INL/CON-24-76242-Revision-0

Root Cause Correlation Analysis of Software Failures via Orthogonal Defect Classification and Natural Language Processing

June 2024

Edward Chen, Han Bao, Tate H Shorthill

INL/CON-24-76242-Revision-0

Root Cause Correlation Analysis of Software Failures via Orthogonal Defect Classification and Natural Language Processing

Edward Chen, Han Bao, Tate H Shorthill

June 2024

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Root Cause Correlation Analysis of Software Failures via Orthogonal Defect Classification and Natural Language Processing

Edward Chen^{1,*}, Han Bao^{1}, Tate Shorthill^{2}

^{1}Idaho National Laboratory, Idaho Falls, ID; ^{2}Department of Mechanical Engineering and Materials Science, University of Pittsburgh, Pittsburgh, PA

[leave space for DOI, which will be inserted by ANS – do not highlight this line]

ABSTRACT

Systems theoretic process analysis (STPA) is becoming an increasingly popular technique to assess how complex digital software systems can fail. Rather than defining failures by their observable failure events, which may be sparse especially for safety rated nuclear digital instrumentation and control systems (DI&C), failures are defined as postulated unsafe actions under specific contextual conditions. This permits a top-down analysis of system hazards and identifies whether imposed constraints and requirements can sufficiently address undesirable hazards. However, STPA is a qualitative approach at identifying inadequacies in the development process and cannot currently be used to quantify unsafe action likelihoods for probabilistic risk assessment. Therefore, in this work, we examine the root causes of software failure and explore whether a consistent correlation can be linked to specific unsafe action classes. We implement Lbl2Vec, an unsupervised document classification and retrieval algorithm, on a database of 4,096 software defect reports acquired from various open-source software systems. By analyzing sentence structure, embedded labels, and word vectors, we show that certain defect types positively correlate to specific unsafe action classes over others. The correlations developed can be used to estimate the failure probability of safety intended DI&C systems which provides a licensing basis for nuclear plant modernization efforts.

Keywords: Probabilistic risk analysis, root cause analysis, software reliability, failure, natural language processing

1. INTRODUCTION

Digital instrumentation and control systems (I&C) are critical for the economic viability of nuclear power plants as traditional analog instrumentation is becoming increasingly obsolete. For existing nuclear power plants looking to modernize their I&C systems, a license amendment request is required pursuant to 10 CFR 50.90 [1]. As part of the request, a licensee must demonstrate the no significant hazard consideration which specifies that a modification to the facility does not (1) involve a significant increase in the probability or consequences of an accident previously evaluated, (2) create the possibility for a new or different kind of accident from any previously evaluated, or (3) involve a significant reduction in the margin of safety [1]. From this perspective, probabilistic risk assessment is a critical methodology used for the licensing and safety analysis of nuclear power plants.

Whereas conventional fault tree analysis (FTA) identifies the inability to perform a function as failure basic events of hardware and analog components, software failure basic events must be treated differently as how they fail differs significantly. First, software does not fail randomly like hardware; software is a design abstraction of function without physical realization and thus may contain failures beyond an inability to

*edward.chen@inl.gov

perform [2]. To address this issue, systems-theoretic process analysis (STPA) was proposed and introduced the concept of unsafe control actions (UCAs) as failures of systems. A UCA is, in essence, an inadvertent unsafe misbehavior of a system (or interaction) that can lead to an undesirable loss [3] and is one form of software failure. STPA analysis has been used to evaluate the safety of digital I&C systems in nuclear power plants and has shown promise in qualitative failure identification [2]. Integration of STPA into FTA has also been completed in both HAZCADS [4] and RESHA [5, 6], which are two notable methodologies that attempt to model software failures for risk analysis. In both methodologies, it has been shown that STPA is effective at qualitatively identifying faults and misbehaviors in digital I&C systems [4]. However, while software failures are identified and may be used for qualitative analysis, they cannot be fully applied for the no significant hazard consideration as quantification of software failure basic events is not within the scope of either methodology. Without failure quantification, it is difficult to justify a digital I&C system upgrade does not involve a significant increase in the probability or consequences of an accident previously evaluated. Preliminary work for the quantification of software-based basic events was completed in the ORCAS methodology [7] for reliability analysis in nuclear digital I&C systems. However, several major limitations reduced the efficacy of the methodology, namely, (a) significant human subjectivity and bias in the determination of root cause causality to software failures and (b) the limited number of data samples to support root cause causality to software failures.

