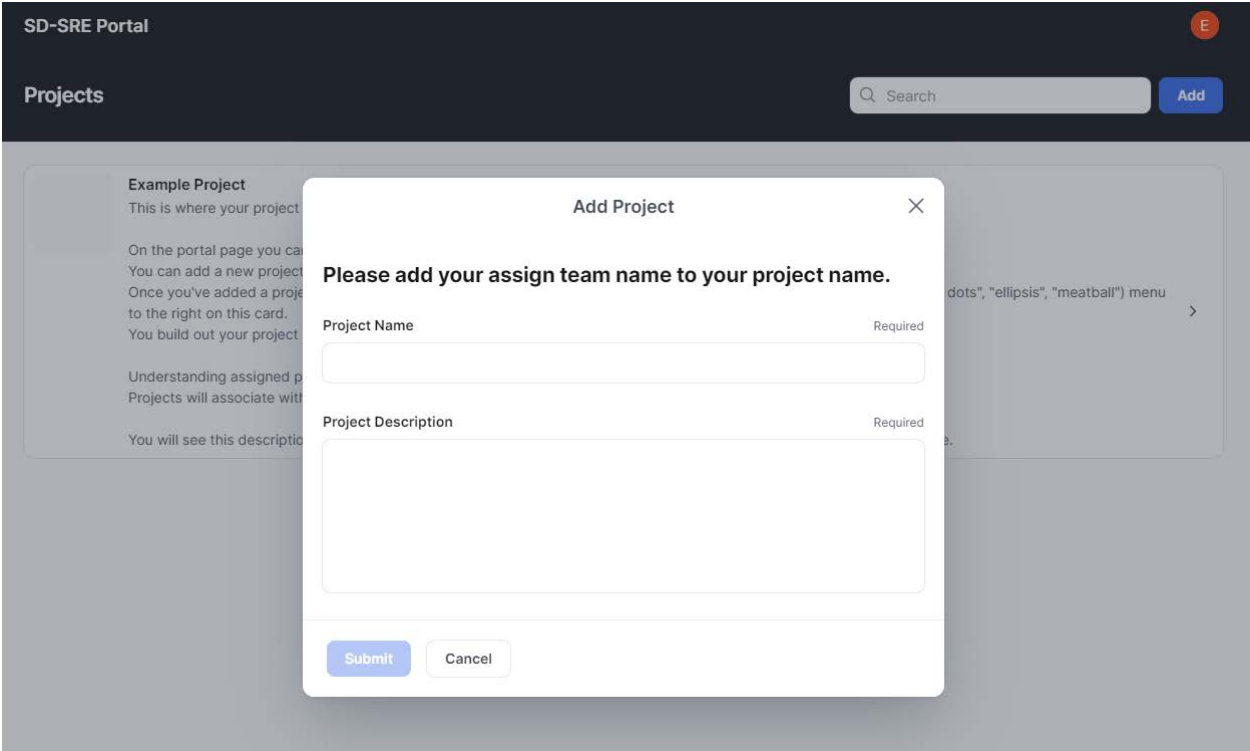
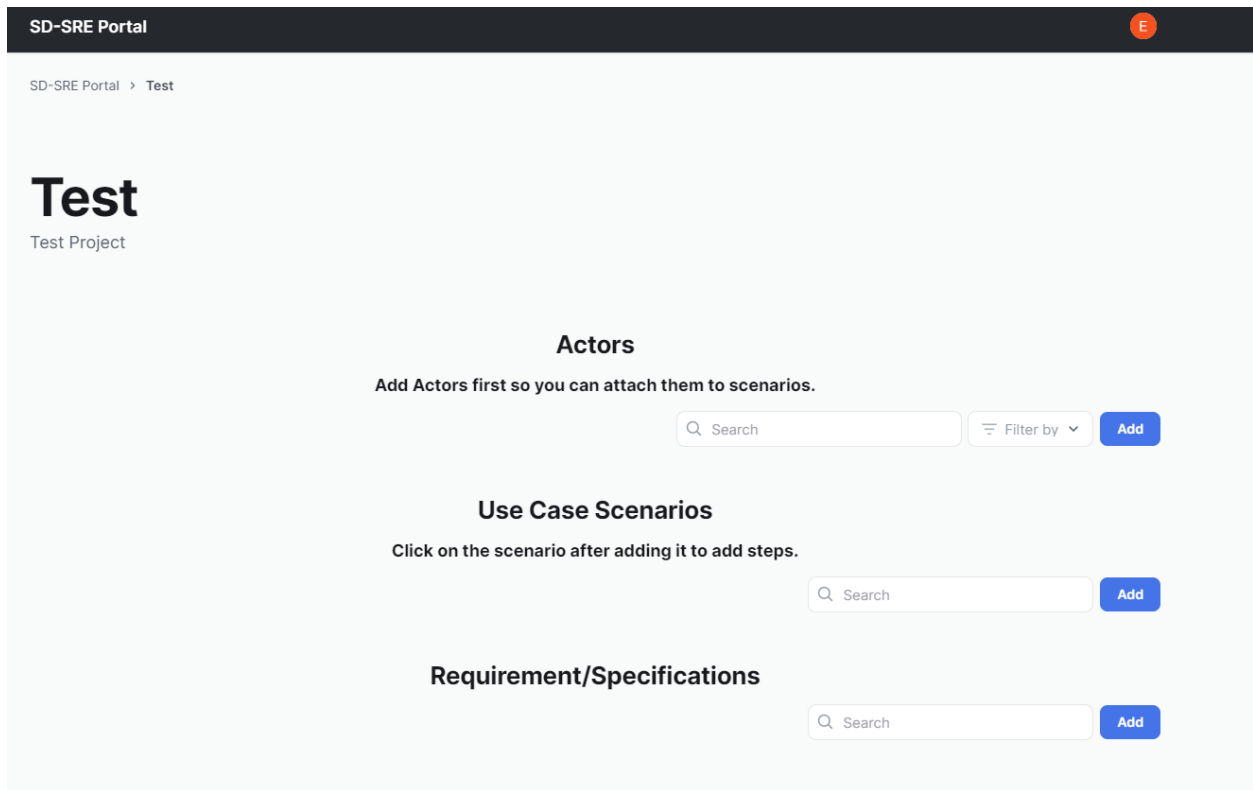


Figure 47: SD-SRE Portal add projects ability

The blank SD-SRE portal when a new project is added is shown below in Figure 48 below. It shows the ability to add actors, use case scenarios, and requirements/specifications.



