Are Elevator Software Robust against Uncertainties? Results and Experiences from an Industrial Case Study

Liping Han
liping@simula.no
Nanjing University of Aeronautics and Astronautics
Nanjing, China
Simula Research Laboratory
Oslo, Norway

Tao Yue
tao@simula.no
Simula Research Laboratory
Oslo, Norway

Shaukat Ali
shaukat@simula.no
Simula Research Laboratory
Oslo, Norway

Aitor Arrieta
aarrieta@mondragon.edu
Mondragon University
Mondragon, Spain

Maite Arratibel
marratibel@orona-group.com
Orona
Hernani, Spain

ABSTRACT

Industrial elevator systems are complex Cyber-Physical Systems operating in uncertain environments and experiencing uncertain passenger behaviors, hardware delays, and software errors. Identifying, understanding, and classifying such uncertainties are essential to enable system designers to reason about uncertainties and subsequently develop solutions for empowering elevator systems to deal with uncertainties systematically. To this end, we present a method, called RuCynefin, based on the Cynefin framework to classify uncertainties in industrial elevator systems from our industrial partner (Orona, Spain), results of which can then be used for assessing their robustness. RuCynefin is equipped with a novel classification algorithm to identify the Cynefin contexts for a variety of uncertainties in industrial elevator systems, and a novel metric for measuring the robustness using the uncertainty classification. We evaluated RuCynefin with an industrial case study of 90 dispatchers from Orona to assess their robustness against uncertainties. Results show that RuCynefin could effectively identify several situations for which certain dispatchers were not robust. Specifically, 93% of such versions showed some degree of low robustness against uncertainties. We also provide insights on the potential practical usages of RuCynefin, which are useful for practitioners in this field.

CCS CONCEPTS

• **Software and its engineering → Software performance**; **Empirical software validation**; **Software testing and debugging**; • **Computer systems organization → Embedded systems**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3558955

KEYWORDS

Industrial elevator systems, Robustness assessment, Cynefin framework, Uncertainty

ACM Reference Format:
Liping Han, Tao Yue, Shaukat Ali, Aitor Arrieta, and Maite Arratibel. 2022. Are Elevator Software Robust against Uncertainties? Results and Experiences from an Industrial Case Study. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3558955

1 INTRODUCTION

Orona[1], Spain, is a renowned industrial elevator developer manufacturing a variety of elevators based on customers' requirements. Industrial elevators, being Cyber-Physical Systems (CPS), operate in dynamic and uncertain environments (e.g., uncertain human behaviors), and continuously evolve throughout their life-cycles. It is important for elevator systems to handle various uncertainties robustly and maintain good Quality of Services (*QoS*) after bug fixing, software update, etc. Thus, it is essential to assess their robustness against uncertainties during their development and operation.

To this end, we propose RuCynefin and use it to assess the robustness of 90 dispatchers of an industrial elevator system in the Software in the Loop (SiL) configuration provided by our industrial partner – Orona, under a variety of uncertainties. The dispatcher is the key software module of an elevator system responsible for the optimal scheduling of the elevator, while ensuring an acceptable level of *QoS* measured, e.g., with average waiting time. Considering a large number of uncertainties and impracticality of including all of them at once, in this paper, we only focus on passengers' uncertainties. These uncertainties are related to passengers' attributes and characteristics of using elevators, such as required time to get in/out of an elevator.

Our RuCynefin method is based on the Cynefin framework [32], which has been widely used for decision-making under uncertainty

[1] https://www.orona-group.com/

[20, 21, 47], to categorize uncertainty during software development [44], and aid clinical reasoning [36]. RuCynefin consists of an uncertainty generator, a classification algorithm that identifies and classifies various passenger uncertainties, and a novel robustness metric based on the classification. We conducted an empirical evaluation with the 90 industrial dispatcher versions from Orona. Results show that RuCynefin is highly effective in identifying uncertain factors and their interactions that need to be optimized. More specifically, 93% of dispatchers exhibited a certain degree of low robustness against uncertainties. Moreover, we present our experiences and lessons learned in the form of practical usages, which, we believe, are valuable for researchers and industrial practitioners who are interested in robustness assessment of elevator systems or other similar CPS, against uncertainties. Hence, we summarize our main contributions as follows: 1) proposed RuCynefin, a novel method for assessing the overall robustness of dispatchers of industrial elevator systems against uncertainties, based on the Cynefin framework; 2) extensively evaluated RuCynefin with an industrial case study of 90 dispatchers; and 3) shared our experiences and lessons learned.

Structure. Section 2 introduces the industrial context. Section 3 presents RuCynefin. The empirical study is presented in Section 4, followed by the practical usages of RuCynefin (Section 5), related work (Section 6). We conclude the paper in Section 7.

2 INDUSTRIAL CONTEXT

Orona designs, manufactures, installs, and maintains various types of vertical transportation systems (e.g., elevators and escalators) that are deployed in many types of buildings (e.g., hospitals, airports, supermarkets) [24]. In this paper, we focus on **elevator systems**, which transport passengers between floors of a building, while maintaining service robustness and passengers' comfortableness.