Therefore, in this work, we aim to improve the quantification of STPA identified software failure events to support licensing and risk analysis of digital I&C systems. We examine a large database of software defects and assess whether a consist relationship exists between defects and inadvertent unsafe actions made by digital systems. The contribution of this work includes the development and improvement of a software failure correlation table from a large dataset of publicly available software defect reports. This correlation table can then be used to estimate the failure probability of software basic events in FTA. A natural language processing model (i.e., Lbl2Vec) is demonstrated to automate the analysis of software defect reports to reduce human bias and subjectivity in document processing. Based on the results, this work also discusses the major challenges in utilizing root cause analysis for software failure probability quantification.

2. METHODOLOGY

The overall methodology for dataset preprocessing, model construction, and performance evaluation can be seen in Figure 1. In the figure, dashed boxes are inputs and outputs of the methodology, arrows are processes between steps, and solid boxes are intermediate step outcomes. Each box has a label (A through H) that describes what is required and completed by a process arrow. Boxes A and D are related to the preprocessing of defect reports and construction of training datasets; boxes B, C, and E are related to model construction and development; and boxes G, F, and H are related to model efficacy evaluation. Processes are labeled via their start and end connecting boxes (e.g., AB) and describe what was completed between steps. Each box and process are described in the following sections below.

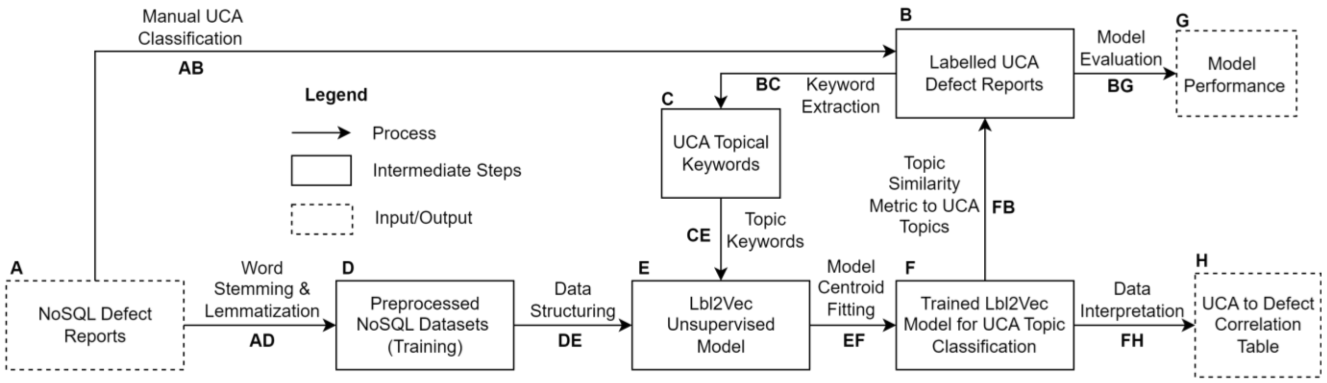
Manual UCA Classification
AB
Legend
Process
Intermediate Steps
Input/Output
B
Labelled UCA Defect Reports
Model Evaluation
BG
G
Model Performance
C
BC
Keyword Extraction
UCA Topical Keywords
CE
Topic Keywords
Topic Similarity Metric to UCA Topics
FB
A
NoSQL Defect Reports
Word Stemming & Lemmatization
AD
D
Preprocessed NoSQL Datasets (Training)
Data Structuring
DE
E
Lbl2Vec Unsupervised Model
Model Centroid Fitting
EF
F
Trained Lbl2Vec Model for UCA Topic Classification
Data Interpretation
FH
H
UCA to Defect Correlation Table

Figure 1. Overall model development and evaluation workflow.

2.1. Dataset Construction and Preprocessing

Four different datasets were used to evaluate and train the Lbl2Vec model. Three defect datasets were retrieved from reference [8] which represent three popular NoSQL database management software systems: MongoDB, Cassandra, and HBase (Box A). The named datasets contain 1,618, 1,095, and 1,383 defect reports, respectively. These datasets were selected as the data have already been preprocessed to a certain extent. Specifically, defect reports are labeled based on (a) at which point during the software development life cycle each defect was detected, (b) which orthogonal defect class each defect most likely belongs to based on expert interpretation, and (c) how the software function was impacted. An orthogonal defect class identifies which major category a defect belongs based on common defect characteristics and is used by companies such as IBM [9] to conduct defect trend analysis. In this work, it permits the development of a correlation between defect class and UCA class.