*Figure 48: SD-SRE Portal blank project*



The ability to add actors is shown below in Figure 49 below. It allows users to indicate how sensitive the actors access is to the proposed system.

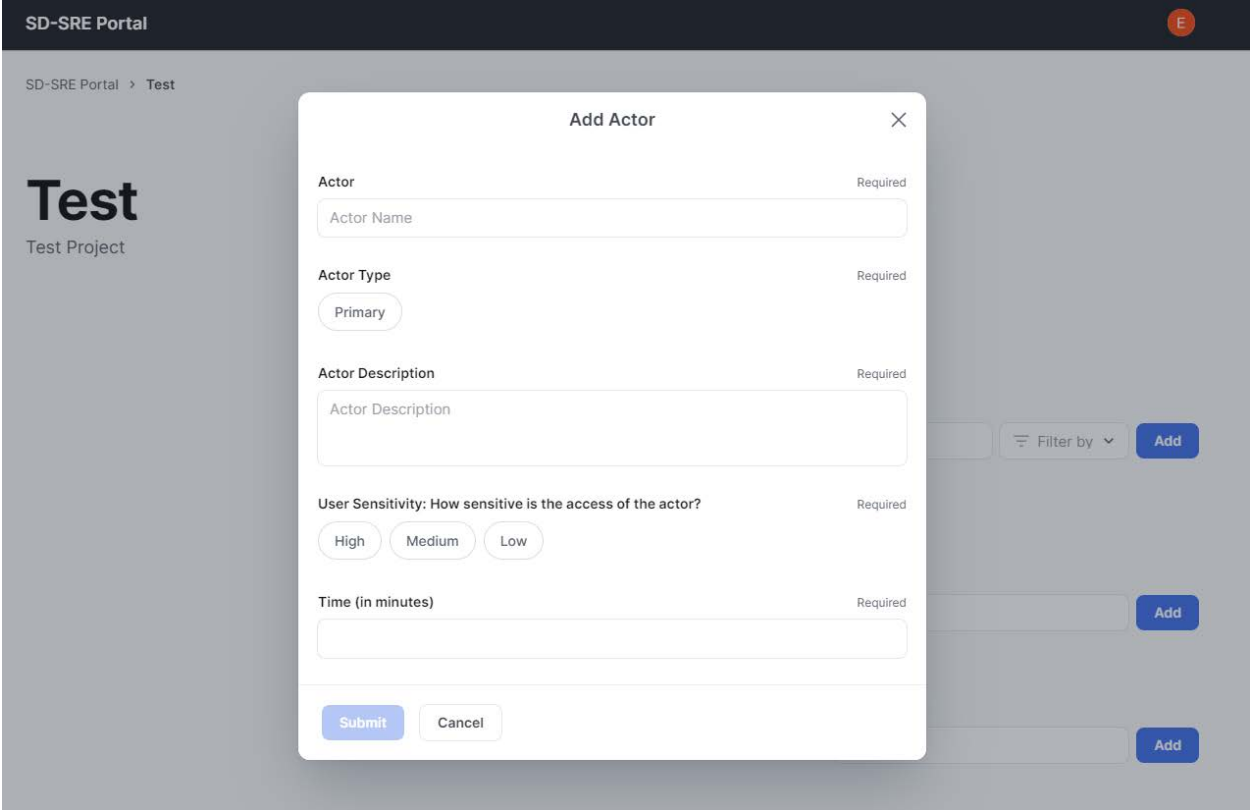


Figure 49: SD-SRE Portal add actors' ability

The ability to add scenarios, attach actors, to detail the preconditions and trigger conditions are shown in Figure 50 below.