Figure 1 presents an example of an elevator system with three elevators serving *N* floors (*Floor 1 - Floor N*) of a building. Each floor is equipped with a landing operating panel to collect passenger calls. Corresponding to each elevator, there is a designated controller that controls movements of the elevator between floors. All the control panels and controllers receive instructions from the *Traffic Master* via the Controller Area Network (CAN) bus. The *Traffic Master* is a software component that has an essential module called traffic **dispatcher** responsible for handling all passenger calls.

A dispatcher responds to passenger calls and schedules elevators at an acceptable level of ***QoS*** measured with various metrics, which are calculated with *Time List* formed with *Waiting Time* (*WT*), *Transit Time* (*TT*) and *Time to Destination* (*TD*). As shown in Figure 1, *WT* is the time between a passenger registering a call on a certain floor until an elevator arrives and its door begins to open. If the elevator door is already open when the passenger arrives, *WT* for this passenger is 0. *TT* is the time between the elevator door begins to open until it reaches the passenger's destination floor and begins to open the elevator door again. *TD* is the sum of the passenger's waiting time and transit time.

With *WT*, *TT*, and *TD*, we define the following *QoS* metrics [9]: 1) *AWT* is the average waiting time of all passengers whose calls have been answered within a certain period of time; 2) *LWT* is the longest waiting time experienced by a passenger within a certain period of time; 3) *ATT* is the average transit time of all passengers who completed their journeys within a certain period of time; 4) *LTT* is the longest transit time experienced by a passenger who completed her/his journey within a certain period of time; 5) *ATD* is the average time to destination of all passengers within a certain period of time; 6) *LTD* is the longest time to destination experienced by a passenger within a certain period of time.

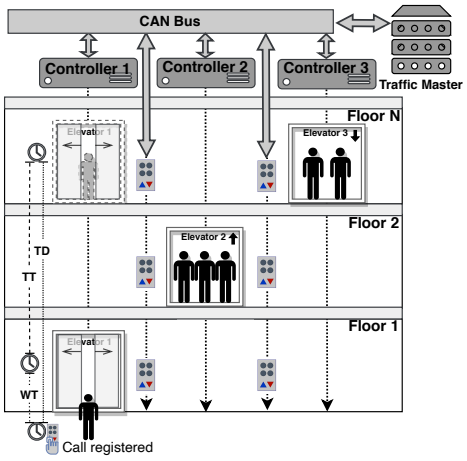

Figure 1: A Simplified Elevator System

We study the **robustness**, which refers to the degree to which a dispatcher's *functional performance* is not degraded in the presence of uncertainties. Such performance is measured with the *QoS* metrics. *Functional performance*, in the context of CPS, refers to the properties derived indirectly from the output of the CPS, rather than its efficient usage of the computational resources [23]. Such properties evaluate the quality of a CPS's output due to the lack of a more direct way to evaluate its behavior [23], which is our case.

We focus on the SiL enabled with the commercial **Elevate**[2] simulator. Elevate can be configured with various settings such as building types, passenger information (e.g., how many passengers using the elevator and what are the attributes of each passenger), and elevator information (e.g., car area, capacity). As shown in Figure 2, Elevate requires the following inputs: 1) simulation settings (e.g., time slice between simulation calculations) and the selection of a dispatcher, i.e., Orona's dispatcher in our context; 2) building configurations such as the height and number of floors; 3) elevator system design parameters such as the number of elevators, capacity, speed and car area; and 4) passenger traffic data, including passengers' attributes such as *Mass*, and information about passenger calls (e.g., passenger's arrival and destination floors). After setting these parameters, a simulation can be run. When it is completed, Elevate generates analysis reports including values of the *QoS* metrics, and detailed reports about *WT* and *TT* values of each passenger, etc.

A dispatcher is tested under various **traffic profiles** (an input file to Elevate) to ensure that it can handle different passenger traffic with an acceptable level of *QoS*. A traffic profile contains information about passengers using the elevators. For each passenger, the

[2] https://elevate.helpdocsonline.com/home

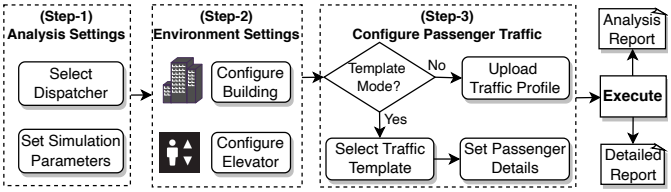

Figure 2: Software-in-the-loop (SiL) with Elevate

following information is specified: 1) *Arrival Time* - the time (in seconds) at which a passenger arrives on the floor and calls the elevator; 2) *Arrival Floor* and 3) *Destination Floor* - the floor number of the passenger's arrival and destination; 4) *Mass* - the passenger's weight in kilograms; 5) *Capacity Factor* - a percentage value based on which a passenger decides whether to enter in an elevator; 6) *Loading Time* and 7) *Unloading Time* - the time required for a passenger to enter or exit the elevator, in seconds. A traffic profile can either be generated with a selected **traffic template** or uploaded by users (Figure 2). Note that, with a traffic template, Elevate generates a list of passengers with the same values for (*Mass*, *Capacity Factor*, *Loading Time* and *Unloading Time*), and a fixed set of values for (*Arrival Time*, *Arrival Floor* and *Destination Floor*).