One additional dataset, named OpenPilot (found at https://github.com/commaai/openpilot), is used to validate the trained language model (Box F). OpenPilot is a popular open-source vehicle autopilot software that automatically drives a vehicle based on video imagining collected from a dashboard camera. The variation in software function across the three datasets is desirable as one of our fundamental assumptions is that the developed defect-correlation table is function agnostic. The dataset from OpenPilot is obtained using an open-source web crawler (found at https://github.com/ibrr1/GitHub-Issues-Crawler-Tool). The Github Issue Crawler Tool is an automatic data collection tool that traverses issues from any Github repository and processes them into text files. Each text file contains developer labels that identify what type of defect is reported (e.g., bug, enhancement, cleanup, docs, and controls). Unlike the defect reports from the NoSQL datasets, the collected OpenPilot defect reports do not contain orthogonal defect class labels and must be manually labeled. Orthogonal defect classification of OpenPilot defect reports follows the work presented in [9] and [10]. Of the 2,274 closed issue reports, 107 contain the label "bug" which forms the testing dataset. Only bug-labeled reports are used due to their clear relevance to software functions. In Table I, the distribution of orthogonal defect class per dataset is presented. Column 2 shows the total number of defect reports per dataset; columns 3 through 7 show the orthogonal defect classes per dataset; and column 8 shows the number of defects that were either not classified or determined to be not relevant.

Table I. Distribution of orthogonal defect classes per dataset.

Dataset	# of Reports	Algorithm	Assignment	Function	Check	Timing	Other
Hbase	1383	855	51	193	110	11	163
MongoDB	1618	864	69	226	177	27	255
Cassandra	1096	645	49	124	120	4	153
OpenPilot	107	63	4	14	11	2	13

Technically, the raw defect reports may be used to train the language model but will result in a poor topic classification as word choice in the reports varies significantly. Stemming and lemmatization (Process AD) is a document preprocessing methodology that trims words to their roots by removing pluralization, tenses, and suffixes. For example, one common stemming conducted in the reports is the conversion of "correctly" to "correct." In this work, the Porter stemming algorithm is used [11]. Stemming and lemmatization does not fundamentally change the content or meaning of the defect report and is intended to improve model performance. This process is critical for relevant model development as developers may also use wordings that are synonymously equivalent but semantically different to describe what was wrong with the software.

After stemming and lemmatization, the data is fed into the Lbl2Vec model (Process DE) as a list of text files. White space characters (e.g., spaces, newline characters) are not stripped from the text files. All NoSQL defect reports are used in the training process of the language model. Only the OpenPilot defect reports are used to test and evaluate the efficacy of the model.

2.2. Lbl2Vec Language Model

Lbl2Vec [12] (Box E) is an unsupervised document clustering technique that works by creating and relating jointly embedded word, document, and label vectors to user-specified topics and keywords. Documents that share sufficient sematic similarity through embedded vectors are thus classified by topic. Since all learned embeddings share the same feature space, their distance can be considered their semantic similarity. To learn a label embedding, all document embeddings that are close to the descriptive keyword embedding of a topic are clustered together to compute a topic centroid (Process EF) [12]. Distance from a topic centroid determines the similarity metric of a document and ranges from 0 to 1. Lbl2Vec uses the cosine similarity metric to determine how similar a document is to a topic centroid [12]. This method of automated document topic classification reduces the need for annotation costs and can expediate document processing [12]. Lbl2Vec is optimal for our use case as defect reports are information dense but require significant expert knowledge and time to interpret. In addition, each defect report describes a unique software defect that is laden with contextual and environmental information about the system.

The output of Lbl2Vec is an array of similarity metrics for each user-defined topic ranging from 0 to 1, where 0 represents no similarity, and 1 represents perfect similarity (Process FB). In Table II, a sample of the output of Lbl2Vec similarity metric per topic is shown. In column 1, the defect report number is reported; column 2 shows the most likely topic as deduced by Lbl2Vec; and columns 3 through 6 show the similarity metric for each user-defined topic. Lbl2Vec decides the most likely topic by taking the max similarity across all topics. As such, certain outcomes that we have considered may be possible. First, for report 1 in Table II, this is the optimal outcome where (a) the similarity difference between topics is significant such that the model has not confused the document type with other topics, and (b) the confidence in a single topic category is high. Report 2 similarity metrics is another observed outcome. This case is where the model has high confidence in two conflicting topics (i.e., UCA-A and UCA-B). The choice for UCA-B was chosen by the model as it has the highest value; however, we consider this result inadequate as the difference to the nearest topic is too small. The last case, report 3, is where the topic classifications are all low even though UCA-C seems to be the most likely topic and has a higher similarity metric than all other topics. This result is also unacceptable because although UCA-C has achieved a higher similarity metric than other topics, its similarity to the topic itself is inadequate. Therefore, in this work, a topic is considered of that class if it exceeds (a) a similarity metric greater than 0.5, and (b) the difference between the selected topic and the nearest topic is greater than 0.25. The value 0.5 was selected as it indicates that the report is more likely than not to be part of that topic class. A difference of 0.25 was selected as it represents the coincidence probability that any class is selected and suggests sufficient difference to other topics. The entire process for UCA topic classification by Lbl2Vec is as follows:

- Determine the topic similarity metric for all defect reports in the dataset.
- Eliminate all defect report classifications that have similarity metrics in all classes lower than 0.5.
- Eliminate all defect report classifications that have a difference in similarity lower than 0.25.
- Use remaining defect report classifications for model evaluation.