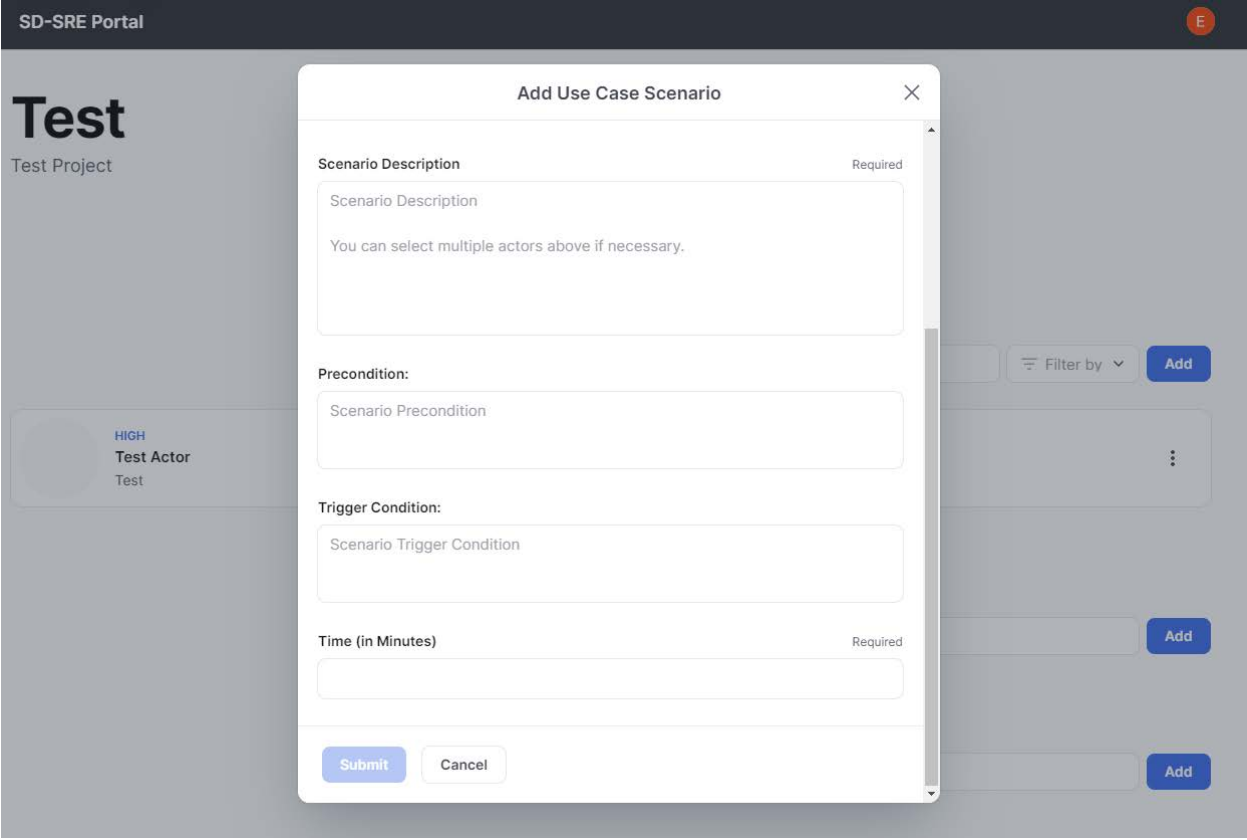
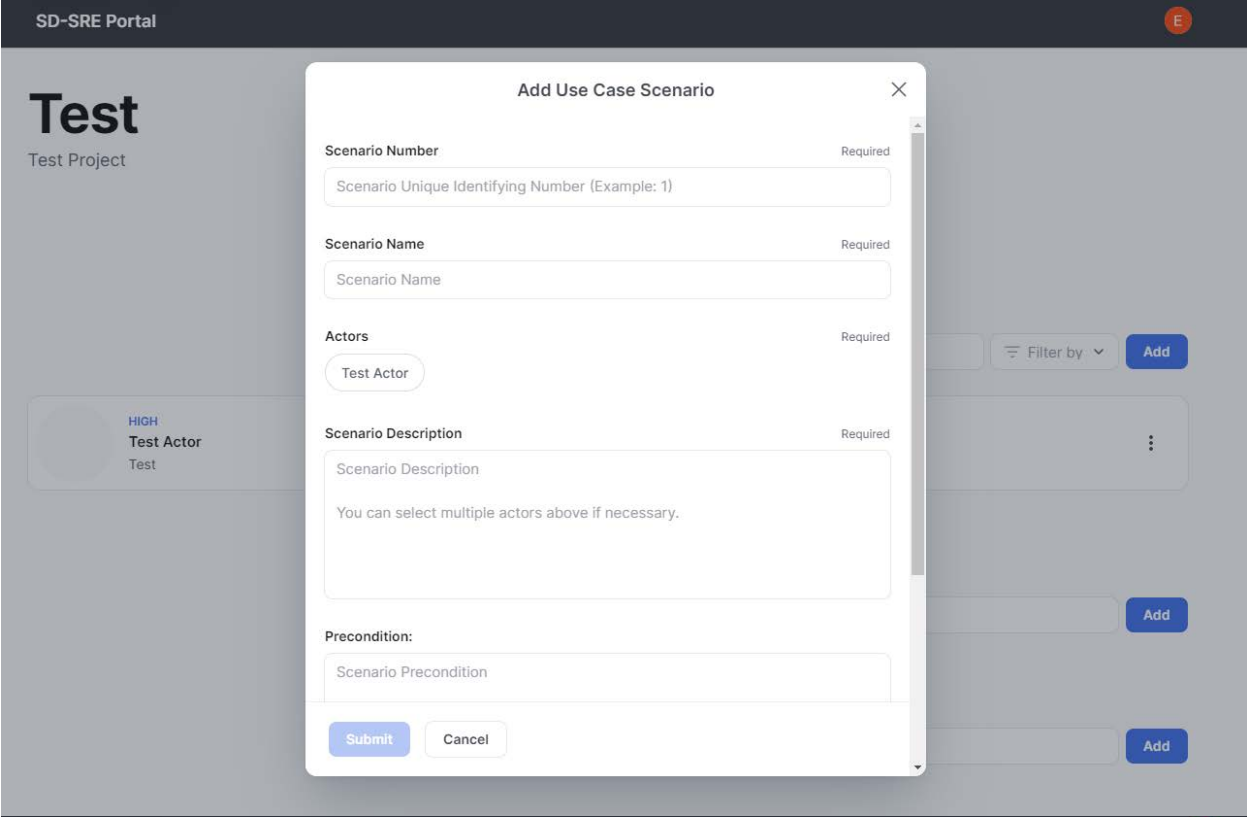


Figure 50: SD-SRE Portal add scenario ability

The ability to add scenario steps is shown in Figure 51 below.