We employed **90 dispatcher versions from Orona**, among which 89 versions (D01-D89) were manually generated by a domain expert using different mutation operators (e.g., relational operator change) and different changes on the parameters based on the original dispatcher (D00). Note that a total of 99 versions were initially generated, ten out of them were faulty, and the remaining 89 versions, which were reviewed by another domain expert, do not provoke functional failures (e.g., leaving a call unattended), but rather result in a sub-optimal elevator assignment affecting *QoS*. This means that various dispatcher versions lead to various degrees of functional performance degradation such as increased average waiting time.

3 THE RUCYNEFIN METHODOLOGY

The overall process and key phases of RuCynefin for assessing dispatcher's robustness are presented in Figure 3. As shown in the figure, RuCynefin first generates uncertain traffic profiles based on uncertain factors being investigated, using recommended configurations in the CIBSE Guide D [9]. Each uncertain traffic profile is then executed with the same configurations, to simulate various uncertain situations. After executing all the simulations, the uncertain situations are classified into the Cynefin contexts based on the simulation outputs. With the classification results, the robustness scores are further calculated to assess the dispatcher's robustness.

In the rest of this section, we first present a short summary of the Cynefin framework in Section 3.1, followed by the generation of uncertain traffic profiles in Section 3.2. We then present the classification algorithm in Section 3.3. Finally, the calculation of the robustness scores is given in Section 3.4.

3.1 The Cynefin Framework

The Cynefin framework [32], originated in knowledge management, helps decision makers to make sense of, e.g., (unspecified) problems, uncertain situations, cybersecurity risk [7], such that they can eventually come to consensuses when making decisions under uncertainty [32]. Cynefin has five transferable contexts: 1) *Simple* – cause and effect are understood and predictable; 2) *Complicated* – cause and effect can be determined with sufficient data; 3) *Complex* – cause and effect may be determined after an event; 4) *Chaotic* – cause and effect are not discernible; and 5) *Disorder* (also called *confused*) [32]. Each of the first four defined contexts corresponds to an action strategy. To apply Cynefin, one must categorize, e.g., problem space, uncertain situations into one of the four contexts based on the *relationship* between *cause* and *effect* [44]. Note that *Disorder* is the context where you don't know which of the other four contexts the current situation should be. Thus, more information should be gathered when a *Disorder* context is identified so as to move to the other defined contexts. In this paper, we adapt the Cynefin contexts (except for *Disorder*) for assessing the robustness of industrial elevator systems, by categorizing the impact (*relationship*) of uncertainties (*cause*) on the dispatcher's robustness measured with the *QoS* metrics.

3.2 Generating Uncertain Traffic Profiles

To simulate **uncertain situations**, we implemented *Uncertain Traffic Profiles Generator*, which generates **uncertain traffic profiles** for given **uncertain factors** related to passengers with Algorithm 1. Assuming a set of uncertain factors $U = \{u_i | i = 1, 2, ..., un\}$, where u_i is the ith uncertain factor (e.g., uncertain *Mass*), un is the number of uncertain factors, the uncertain situations caused by uncertain factor set U can be represented as $US = \{us_{i,j} | i = 1, 2, ..., un; j = 1, 2, ..., C^i_{un}\}$, where $us_{i,j}$ is the jth uncertain situation caused by the i number of uncertain factors. The number of all possible uncertain situations generated under uncertain factor set $U = \{u_i | i = 1, 2, ..., un\}$ is $C^1_{un} + C^2_{un} + ... + C^{un}_{un} = 2^{un} - 1$. Hence, US in total will have $2^{un} - 1$ types of uncertain traffic profiles covering all the uncertain situations caused by the single uncertain factor and t-way interactions between several uncertain factors in U. For instance, for studying three uncertain factors, e.g., *Mass*, *Loading* and *Unloading Times*, the number of uncertain situations caused by single uncertain factor, 2-way interactions and 3-way interactions are: 3 (C^1_3, $\{us_{1,1} = usM, us_{1,2} = usL, us_{1,3} = usU\}$), 3 ($C^2_3$, $\{us_{2,1} = usM - L, us_{2,2} = usM - U, us_{2,3} = usL - U\}$), and 1 ($C^3_3$, $\{us_{3,1} = usM - L - U\}$), respectively. Thus, the uncertain situation set US totally has 7 types of ($C^1_3 + C^2_3 + C^3_3$) uncertain situations.

Uncertain traffic profiles of an uncertain situation are produced by generating a value for each of its uncertain factors within predefined intervals based on a baseline profile P_b (i.e., the original traffic profile without uncertainties and configured with the recommended values), while ensuring that the average value of all the passengers meets the recommended values, e.g., from the CIBSE Guide D [9]. With these recommended values, we can define interval set $I = \{I_i | i = 1, 2, ..., un\}$, where, $I_i = [a_i, b_i]$ is the interval for uncertain factor u_i. The minimum and maximum values of the interval are chosen by $\pm x$ based on the recommended value. For example, the recommended value for *Mass* in Europe is 75KG, whereas by choosing $x = 5$, we obtain the minimum $a_i = 70$ KG and the maximum $b_i = 80$ KG for the interval.