Table II. Sample similarity metrics derived from Lbl2Vec on NoSQL defect reports.

Report #	Likely Topic	UCA-A	UCA-B	UCA-C	UCA-D
1	UCA-A	0.853	0.263	0.216	0.332
2	UCA-B	0.563	0.610	0.362	0.253
3	UCA-C	0.032	0.015	0.361	0.086

2.2.1. STPA Classification of Orthogonal Defects and Keyword Extraction

While knowledgeable experts may be able to classify defects into UCA classes, the process is time consuming and inefficient. Lbl2Vec is used to automate the classification of defect reports into the four UCA classes most closely resembled. In this case, the four UCA classes [3] listed below are the target topic groups:

- UCA-A: Not providing a control action leading to a hazard
- UCA-B: Providing a control action leading to a hazard
- UCA-C: Providing a control action that is too early, too late, or in the wrong order
- UCA-D: Providing a control action that lasts too long or is stopped too soon.

For each UCA topic class, selection of keywords is critical for the correct clustering of defect reports within the model. Keyword selection is based off the word frequency discovered for each defect report. First, from the NoSQL defect database, 100 randomly selected reports are chosen which are then reviewed by two separate researchers (referred to a R1 and R2) familiar with STPA and systems analysis (Process AB). Each researcher independently selects a UCA class that they believe the defect has caused. UCA class selection is based on how the developer describes the software failure and how it has impacted users, functional, or non-functional requirements. Of the 100 selected reports, 85 reports had concurrent UCA selections between the two researchers (Box B). We used Cohen's Kappa (k) to analyze inter-rater agreement [13]. Cohen's Kappa measures the agreement between two raters that classify items in mutually exclusive categories and is defined in Equation (1). Here p_0 is the relative observed agreement between raters, and p_c is the probability of agreement by chance. Cohen's Kappa may be interpreted as: *poor* when $k < 0$, *slight* when $0 \leq k < 0.2$, *fair* when $0.2 \leq k < 0.4$, *moderate* when $0.4 \leq k < 0.6$, *substantial* when $0.6 \leq k < 0.8$, and *almost perfect* when $0.8 \leq k < 1$ [13]. A negative Kappa implies disagreement between classifications. Kappa will also be used to interpret efficacy of the Lbl2Vec language model at UCA topic classification. In Table III, the UCA classification results made by R1 and R2 are presented. The determined Kappa for each dataset is provided in column 2, while in columns 3 through 6, the number of matching UCA classifications by both researchers is shown. In column 7, the total number of matching classifications is shown; while in column 8, the number of defects reports from each dataset is presented. All Kappa values are *substantial* which further suggests that a consistent correlation between defects and UCA class exists.

$$k = \frac{p_0 - p_c}{1 - p_c} \tag{1}$$

Table III. Distribution of defect report classifications per dataset.

Dataset Name	Kappa	UCA-A	UCA-B	UCA-C	UCA-D	Matching	Sample Size
Hbase	0.79	10	19	1	3	33	39
Cassandra	0.85	6	14	3	1	24	27
MongoDB	0.76	10	7	10	1	28	34
Total	0.80	26	40	14	5	85	100

From the matching UCA classifications of defect reports, we analyzed the word frequencies used by developers to extract keywords for Lbl2Vec (Process BC). Conjunctions, prepositions, and words shorter than two letters were filtered from the frequency analysis. In Figure 2, a histogram of word frequencies per UCA class are shown. Words that appear only once are trimmed from the histogram as they are deemed unlikely to be keywords. Words that appear in multiple classifications are assigned as keywords to the class that has the highest occurrence frequency. In the case of "nullpointexception," it is assigned to UCA-B as a keyword as it appears more frequently than for UCA-A. Ten words based on frequency of occurrence from each histogram are selected as UCA topic keywords (Process CE) and shown in Table IV (Box C).