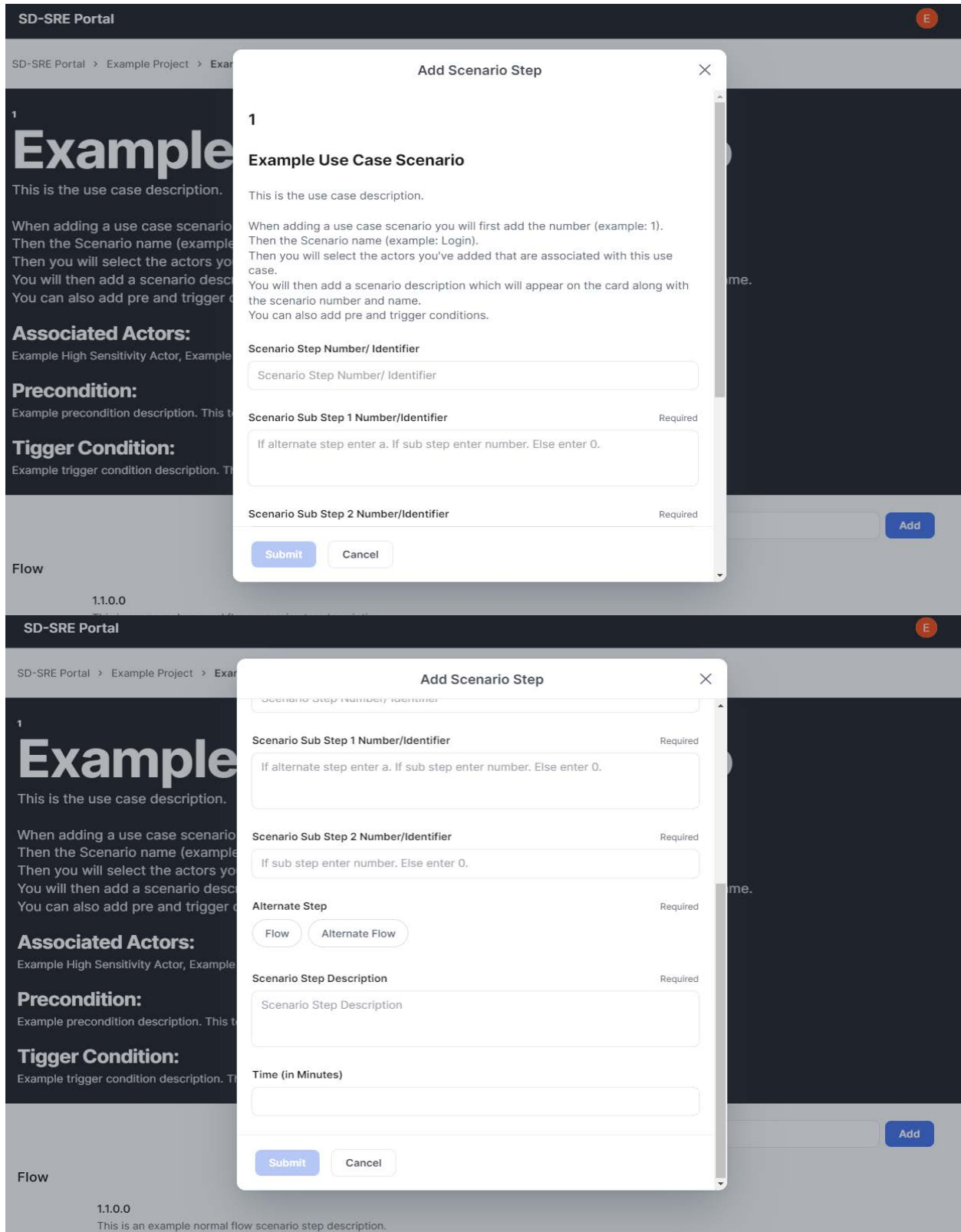
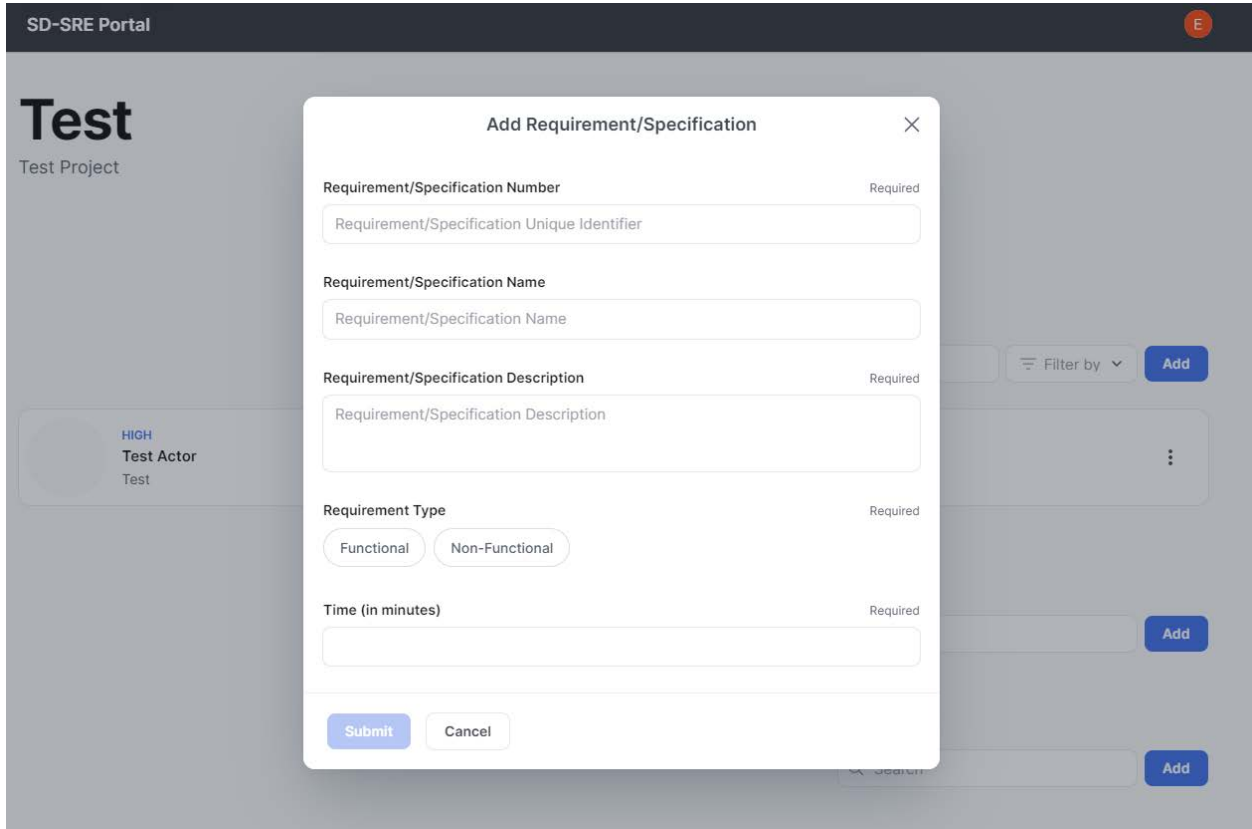


Figure 51: SD-SRE Portal add scenario steps ability

The ability to add requirements or specifications is shown below in Figure 52 as well as the ability for users to detail if the requirement is functional and non-functional. Figure 53 below shows the nonfunctional categories requirements/specifications can be classified as.



The screenshot shows the 'SD-SRE Portal' interface with a modal window titled 'Add Requirement/Specification'. The modal contains the following fields and options:

- Requirement/Specification Number** (Required): A text input field with the placeholder 'Requirement/Specification Unique Identifier'.
- Requirement/Specification Name**: A text input field with the placeholder 'Requirement/Specification Name'.
- Requirement/Specification Description** (Required): A larger text input field with the placeholder 'Requirement/Specification Description'.
- Requirement Type** (Required): Two radio button options, 'Functional' and 'Non-Functional', with 'Functional' selected.
- Time (in minutes)** (Required): A text input field.

At the bottom of the modal are 'Submit' and 'Cancel' buttons. The background shows a 'Test Project' page with a 'HIGH Test Actor Test' card and several 'Add' buttons.