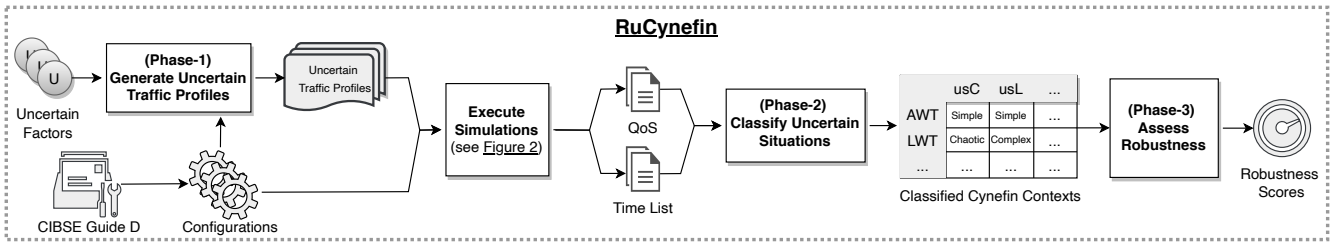

Figure 3: Overview of RuCynefin

Algorithm 1: Generating Uncertain Traffic Profiles

Input: Uncertain factor set U, interval set I, number of repetitions per generation NR, baseline profile P_b
Output: Uncertain traffic profile set P

1 **for** $us_{i,j}$ *in* US **do**
2 **for** $k \leftarrow 1$ **to** NR **do**
3 $P_{i,j,k} \leftarrow P_b$
4 $SatisfyConstraint \leftarrow$ FALSE
5 **while** $!SatisfyConstraint$ **do**
6 **for** $p_{i,j,k,m}$ *in* $P_{i,j,k}$ **do**
7 **for** u_i *in* $us_{i,j}$ **do**
8 generate value of u_i of $p_{i,j,k,m}$ within I_i
9 **if** *average value of* u_i *in* $P_{i,j,k}$ *equals to the value of* u_i *in* P_b **then**
10 save $P_{i,j,k}$
11 $SatisfyConstraint \leftarrow$ TRUE

For each uncertain situation, we repeat the generation NR times. The kth uncertain traffic profile generated under uncertain situation $us_{i,j}$ can be represented as $P_{i,j,k} = \{p_{i,j,k,m} | i = 1, 2, ..., un; j = 1, 2, ..., C^i_{un}; k = 1, 2, ..., NR; m = 1, 2, ..., NP\}$, where $p_{i,j,k,m}$ is the mth passenger of uncertain traffic profile $P_{i,j,k}$. As shown in Algorithm 1, we first initialize the uncertain traffic profile $P_{i,j,k}$ with the baseline profile P_b (Line 3), then we generate the uncertain factor's value of each passenger in uncertain traffic profile $P_{i,j,k}$ within the predefined interval (Line 8). To avoid the generated uncertain traffic profiles deviating too much from the baseline, we only keep those profiles whose average values of each uncertain factor are equal to their corresponding values in the baseline profile (Lines 9-10). For example, suppose we want to generate uncertain traffic profiles of the baseline profile based on uncertain factor *Mass*, we generate each passengers' *Mass* within the predefined interval and only keep profiles that have the average value of *Mass* of all the passengers equal to that of the baseline profile.

3.3 Automatically Classifying Uncertain Situations into the Cynefin Contexts

An execution, with the dispatcher and Elevate[3], under an uncertain situation $us_{i,j}$ simulated by a specific uncertain traffic profile $P_{i,j,k}$, produces *Time Lists* and a set of values for the *QoS* metrics (e.g., *AWT*). After running simulations with all generated uncertain traffic profiles P of all uncertain situations US, we classify, with Algorithm 2, each uncertain situation $us_{i,j}$ into a Cynefin context.

As shown in Algorithm 2, output $C = \{c_{i,j,l} | i = 1, 2, ..., un; j = 1, 2, ..., C^i_{un}; l = 1, 2, ..., nq\}$, $c_{i,j,l}$ is the classified Cynefin context of uncertain situation $us_{i,j}$ for the lth *QoS* metric, where nq is the total number of *QoS* metrics.

Q (Line 1) is the set of values for *QoS* metrics represented as $Q = \{Q_{i,j,k} | i = 1, 2, ..., un; j = 1, 2, ..., C^i_{un}; k = 1, 2, ..., NR\}$, $Q_{i,j,k} = \{q_{i,j,k,l} | l = 1, 2, ..., nq\}$, $Q_{i,j,k}$ is the kth set of values of *QoS* metrics under uncertain situation $us_{i,j}$, $q_{i,j,k,l}$ is a value of the lth *QoS* metric in $Q_{i,j,k}$. T (Line 1) is the set of *Time Lists*, which can be further represented as $T = \{T_{i,j,k,l} | i = 1, 2, ..., nu; j = 1, 2, ..., C^i_{un}; k = 1, 2, ..., NR; l = 1, 2, ..., nq\}$, $T_{i,j,k,l}$ represents the *Time List* corresponding to the $q_{i,j,k,l}$.