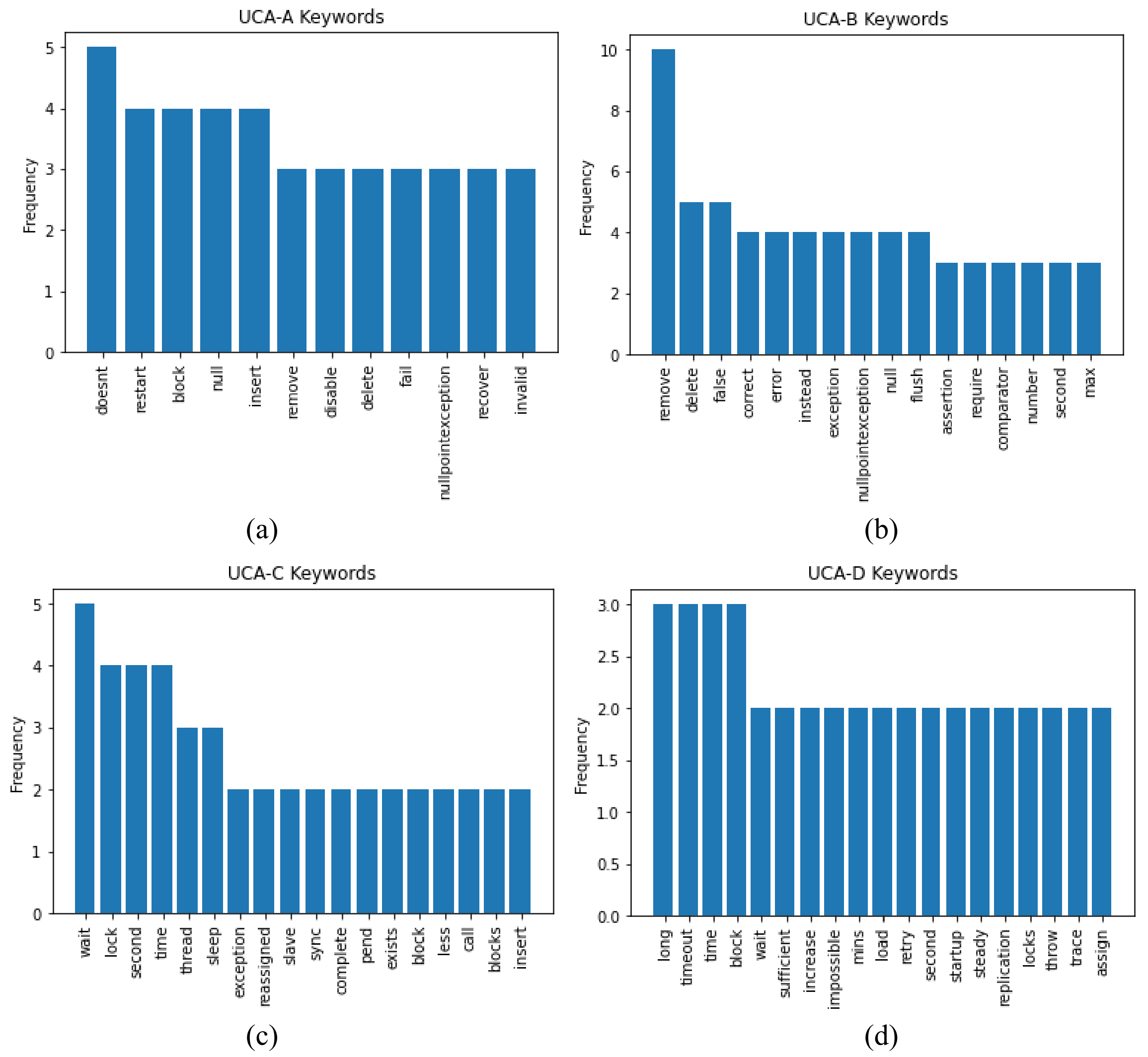

Figure 2. Defect description word usage frequency histograms for keyword extraction for (a) UCA-A, (b) UCA-B, (c) UCA-C, and (UCA-D) labels.

Table IV. UCA class topical keywords selected for Lbl2Vec model development.

Topic	**Keywords**
UCA-A	doesnt, restart, block, null, insert, disable, fail, recover, invalid, miss
UCA-B	remove, delete, false, correct, error, instead, exception, nullpointexception, flush, assertion
UCA-C	wait, lock, second, time, thread, sleep, exception, reassigned, slave, sync
UCA-D	long, timeout, block, sufficient, increase, impossible, load, retry, replication, trace