*Figure 52: SD-SRE Portal add requirement/specification ability*

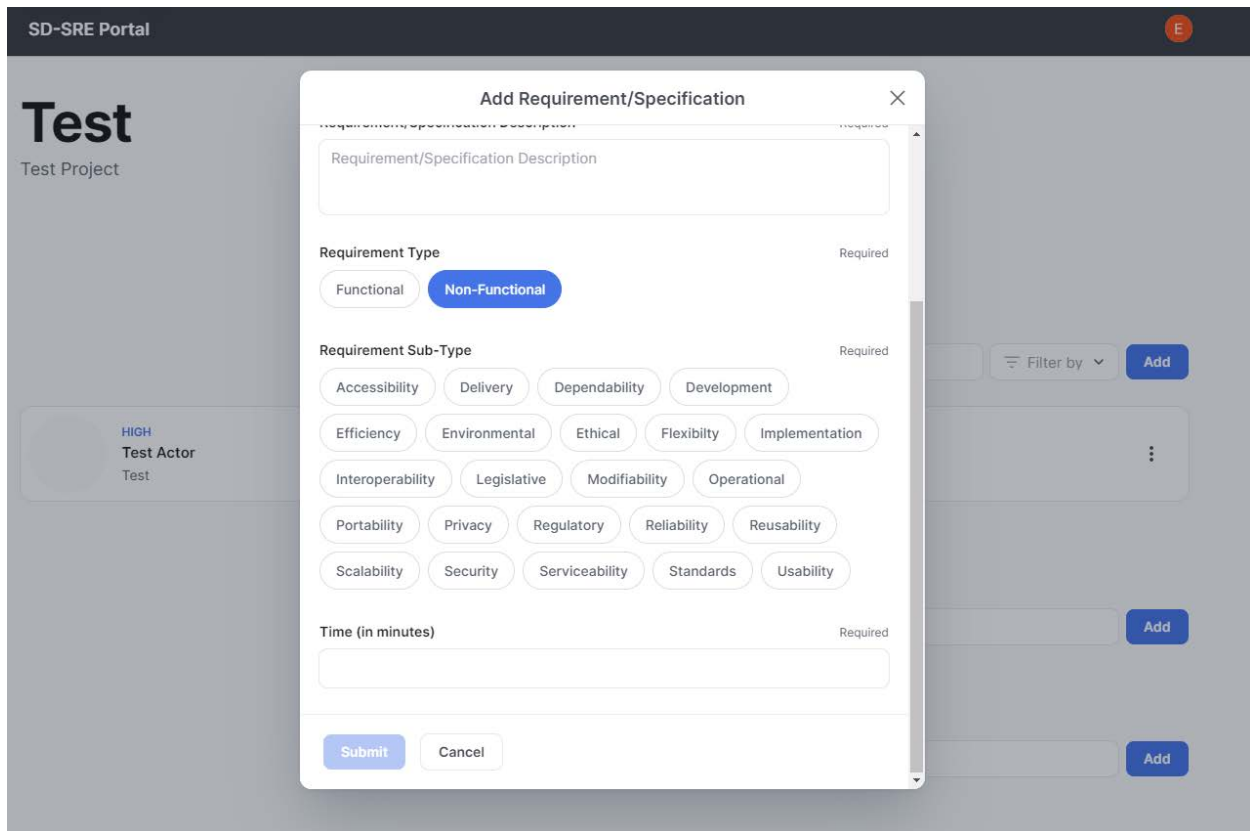
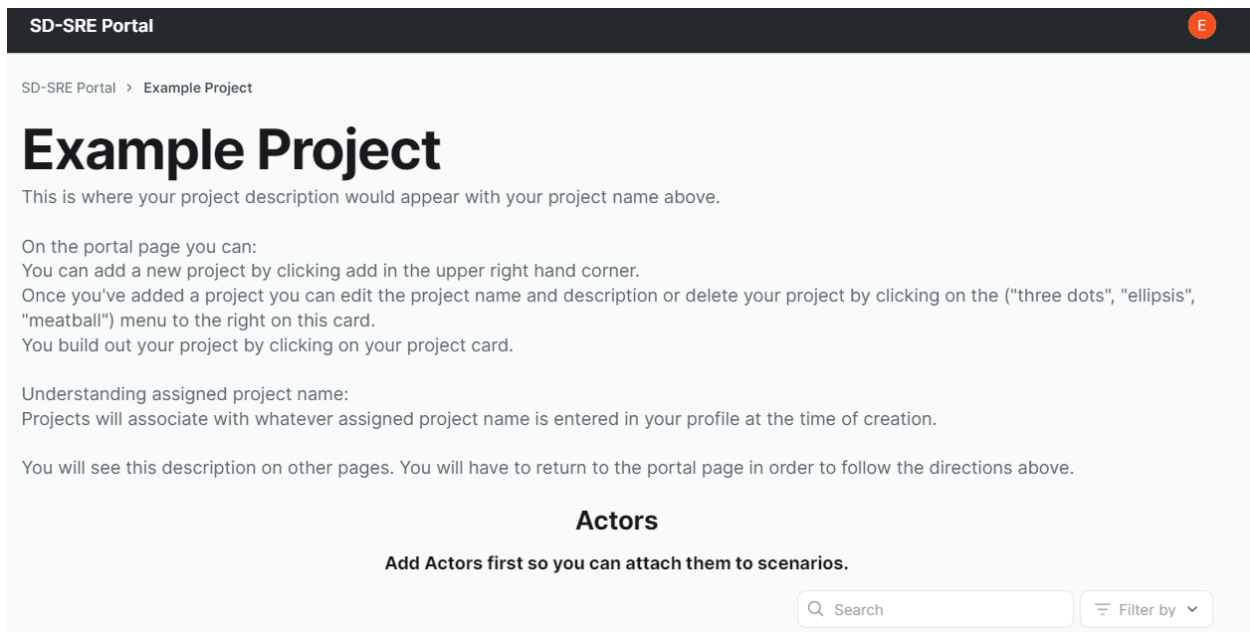


Figure 53: SD-SRE Portal add nonfunctional category requirement/specification ability

Below in Figure 54 is an example of what a system documentation looks like once entered in the SD-SRE portal.



## Actors

Add Actors first so you can attach them to scenarios.