Lines 2-4 study the impact of uncertain situations on the *QoS* based on collected values of *QoS* metrics, in a coarse-grained manner. Simulating one uncertain situation NR times produces NR values for each *QoS* metric. For each *QoS* metric, these values are compared with one value of the baseline traffic profile using the one sample Wilcoxon signed rank test at the significance level of 0.05. If a resulting *p-value* is less than 0.05, then it means that the uncertain situation has significant impact on a specific *QoS*. The calculation is as follows:

$$p_{qos,i,j,l} = OSW((q_{i,j,1,l}, ..., q_{i,j,NR,l}), q_{bas,l}) \quad (1)$$

Lines 5-7 show a fine-grained way of classification; the NR generated *Time Lists* are compared with the one for the baseline traffic profile, using the paired Wilcoxon signed rank test at a significance level of 0.05, which results NR *p-values*:

$$p_{tl,i,j,k,l} = PW(T_{i,j,k,l}, T_{bas,l}) \quad (2)$$

In Equations 1 and 2, $OSW(x, y)$ and $PW(x, y)$ are one sample and paired Wilcoxon signed rank tests, respectively, which are one-tailed tests for checking if an uncertain situation leads to significantly longer time; $p_{qos,i,j,l}$ is the *p-value* corresponding to the lth *QoS* metric under uncertain situation $us_{i,j}$; $q_{bas,l}$ is the value of the lth *QoS* metric generated by the baseline profile; $(q_{i,j,1,l}, ..., q_{i,j,NR,l})$ is an array of all the values of the lth *QoS* metric generated by NR uncertain traffic profiles of $us_{i,j}$; $p_{tl,i,j,k,l}$ is the *p-value* corresponding to the *Time List* related to the lth *QoS* metric generated by uncertain traffic profile $P_{i,j,k}$; and $T_{bas,l}$ is the *Time List* for to the lth *QoS* metric generated by the baseline profile.

[3]Note that Elevate only generates *WT* and *TT*, from which *TD* is derived.

Algorithm 2: Cynefin Classification Algorithm

```
Input: Simulation results SR
Output: Classification results C
1  Extract Q and T from SR
2  for Q_{i,j,k} in Q do
3      for q_{i,j,k,l} in Q_{i,j,k} do
4          get p_{qos,i,j,l} using Eq. 1
5  for T_{i,j,k,l} in T do
6      for k ← 1 to NR do
7          get p_{tl,i,j,k,l} using Eq. 2
8      get Conf_{i,j,l} using Eq. 3
9  for us_{i,j} in US do
10     for l ← 1 to nq do
11         if p_{qos,i,j,l} ≥ 0.05 then
12             if Conf_{i,j,l} > 0.5 then
13                 c_{i,j,l} ← Simple
14             else if Conf_{i,j,l} < 0.5 then
15                 c_{i,j,l} ← Complicated
16             else
17                 c_{i,j,l} ← Border(Simple, Complicated)
18         else
19             if Conf_{i,j,l} > 0.5 then
20                 c_{i,j,l} ← Complex
21             else if Conf_{i,j,l} < 0.5 then
22                 c_{i,j,l} ← Chaotic
23             else
24                 c_{i,j,l} ← Border(Complex, Chaotic)
```

Line 8 obtains a value denoting the confidence of an observation about an uncertain situation $us_{i,j}$ not leading to a significant impact on generated *Time Lists*. Simply put, a confidence is the number of *p-values* that are greater than or equal to 0.05 out of the total comparisons. For example, if $NR = 10$, and among all the 10 *p-values*, eight of them are greater than or equal to 0.05, we then say with the 80% confidence that the 10 *Time Lists* are not significantly degraded than the baseline one, as calculated below:

$$Conf_{i,j,l} = (NR - IC_{i,j,l})/NR \tag{3}$$

$$IC_{i,j,l} = \sum_{k=1}^{NR} Sym(p_{tl,i,j,k,l}) \tag{4}$$

where $IC_{i,j,l}$ is the number of uncertain traffic profiles generated under uncertain situation $us_{i,j}$, which have led to a significant influence on *Time List(s)* corresponding to the lth *QoS* metric; $Sym(x)$ is a symbolic function defined as:

$$Sym(x) = \begin{cases} 0 & x \geq 0.05 \\ 1 & x < 0.05 \end{cases} \tag{5}$$

The other lines of Algorithm 2 perform classifications based on the degree of impacts of the uncertain situation on the dispatcher. For example, if no significant impacts are observed from all comparisons of the *QoS* metrics and *Time List* (e.g., *AWT* and *WT Time List*), it is a *Simple* context, as $p_{qos,i,j,l} \geq 0.05$ and $Conf_{i,j,l} = 1$ (which is greater than 0.5, satisfying the condition in Line 12 of Algorithm 2). Conversely, $p_{qos,i,j,l} < 0.05$ and $Conf_{i,j,l} < 0.5$ imply that the uncertain situation has the highest degree of impacts on the dispatcher, i.e., both *QoS* and *Time List* are significantly degraded by uncertainties, and thus it is a *Chaotic* context.