3. RESULTS

3.1. Evaluation of Model Performance on Training and Testing Datasets

The trained Lbl2Vec is first applied to the NoSQL datasets presented in Table III to calculate a similarity metric for each defect report to UCA topics. We compare the document topic classification to the classifications derived by researchers R1 and R2 with the Lbl2Vec classifications shown in Table V. In column 2, the Kappa value between the researchers and the Lbl2Vec is presented. In columns 3 through 6, the Lbl2Vec UCA classifications are shown. In column 7, the number of matchings classifications made by Lbl2Vec to the total number of matching classifications made by researchers R1 and R2 is shown. The last row of the table shows the overall Kappa and the overall number of matching UCA classifications to the classifications made by the researchers. For instance, column 3, row 5 shows that of the 26 matching UCA-A classifications made by the researchers, 19 of them made by Lbl2Vec also matched. While the Kappa value is lower than those made by the researchers, an overall Kappa of 0.702 still implies *substantial* agreement. This suggests the model is somewhat capable of correctly determining UCA classes that are concurrent with human interpretation.

Table V. Lbl2Vec UCA topic classifications and Kappa value.

Dataset Name	Kappa (k)	UCA-A	UCA-B	UCA-C	UCA-D	Matching/Total
Hbase	0.636	7	15	1	1	24/33
Cassandra	0.667	4	11	3	0	18/24
MongoDB	0.809	8	6	10	0	24/28
Total	0.702	19/26	32/40	14/14	1/5	66/85

Applying Lbl2Vec on the entire NoSQL dataset, we derive a UCA classification for all defect reports. Of the 4,096 defect reports, 570 were determined to be inapplicable as (a) the topic similarity metric was less than 0.5, or (b) the difference to the next closest class was less than 0.25. The remaining 3,526 classifications were split as follows: 1,673 defect reports were labeled as UCA-A, 1,253 were labeled as UCA-B, 419 were labeled as UCA-C, and 181 were labeled as UCA-D. For each UCA classification, we examined the frequency that they were also labeled as a specific orthogonal defect class. In the case of the 1673 UCA-A labeled defect reports:

- 739 belonged to the algorithm defect class.
- 255 belonged to the assignment defect class.
- 800 belonged to the checking defect class.
- 562 belonged to the function defect class.
- 95 belonged to the timing defect class.

These values were then used to calculate the correlation between a UCA-A type failure given an algorithm defect exists in the software. This was applied to all orthogonal defect classes for all UCA classes. The final correlations are shown in Table VI as the defect-UCA correlation table. A comparison is made between the manually derived by the researchers and the Lbl2Vec classified UCA correlation table. For each UCA column, the two values are placed side-by-side. Of the 20 defect-to-UCA correlations, nine Lbl2Vec derived correlations matched researcher derived correlations within a margin of ± 0.05, five were greater, and six were lower. While not all values match, this does imply that Lbl2Vec is incapable and inaccurate. Instead, as the dataset used by Lbl2Vec to derive the correlation was larger than those used by the researchers, it simply implies that the Lbl2Vec correlations are more data informed.

Table VI. Defect-UCA Correlation table derived from manual vs. Lbl2Vec classification.

Orthogonal	UCA Class Correlation, $P(UCA_X \mid \text{defect})$							
Defect Class	UCA-A		UCA-B		UCA-C		UCA-D	
$F(\text{defect})$	Manual	Lbl2Vec	Manual	Lbl2Vec	Manual	Lbl2Vec	Manual	Lbl2Vec
Algorithm	0.331	0.442	0.137	0.098	0.339	0.321	0.194	0.138
Assignment	0.284	0.153	0.648	0.504	0.057	0.101	0.011	0.039
Checking	0.305	0.478	0.262	0.338	0.241	0.139	0.191	0.043
Function	0.385	0.336	0.240	0.423	0.240	0.109	0.135	0.132
Timing	0.189	0.057	0.054	0.083	0.622	0.759	0.135	0.101

Table VI is analyzed as follows for UCA-A probabilities; first, given the occurrence probability of a specific defect class (e.g., algorithm) remaining in the software after software development life cycle activities, the probability of a UCA-A is the matrix product between the defect class occurrence probability and the defect-UCA-A correlation. The full equation for deriving UCA-A event probabilities can be seen in Equation (2).

$$P(\mathrm{UCA_A}) = \begin{bmatrix} P(\mathrm{UCA}_A|\mathrm{Alg.}) \\ P(\mathrm{UCA}_A|\mathrm{Asi.}) \\ P(\mathrm{UCA}_A|\mathrm{Fnc.}) \\ P(\mathrm{UCA}_A|\mathrm{Chk.}) \\ P(\mathrm{UCA}_A|\mathrm{Tim.}) \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} F(Alg.) \\ F(Asi.) \\ F(Fnc.) \\ F(Chk.) \\ F(Tim.) \end{bmatrix} \tag{2}$$