- HIGH**  
**Example High Sensitivity Actor**  
Here is an example of high sensitivity access actor. This actor is likely to be an admin or a manager. >
- MEDIUM**  
**Example Medium Sensitivity Actor**  
Here is an example of a medium sensitivity access actor. This actor is likely to be a contributor or a cashier. >
- LOW**  
**Example Low Sensitivity Actor**  
Here is an example of low sensitivity access actor. This actor is likely to be a reader. >

## Use Case Scenarios

Click on the scenario after adding it to add steps.

- 1**  
**Example Use Case Scenario**  
This is the use case description.  
  
When adding a use case scenario you will first add the number (example: 1).  
Then the Scenario name (example: Login).  
Then you will select the actors you've added that are associated with this use case.  
You will then add a scenario description which will appear on the card along with the scenario number and name.  
You can also add pre and trigger conditions. >

## Requirement/Specifications

### Functional

- 1**  
Here is an example of a functional requirement/specification. The name of the specification is not required but the number is.  
This card shows the requirement/specification number and description.  
You will have to click on the card to see the name of the requirement/specification. >  
  
Requirement sort by requirement sub-type and number (A-Z) but are sorted into two groups: flow and alternate flow.
- 2**  
This is an example functional requirement/specification. >

## Non-Functional

3

Here is an example of a non-functional requirement/specification. The name of the specification is not required but the number is.

This card shows the requirement/specification number and description.

You will have to click on the card to see the name of the requirement/specification.

For non-functional requirements you will also have the ability to attach a non-functional sub-type (example: accessibility, reusability, security). If you select a functional sub-type it will appear below this description on the card.

Requirement sort by requirement sub-type then by number (A-Z) but are sorted into two groups: flow and alternate flow.

Security

4

This is an another example non-functional requirement with an security sub-type.

Security

6

Example requirement/specification to show sorting by sub-type then number.

Security

5

This is an example Non-functional requirement with an usability sub-type.

Usability

*Figure 54: SD-SRE Portal example of instructional project*

Below in Figure 55 is an example of what a system use case scenario documentation looks like once entered in the SD-SRE portal.

SD-SRE Portal E

SD-SRE Portal > Example Project > Example Use Case Scenario

# 1 Example Use Case Scenario

This is the use case description.

When adding a use case scenario you will first add the number (example: 1).  
Then the Scenario name (example: Login).  
Then you will select the actors you've added that are associated with this use case.  
You will then add a scenario description which will appear on the card along with the scenario number and name.  
You can also add pre and trigger conditions.

**Associated Actors:**  
Example High Sensitivity Actor, Example of a Low Sensitivity Actor

**Precondition:**  
Example precondition description. This text is what you entered for precondition.

**Trigger Condition:**  
Example trigger condition description. This text is what you entered for trigger condition.

## Flow

### 1.1.0.0

This is an example normal flow scenario step description.

Scenarios will sort by numbering/identifier (A-Z).

Understanding the numbering:

This is the numbering that will appear on the scenario step card.

A.B.C.D

A= Use Case Scenario Number - This is the small green/white number you see above the scenario name.

B= Scenario number- Entered in scenario step Identifier

C= Scenario 1st sub step: This is often where people identify a sub flow with the letter a- Entered in Scenario Sub Step 1 Identifier

D= Scenario 2nd sub step - Entered in Scenario sub Step 2 Identifier

>

### 1.2.0.0

This is an example of a step 2 of a normal flow of use case scenario 1.

>

### 1.3.0.0

This is an example of a step 3 of a normal flow of use case scenario 1.

>

### 1.4.0.0

This is an example of a step 4 of a normal flow of use case scenario 1.

>

### 1.5.0.0

This is an example of a step 5 of a normal flow of use case scenario 1.

>

### 1.5.1.0

This is an example of a sub step 1 of step 5 of a normal flow of use case scenario 1.

>

## Alternate Flow

### 1.1.a.0

This is an example alternate flow scenario step description.

This example shows that this is an alternate flow step to use case scenario 1 step 1.

Scenarios will sort by numbering/identifier (A-Z).

Understanding the numbering:

This is the numbering that will appear on the scenario step card.

A.B.C.D

A= Use Case Scenario Number - This is the small green/white number you see above the scenario name.

B= Scenario number- Entered in scenario step Identifier

C= Scenario 1st sub step: This is often where people identify a sub flow with the letter a- Entered in Scenario Sub Step 1 Identifier

D= Scenario 2nd sub step - Entered in Scenario sub Step 2 Identifier

>

### 1.1.a.1

This is an example of a sub step 1 of step alternate step 1 of a normal flow of use case scenario 1.

>

### 1.1.a.2

This is an example of a sub step 2 of step alternate step 1 of a normal flow of use case scenario 1.

>

*Figure 55: SD-SRE Portal example of use case scenario*



## 8.2 All Suggested Security Related Requirements

Topic	Sub-Topic	Requirement
Authentication	Authentication	The system shall authenticate users before authorizing users.
Authentication	Authentication	The system shall have a different authentication solutions for internal access and public access.
Authentication	Authentication	The system shall have multi-factor authentication.
Authentication	Least Privileges	The system shall enforce least privileges for users.
Authentication	Least Privileges	The system shall provide minimal privileges to third party systems where required.
Authentication	Third Party	The system shall verify communication from third party systems.
Authentication	Least Privileges	The system shall use a secure access control mechanism for authentication prior to accessing sensitive data.
Authorization	Permissions	The system shall not assign permissions by role.
Authorization	Permissions	The system shall verify a user's access to a feature by user and not role.
Authorization	Permissions	The system shall verify a user's access to data by user and not role.
Authorization	Transactions	The system shall have a second form of authorization for transactions.
Authorization	Access Control	The system shall use an attribute-based access control over a relationship-based access control over a role-based access control.
Authorization	Access Control	The system shall exit safely when authorization checks fail.
Authorization	Access Control	The system shall provide generic failure messages when authorization fails.
Authorization	Tokens	The system shall hash all randomized token values.
Authorization	Permissions	The system shall deny access by default.
Authorization	Permissions	The system shall validate the permissions on every request.
Database		The system shall only allow database access through local socket file or named pipe.
Database		The system shall only bind the database on local host.
Database		The system shall have database servers in a separate DMX isolated from the application server.
Database		The system shall only allow encrypted connections to databases.
Database		The system shall have a trusted digital certificate on the servers.