3.4 Calculating Robustness Scores

To evaluate the overall robustness of a dispatcher in the presence of various uncertainties, the overall robustness score is calculated, based on the percentage of each Cynefin context among all the classified Cynefin contexts, as follows:

$$RScore = \sum_{cc=Simple}^{Chaotic} (Per_{cc} * RC_{cc}) \tag{6}$$

where Per_{cc} is the percentage of a particular Cynefin context cc, among all the classification results. More specifically, in our case study, if we evaluate the overall robustness of a dispatcher, the total number of classifications is 90, i.e., 15 (uncertain situations) × 6 (*QoS* metrics). When calculating *QoS* specific robustness (e.g., the *AWT* robustness score), the total number of classifications is 15. Similarly, when calculating the robustness score under a specific uncertain situation (e.g., *usM*), the total number of classifications is 6. RC_{cc} is the robustness coefficient of cc. Context *Simple* means that introduced uncertainties don't significantly worsen both *QoS* and *Time List*, so the robustness coefficient is set as 1. The robustness coefficients of the other Cynefin contexts are defined based on various extents of the impact of uncertainties on the elevator's robustness measured with *QoS* and *Time List*. Hence, the robustness coefficients of all the Cynefin contexts are, by default, defined as: $\{RC_{Simple} = 1; RC_{Border(Simple,Complicated)} = 0.875; RC_{Complicated} = 0.75; RC_{Complex} = 0.5; RC_{Border(Complex, Chaotic)} = 0.375; RC_{Chaotic} = 0.25\}$. Note that these default coefficients can be customized for specific application contexts.

With these coefficients, one can see that when evaluating the overall robustness of an elevator dispatcher, a high *RScore* implies a higher robustness, thereby preferred. An ideal dispatcher should be robust at all time under all possible uncertainties. We define such a dispatcher as *Robustestly-behaved dispatcher*, which should be classified into *Simple* in all cases, and has its *RScore* being 100.

4 EMPIRICAL EVALUATION

In this section, we first present the experiment design and execution (Section 4.1), followed by the results and analysis (Section 4.2), and the threats to validity in Section 4.3.

4.1 Experiment Design and Execution

Research Questions (RQs). RQ1: How is the overall robustness of the dispatchers, when measured with robustness score (Eq. 6)? **RQ2:** How do the different dispatchers perform in terms of each *QoS* metric (which is considered as an atomic robustness metric) across the various Cynefin contexts? **RQ3:** How do the uncertain factors and their interactions affect the robustness of the dispatchers? **RQ4:** How do the two-way interaction of the uncertain situations and the *QoS* metrics affect the robustness of dispatchers?

Settings. We employed an office building configuration from Section 4.8 of the CIBSE Guide D [9] in our experiment, which has 6 elevators, 14 floors, and 1120 person population (80 persons per floor). Each elevator is configured as: Capacity (KG): 1600; Car area (m^2): 3.56; Speed (m/s): 2.5; Acceleration (m/s^2): 0.8 and Jerk (m/s^3): 1.0. Moreover, we adopted the modern office lunch peak[4] (*Lunch Peak*) traffic template for generating the baseline traffic profile, which models the lunch peak traffic between 12:15pm and 13:15pm, comprising 1409 passengers. Furthermore, for the passenger attributes, we set the values of *Mass* to 75KG (i.e., the average value for *Mass* in Europe [9]), *Capacity Factor* to 70% based on examples from [9], *Loading* and *Unloading Times* to 1.2 seconds (the average time to enter and exit the elevator according to [9]).

After setting the passenger details, we ran the Orona's dispatcher with Elevate, and obtained the baseline traffic profile. Note that each passenger's *Arrival Time*, *Arrival Floor* and *Destination Floor* are generated by Elevate based on a selected traffic template. We kept the values of these parameters the same throughout the experiment, since changing them will change the traffic template provided by Elevate, which consequently makes the traffic template, e.g., *Lunch Peak*, unrealistic. Instead, each passenger in the generated traffic profile has the same attributes, i.e., *Mass*, *Capacity Factor*, *Loading* and *Unloading Times*. However, these attributes are unpredictable in reality. Thus, we consider them as uncertain factors in this experiment.

Execution. With the obtained baseline profile, we generate uncertain traffic profiles (Algorithm 1) based on investigated uncertain factors, i.e., *Mass*, *Capacity Factor*, *Loading Time*, and *Unloading Time*. We set $x = 20\%$ for each uncertain factor, to define intervals, except for *Capacity Factor*. This is because its maximum value is usually 80% [9] and its baseline value is set as 70%, so we use ±10, instead of ±20%, to avoid generating values over 80%. The four uncertain factors lead to 15 uncertain situations (the size of US being 15) corresponding to 4 one way and 11 t-way uncertain interactions. With Elevate, we generated $NR = 10$ traffic profiles for each uncertain situation, and consequently we obtained 150 uncertain traffic profiles, in total.