Lbl2Vec is also applied to the OpenPilot dataset to validate that the correlations are consistent across different software functions. Note that as the OpenPilot dataset only contains 107, the results will have variations in a similar manner to the manually derived correlation table. In Table VII, the correlations derived for all three approaches are shown: labeled as "Man" for manual classifications by the researchers, "NoSQL" for Lbl2Vec on the NoSQL datasets, and "OP" for Lbl2Vec on the OpenPilot dataset. For the OpenPilot correlation columns, many of the correlations are zero as Lbl2Vec did not classify any defects into those UCA classes. This does not imply Lbl2Vec is incapable of UCA class identification, only that potentially the OpenPilot dataset was too small for those classes. We examined the exact Lbl2Vec classifications on the OpenPilot dataset to try to understand this result (see **Error! Reference source not found.**).

Table VII. Defect-UCA correlation table derived via manual interpretation, Lbl2Vec classification on the NoSQL datasets, and Lbl2Vec on the OpenPilot dataset.

Defect Class	**UCA Class Correlation**											
	UCA-A			**UCA-B**			**UCA-C**			**UCA-D**		
	Man	NoSQL	OP	Man	NoSQL	OP	Man	NoSQL	OP	Man	NoSQL	OP
Alg.	0.33	0.44	0.37	0.14	0.10	0.05	0.34	0.32	0.3	0.19	0.14	0.28
Assi.	0.28	0.15	0.0	0.65	0.50	1.0	0.06	0.10	0.0	0.01	0.04	0.0
Check.	0.30	0.48	0.50	0.26	0.34	0.29	0.24	0.14	0.21	0.19	0.04	0.0
Funct.	0.38	0.34	0.51	0.24	0.42	0.14	0.24	0.11	0.0	0.14	0.13	0.07
Tim.	0.19	0.06	0.0	0.05	0.08	0.0	0.62	0.76	0.0	0.14	0.10	0.0

The 94 orthogonal-defect-labeled reports were manually classified by R1 into UCA classes and compared to the Lbl2Vec classifications. Of the 94 reports, only 86 could be clearly manually classified into a UCA class. The other eight lacked descriptive information to sufficiently derive a UCA class. Lbl2Vec matching and mismatching classifications can be seen in **Error! Reference source not found.**. In columns 2 to 6, the matching to total manual UCA classification can be seen. For instance, for column 2, only 26 of 32 UCA-A classifications made by Lbl2Vec matched the researcher's classifications. Ultimately, the derived Kappa value suggests that Lbl2Vec is only *moderately* capable at correctly determining some UCA classes, namely UCA-A and UCA-B. The Kappa value for UCA-C and UCA-D suggest *fair* and *slight* respectively and is most likely attributed to the small sample size.

Table VIII. Kappa values for Lbl2Vec on OpenPilot with number of classifications per UCA.

Value	**UCA-A**	**UCA-B**	**UCA-C**	**UCA-D**	**Match/Total**
Lbl2Vec/Manual	26/32	32/42	3/5	2/7	63/86
Kappa Value	0.75	0.68	0.47	0.05	0.64

4. CHALLENGES AND OPPORTUNITIES

The research presented suggests that an alternative approach to software failure quantification may exist through root cause analysis correlations to specific UCA events. This would greatly improve upon the current approach of software basic event quantification which uses the IEC 61508 [14] safety integrity level (SIL) 4 bounding estimates [15]. While SIL 4 bounding estimates are currently agreed to be the necessary reliability for safety critical digital systems, it also produces overly conservative estimates of failure probability in FTA of software-based digital I&C systems in nuclear power plants [15]. By analyzing defects detected and their associated orthogonal defect class through the software development life cycle, an estimate of the occurrence probability of UCA events can be obtained. The application of Lbl2Vec for software defect mining also significantly reduces the amount of human effort required to develop the defect-UCA correlation table. By assessing an even larger dataset of software defects (such as in https://www.kaggle.com/datasets/davidshinn/github-issues), a global correlation table may be derived to assess the reliability of any software system regardless of function or usage.

However, there are several challenges that need to be addressed first. In terms of model development, parameter optimization unsurprisingly significantly impacts the final UCA topic classification. Parameters that need further examination include, but are not limited to, the number and choice of keywords. Although our selection of keywords was based on occurrence frequency within labeled reports, it is not known whether technical jargon would result in better topic classification. Another challenge in this approach is developing consistent labeling of UCAs for keyword extraction. While R1 and R2 had good Kappa agreement, this does not imply that all defect-UCA classifications were correct or consistent. For instance, for certain defect reports, it could not be agreed upon whether UCA-C or UCA-D was a better fit for those reports. Furthermore, while Lbl2Vec is an unsupervised methodology for topic classification, the choice of defect reports to include in the training of the model impacts which word embeddings are associated with keywords and may vary the final topic centroid, ultimately impacting UCA topic classification. Therefore, while the use of natural language processing (i.e., Lbl2Vec) is promising, a consistent methodology for identifying the correct and relevant training data is still needed. As the statement goes, “garbage in, garbage out” is more relevant than ever for proper model development.