Database		The system shall store database credentials outside the Webroot, not in the source code, nor in repositories.
Database		The system database shall not have root, sa, or sys accounts.
Database		The system shall not grant account administrative rights over a database instance.
Database		The system shall apply permissions at the table level.
Database		The system shall apply permissions at the column level.
Database		The system shall apply permissions at the row level.
Database		The system shall store the database transactions logs on a separate disk than the main database files.
Database		The system shall block access to underlying tables.
Database		The system shall require access through restricted views.
Database		The system shall regularly X (many suggestions of how often...no set standard) backup the database files.
Database		The system shall limit server-side sessions based on inactivity and timeout.
Email Collection	Email	The system shall ensure emails contain two parts separated by an at symbol (i.e. XXXX@XXX.XXX). Make sure there is an @ and a .domain.
Email Collection	Email	The system shall ensure domain parts of emails can only contain letters, numbers, hyphens, and periods.
Email Collection	Email	The system shall ensure the part before the @ symbol in the email is no longer than 63 characters.
Email Collection	Email	The system shall ensure the total length of the email is not more than 254 characters.
Email Collection	Email	The system shall verify emails by sending an verification email that requires the user to click a link to verify their email.
Email Collection	Email	The system verification email should contain a pseudo secure random token that is: one-time use, time X (usually 24 hours) limited , and at least 32 characters long.
Email Collection		For systems requiring email address collection the system shall check domains against a blacklist of disposable email domains and return invalid email error to user if email is not acceptable.
Encryption		The system shall be encrypted at the application level.
Encryption		The system shall be encrypted at the database level.
Encryption		The system shall be encrypted at the filesystem level.

Encryption		The system shall be encrypted at the hardware level.
Encryption		For symmetric encryption the system shall use AES with keys that are at least 128 bits but ideally 256 bits.
Encryption		The system shall use encryption algorithms that are approved by NIST's algorithmic validation program or similar entities.
Encryption		The system shall use random padding for algorithms that require padding.
Encryption		The system shall generate keys using a secure pseudo-random number generator.
Encryption		The system shall ensure the generated keys are independent of each other (i.e. keys for encrypting data should be different than key-encrypting keys).
Encryption		The system shall rotate encryption keys after X (refer to NIST SP 800-57) crypto period of time.
Encryption		The system shall rotate encryption keys after X (34 GB for 64-bit keys, 295 exabytes for 128-bit keys) amount of data is encrypted.
Encryption		The system shall track data encryption key matches via ID.
Encryption		The system shall store keys in a secure storage mechanism such as HSMs, key vaults and storage APIs.
Encryption		The system shall protect the configuration files containing the keys with restrictive permissions.
Encryption		The system shall store keys in a separate location as the data it is encrypting.
Encryption		The system shall store encryption keys in an encrypted form.
Encryption		The data encryption keys shall be as strong as the key encryption keys.
Encryption		The system shall not reuse cryptographic keys for multiple functions.
File Upload		The system shall limit file upload size.
File Upload		The system shall validate file inputs before validating extensions.
File Upload		The system shall limit file upload extensions. i.e. if expecting an image then only allow image extensions.
File Upload		The system shall limit file upload to authorized users.
File Upload		The system shall store the files on a different server than the application.
File Upload		The system shall store the files outside the Webroot.
File Upload		The system shall run the file through an antivirus.

File Upload		The system shall have a maximum file name length.
File Upload		The system shall restrict characters in the filename.
File Upload		The system shall generate a new file name for storage.
File Upload		The system shall rewrite image files.
File Upload		The system shall not allow zip files.
File Upload		The system shall scan files for viruses prior to saving them.
Input Validation	Input	The system shall sanitize all inputs.
Input Validation	Input	The system shall encode the data when displayed to the user.
Input Validation		The system shall use the allow list approach for input validation.
Input Validation		The system shall escape user data from special characters when allow list approach is not viable.
Input Validation		The system shall enforce correct syntax for structured fields such as dates. i.e. The system shall force formatting of data for known formatted fields such as birth date, SSN/national number, passport number. If there is a known format for a field the system shall facilitate a way for the user to enter it in that manner.
Input Validation		The system shall enforce values are within the expected range.
Input Validation		The system shall use canonical encoding across all input text to validate no invalid characters are present.
Input Validation		The system shall set maximum length of data for all input fields.
Input Validation		The system shall restrict form submissions.
Input validation		The system shall escape all outputs shown to the user.
Logging	Input	The system shall log the action taken and by which user.
Logging	Input	The system shall not log sensitive information in the log but rather that the action taken and by which user.
Logging	Password change	The system shall log the password change action.
Logging	UserID/Username	The system shall log the creation of a new account.
Logging	Login	The system shall log all login failure attempts.
Logging	Login	The system shall log all forgot password request.
Logging	General	The system shall use a firewall.
Logging	Integrity	The system shall not delete log entries. When changes need to be made the old log entry shall be preserved as well as the new log entry.
Logging	Alert	The system shall alert admins of sensitive log occurrences (i.e. authorization failure).

Logging	Expiration time	The system shall keep log records for at least X 450 days. No set standard but a data breach lifecycle is about 300 days on average therefore logs should be kept 1.5 times longer.
Login	UserID/Username	The system shall only allow case insensitive user IDs.
Login	UserID/Username	The system shall only allow unique user IDs/Username.
Login	UserID/Username	For high security systems the system shall assign a secret username that is not based on users public data.
Login	UserID/Username	The system shall not allow sensitive accounts such as system administrators to log in from the front end of the system.
Login	Password	The system shall ensure the password is not blank.
Login	Password	The system shall require passwords with a minimum of 12 characters.
Login	Password	The system shall require passwords with a maximum of 128 (some still recommend 64...depends on hashing implementation) characters limitation. Note that this may be adjusted in the future as hashing algorithms have more capacity.
Login	Password	The system shall require the user to confirm the password by reentering it.
Login	Password	The system shall not truncate passwords.
Login	Password	The system shall allow usage of all characters in passwords including Unicode, whitespaces, and emojis.
Login	Password	The system shall ensure passwords are rotated at least once a year or immediately when passwords are discovered leaked.
Login	Password	The system shall use a password strength meter and present it to the users.
Login	Password	The system shall block commonly used passwords.
Login	Password	The system shall block previously breached passwords.
Login	Password	The system shall not limit or require the number or type of characters a user may use in the password field (i.e. no password composition rule).
Login	Password	The system shall use the stored password hash when doing password check.
Login	Password	The system shall use a secure password function provided by the language or framework the system is implemented in.
Login	Password	The system shall require the user to reauthenticate when updating sensitive information.