4.2 Results and Analyses

4.2.1 Results of RQ1 - Overall classification results of the dispatchers. Figure 4 presents the overall robustness of the 90 dispatcher versions. Each dispatcher has 15×6 classifications (corresponding to the 15 uncertain situations combined with the 6 *QoS* metrics), each of which is classified into one of the six Cynefin contexts with Algorithm 2. To compare the overall robustness across the dispatchers, the classification results are combined to compute robustness scores (Section 3.4). Robustness scores of all the dispatchers range from 25 to 100, as shown in Figure 4. From the figure, one can observe that, the 90 dispatchers exhibit diverse robustness. For instance, D00, D02, D19, D25, D42 and D78 exhibit the same robustness as the *Robustestly-behaved dispatcher*, i.e., all the 15×6 cases were 100% classified into the *Simple* context, implying an excellent robustness against uncertainties. On the other hand, the rest shows worse or even poor robustness (e.g., D65 and D69) in the presence of various uncertainties. The overall robustness scores range from 46 (D69) to 100, implying that RuCynefin is effective in distinguishing the robustness of dispatchers. Such cases involve dispatchers with modifications to intentionally impact their *QoS*. Moreover, we observe that, **93%** $((90 - 6)/90)$ of all the 90 dispatchers do not achieve the same robustness as the *Robustestly-behaved dispatcher*, and some (e.g., D41, D65, D69) were classified into the *Complex* and even *Chaotic* contexts. These dispatchers are suggested to our industrial partner such that they can improve their performance (e.g., by changing the parameters of the dispatcher).

> Conclusion for RQ1: The 90 dispatchers exhibit diverse robustness in terms of dealing with uncertainties. A few dispatchers (e.g., D00) performed as good as *Robustestly-behaved dispatcher*, while some showed poor robustness (e.g., D69). **Thus, we recommend that engineers shall test the overall robustness of a new version of dispatcher with RuCynefin before deploying it for real operation.**

4.2.2 Results of RQ2 - Comparing the dispatchers across QoS metrics. To compare the *QoS* specific robustness, the robustness scores for each dispatcher, in respect to each *QoS* metric, are presented as the violin plots in Figure 5. From the figure, one can observe that, the average robustness scores in terms of *AWT* and *ATD* are obviously higher than that for the others. This is because, in terms of these two *QoS* metrics, the dispatchers were mostly classified into the *Simple* context, which contributes to the high robustness scores. From Figure 5, we can also observe that the distributions of the robustness scores, in terms of *AWT* and *ATD*, have less variability. It means that the 90 dispatchers have relatively similar robustness in terms of *AWT* and *ATD*, i.e., the robustness with respect to *AWT* and *ATD* are more stable. Moreover, we compared the *QoS* specific robustness score of each dispatcher with that of the hypothetical *Robustestly-behaved dispatcher*, and summarized the number of dispatchers that performed less robust than the hypothetical *Robustestly-behaved dispatcher* in terms of each *QoS* metric, as shown in Figure 5. We can see that a majority of the 90 dispatchers achieved as good robustness as *Robustestly-behaved dispatcher* in terms of *AWT*, followed by *ATD*, when facing uncertainties. One plausible explanation is that the dispatchers were optimized towards reducing passengers' waiting times, which have a direct impact on *AWT* and *ATD*. This led to more cases being classified into the *Simple* context and, consequently, achieve higher robustness when dealing with uncertainties. Such explanation has been confirmed by domain experts from Orona.

We also present the detailed descriptive statistics in terms of each *QoS* metric across the six contexts in Figure 5. From the figure, we first can observe that, in terms of each *QoS* metric, most of the cases were classified into the *Simple* context, with mean values ranging from 9.44 to 12.29, followed by the *Complex* context (with mean values ranging from 1.31 to 4.13). This observation is consistent across the six *QoS* metrics, as *Simple* and *Complex* are ranked as the first and second in terms of the average counts they obtained for each *QoS* metric (i.e., each column). We can then conclude that RuCynefin performed consistently when evaluating the robustness of the dispatchers with each individual *QoS* metric or having them combined (RQ1). Conversely, the numbers of cases being classified

[4] https://elevate.helpdocsonline.com/peters-research-cibse-modern-office-lunch-peak

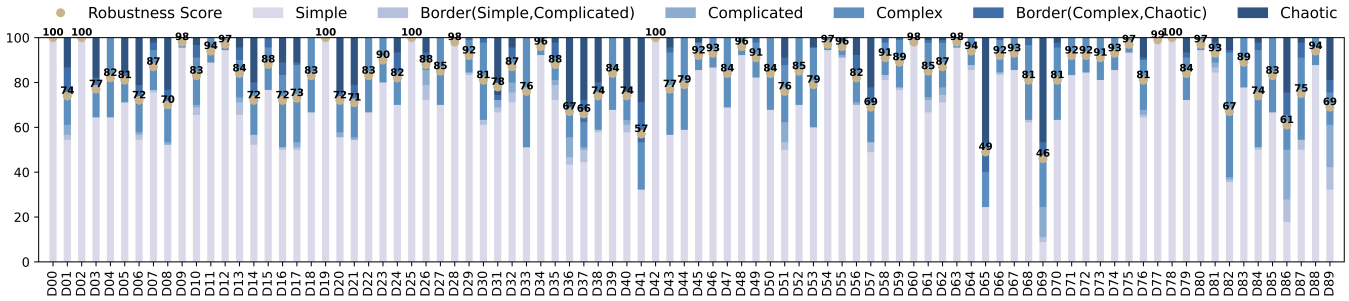

Figure 4: Summarized classification results and the overall robustness of the 90 dispatchers (with IDs going from D00 to D89) – RQ1. The bars are colored differently to represent the six Cynefin contexts.