5. CONCLUSIONS

FTA and event quantification play a crucial role in the licensing and modernization of existing and proposed nuclear digital I&C systems. Therefore, this work proposes a method to automate the root cause analysis of software defects to assist in the quantification of UCAs that may be integrated in fault trees. The results suggest that a consistent correlation between software defects and UCAs exists such that an alternative quantification method may be adopted in lieu of existing bounding estimate approaches. Future work will focus on comparing the results from other natural language processing models applied for root cause analysis to the results derived from Lbl2Vec.

ACKNOWLEDGMENTS

This work is supported under U.S. Government under DOE Contract No. DE-AC07-05ID14517. Accordingly, the U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes. Neither the U.S. Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. References herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, do not necessarily constitute or imply its

endorsement, recommendation, or favoring by the U.S. Government or any agency thereof. The views and opinions of authors expressed do not reflect those of the U.S. Government or any agency thereof.

REFERENCES

[1] Nuclear Energy Institute, "License Amendment Request Guidelines," Nuclear Energy Institute, 2006.

[2] J. Thomas, F. L. d. Lemos and N. Leveson, "Evaluating the Safety of Digital Instrumentation and Control Systems in Nuclear Power Plants," U.S. Nuclear Regulatory Commission, 2012.

[3] N. G. Leveson and J. P. Thomas, STPA Handbook, Massachusetts Institute of Technology, 2018.

[4] Electric Power Research Institute, "HAZCADS: Hazards and Consequences Analysis for Digital Systems - Revision 1," Electric Power Research Institute, 2021.

[5] T. Shorthill, H. Bao, H. Zhang and H. Ban, "A Redundancy-Guided Approach for the Hazard Analysis of Digital Instrumentation and Control Systems in Advanced Nuclear Power Plants," *Nuclear Technology,* vol. 208, no. 5, pp. 892-911, 2021.

[6] H. Bao, H. Zhang, T. Shorthill, E. Chen and S. Lawrence, "Quantitative Evaluation of Common Cause Failures in High Safety-significant Safety-related Digital Instrumentation and Control Systems in Nuclear Power Plants," *Reliability Engineering & System Safety,* vol. 230, 2023.

[7] E. Chen, N. Dinh, T. Shorthill, C. Elks, A. V. Jayakumar and H. Bao, "Application of Orthogonal Defect Classification for Software Reliability Analysis," in *Probabilistic Safety Assessment and Management*, Honolulu, 2022.

[8] F. Lopes, J. Agnelo, C. A. Teixeira, N. Laranjeiro and J. Bernardino, "Automating orthogonal defect classification using machine learning algorithms," *Future Generation Computer Systems,* vol. 102, pp. 932-947, 2020.

[9] International Business Machines, "Orthogonal Defect Classification v 5.2 for Software Design and Code," 12 September 2013. [Online]. Available: https://s3.us.cloud-object-storage.appdomain.cloud/res-files/70-ODC-5-2.pdf.

[10] E. Chen, N. Dinh, T. Shorthill, C. Elks, A. V. Jayakumar and H. Bao, "Application of Orthogonal Defect Classification for Software Reliability Analysis," in *Probabilistic Safety Assessment and Management Conference*, Honolulu, 2022.

[11] M. F. Porter, "An algorithm for suffix stripping," *Program,* vol. 14, no. 3, pp. 130-137, 1980.

[12] T. Schopf, D. Braun and F. Matthes, "Web Information Systems and Technologies," in *Semantic Label Representations with Lbl2Vec: A Similarity-Based Approach for Unsupervised Text Classification*, Cham, Springer International Publishing, 2023, pp. 59-73.

[13] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement,* vol. 20, pp. 37-46, 1960.

[14] International Electrotechnical Commission, "IEC 61508," International Electrotechnical Commission, Geneva, 2010.

[15] R. W. Rolland and R. Schneider, "Lessons Learned in PRA Modeling of Digital Instrumentation and Control Systems," in *Probabilistic Safety Assessment and Management*, Honolulu, 2022.

[16] U.S. Nuclear Regulatory Commission, "50.92 Issuance of Amendment," U.S. Nuclear Regulatory Commission, 25 March 2021. [Online]. Available: https://www.nrc.gov/reading-rm/doc-collections/cfr/part050/part050-0092.html. [Accessed 9 January 2024].