Login	Password	The system shall allow users to paste passwords.
Login	Password	The system shall allow the use of external password helpers such as browser password helpers.
Login	Password	The system shall mask/obscure all password entries.
Login	Password	The system shall allow the user to temporarily view the masked/obscured password.
Login	Password	The system shall allow the user to navigate between the username and password field with a single press of the tab key.
Login	Password	The system shall transport passwords over TLS or other strong Transport Layer protocols.
Login	Password Change	The system shall allow users to change their passwords.
Login	Password Change	The system shall ensure the session is active when changing passwords.
Login	Password Change	The system shall verify current password prior to allowing it to be changed. This is different than forgot password.
Login	Password Change	The system shall uphold all password standards for the new password.
Login	Password Change	The system shall send the user an email that the password has been changed.
Login	Password Change	The system shall alert the user of an account password change request and allow the user the ability to lock/disable the account.
Login	Forgot Password	The system shall use a side channel (email, text, pin) to communicate how to reset password.
Login	Forgot Password	The system shall not allow the user to retrieve old password.
Login	Forgot Password	The system shall require the user to confirm the new password by reentering it.
Login	Forgot Password	The system shall require the user to login once password has been reset. The system shall not auto log in the user.
Login	Forgot Password	The system shall ask the user if they want to invalidate their existing sessions when changing their password.
Login	Forgot Password	The system shall alert the user (i.e. send the user an email) that the password has changed.
Login	Forgot Password	The system shall not lock/disable an account due to a forgot password request.
Login	Forgot Password	The system shall not use security questions for account recovery.
Login	Password Storage	The system shall hash all passwords for storage. The system shall not store plain text passwords.

Login	Password Storage	The system shall salt password hashes.
Login	Password Storage	The system shall pepper salted password hashes.
Login	Password Storage	The system shall use a hashing library that accepts all Unicode characters.
Login	Account Access Failure	The system shall provide a generic message when there is a password failure.
Login	Account Access Failure	The system shall provide a generic message when there is a username failure for both existent and non-existent users.
Login	Account Access Failure	The system shall provide a generic message when the account doesn't exist.
Login	Account Access Failure	The system shall return all responses in the same amount of time.
Login	Account Access Failure	The system shall provide a CAPTCHA feature where a generic message cannot be provided after X (usually 3) failed attempts.
Login	Account Access Failure	The system shall provide an account lock or disable the account after X (usually 3) attempts to login failures within X (usually 5) minutes of attempts.
Login	Account Access Failure	The system lockout/disability functions shall count base on the account and not the I.P. address.
Login	Account Access Failure	The system shall provide a generic message when accounts are locked out/disabled.
Login	Account Access Failure	The system shall use an exponential time lockout system instead of a fixed lockout duration.
Login	Account Access Failure	The system shall allow users to unlock/reenable their account through a forgot login functionality.
Login	Account Access Failure	The system shall alert the user of an account log in failure and provide them the option to lock their account.
Login	Login Alert	The system shall alert the user of a login alert and provide them the ability to lock/disable the account.
Login	Session Management	The system shall prompt the user if they are still active once the system has been idle for X (10 min) amount of time and if user does not respond within 30 seconds the system shall log the user out/invalidate the session.
Login	Session Management	When account access information is updated the system shall prompt the user as to whether or not to invalidate all sessions.
Random Number		The system shall use cryptographically secure Pseudo-Random Number Generators.
Sensitive Information		The system shall require the user to reauthenticate when updating sensitive information.
Sensitive Information		The system shall obscure sensitive information.

Sensitive Information		The system shall allow user to unobscure sensitive information temporarily where applicable.
Sensitive Information		The system shall have only obfuscated selector names in public facing code such as CSS.
Sensitive Information		The system shall hide feature lookup ids.
Sensitive Information		The system shall not expose behind the scenes identifiers to the user.

*Table 41: All currently suggestable security related requirements*



## 9 Glossary

Term	Definition
Click Jacking	Click jacking is a UI redress attack where an attacker layers malicious components on a legitimate web page to redirect a user to a malicious site.
Initial Attack Vector	The method used to gain access unauthorized to a system.
Malicious Insider	An actor with insider information of an organization misused to cause harm, often an employee.
Phishing	A social engineering attack that send a person a message to trick them into giving sensitive information.
Pharming	A social engineering attack that uses a fake website meant to mimic a real website that tricks a user into giving sensitive information.
Smishing	A social engineering attack that send a person a text (SMS) message to trick them into giving sensitive information.
Tab Nabbing	A social engineering attack that falls under the phishing category that allows an attacker to redirect a victim to a duplicated malicious site when they click away from an open tab.
Token-side Jacking	A man in the middle attack where an attacker has stolen a token and used it to impersonate a user.
Vishing	A social engineering attack that send a person a phone call to trick them into giving sensitive information.
Zero Trust	An approach to cybersecurity that eliminates implicit trust and requires constant revision of trust.

## 10 Index

- accuracy, 39, 41, 48, 75, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 122
- attack vector, 3
- AUC. *See* roc
- authentication, 66, 80, 81, 113
- authorization, 66, 82, 83, 113
- brute force, 13
- cyberattacks, 1
- data breach, 2
- database, 66, 92, 93, 113
- denial of service
  - DOS, 5, 6
- distributed denial of service
  - DDOS. *See* : denial of service
- email collection, 66, 84, 113
- encryption, 66, 88, 89, 113
- F1 score, 51
- fall-Out, 51
- false negative, 48, 75
- false positive, 48, 51, 52, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97
- file upload, 66, 94, 95, 113
- input validation, 66, 86, 87, 113
- logging, 66, 96, 97, 113
- login, 66, 76, 77, 78, 113
- man in the middle, 13
- miss rate, 52
- ontologies, 14
- OWASP, 24
- pharming, 2
- phishing, 2
- precision, 50
- random number, 66, 90, 113
- recall, 50
- roc, 52
- semantic, 30
- semantic web technologies, 30
- sensitive information, 66, 78, 113
- smishing, 2
- specificity, 51
- tech support fraud, 2
- true negative, 48, 49, 51
- true positive, 48, 50, 52, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97
- verification, 17
- vishing, 2, 164
- zero trust, 4