into the other four contexts are small. Especially (as shown in Figure 5), contexts *Border(Simple, Complicated)* (row *BorSC*), *Complicated* and *Border(Complex, Chaotic)* (row *BorCC*) have the mean values all below 1. A possible explanation might be that: because two adjacent Cynefin contexts are transferable and being classified into a border context indicates the lack of confidence on which of the two adjacent contexts a case should be classified into (see Algorithm 2). Another explanation might be that the *Chaotic* context captures the situations that both *QoS* and *Time List* are significantly worsen by uncertainties (i.e., quite poor robustness of the dispatchers), while most of the dispatcher versions we investigated performed not so bad when dealing with uncertainties.

In addition, as shown in Figure 5, one can observe that the average counts in terms of the average time related *QoS* (e.g., *AWT*) are higher (or lower) than that in terms of the longest time related *QoS* (e.g., *LWT*) for the *Simple* context (or the *Complex* context). Note that for *Simple*, a higher count is preferred, while a lower count is preferred for *Complex*. This observation suggests that the 90 dispatcher versions performed relatively more robust in terms of the average time related *QoS* than the longest time related *QoS*. Domain experts from Orona suggested that, in general, most of the passengers' trips are not affected by the uncertain factors; however, there might be trips of certain passengers being affected by uncertainties, which likely results in observed low robustness in terms of the longest related *QoS* metrics (e.g., *LWT*).

Conclusion for RQ2: RuCynefin's performance is consistent when evaluating the robustness of the dispatchers with each individual *QoS* metric or having them combined. **The *AWT* and *ATD* robustness are impacted relatively less by uncertainties than the other *QoS* specific robustness. Thus, we recommend using RuCynefin to optimize a dispatcher's robustness under uncertainties with respect to a particular *QoS* of interest.**

4.2.3 Results of RQ3 - Comparing the dispatchers across the uncertain situations. Figure 6 presents the violin plots of the counts classified into a Cynefin context under each uncertain situation. To compare the robustness under each uncertain situation, we calculate the robustness score, in terms of all the *QoS* metrics, for each dispatcher, under each uncertain situation. Results are shown in the last row of Figure 6, from which we can observe that the average

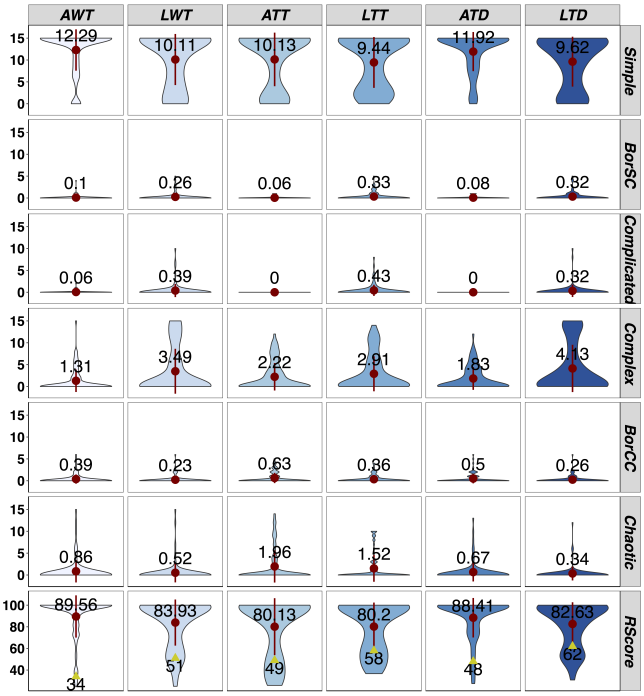

Figure 5: Distributions (in violin plots) of the counts for each Cynefin context (upper 6 rows) and robustness scores (*RScore*, the last row), in terms of each *QoS*, across the 90 dispatchers – RQ2. A red dot with a line represents a descriptive statistics (mean±standard deviation). A yellow triangle in the *RScore* row shows the number of dispatchers performed less robust than *Robustestly-behaved dispatcher* in terms of a *QoS*.

robustness scores under uncertain situations *usC*, *usM* and *usM-C* are below 80, which is lower than that under the other uncertain situations, implying that under *usC*, *usM* and *usM-C* the dispatchers performed worse. The plausible explanation is that *usC*, *usM* and *usM-C* lead to more cases being classified into the Cynefin contexts other than *Simple*; as shown in the first row of Figure 6, the average counts under *usC*, *usM* and *usM-C* are the least (i.e., below 3.9). One can also observe that, based on the average robustness score, *usL-U* is ranked as the first, followed by *usM-L-U* and *usC-L-U*. It means

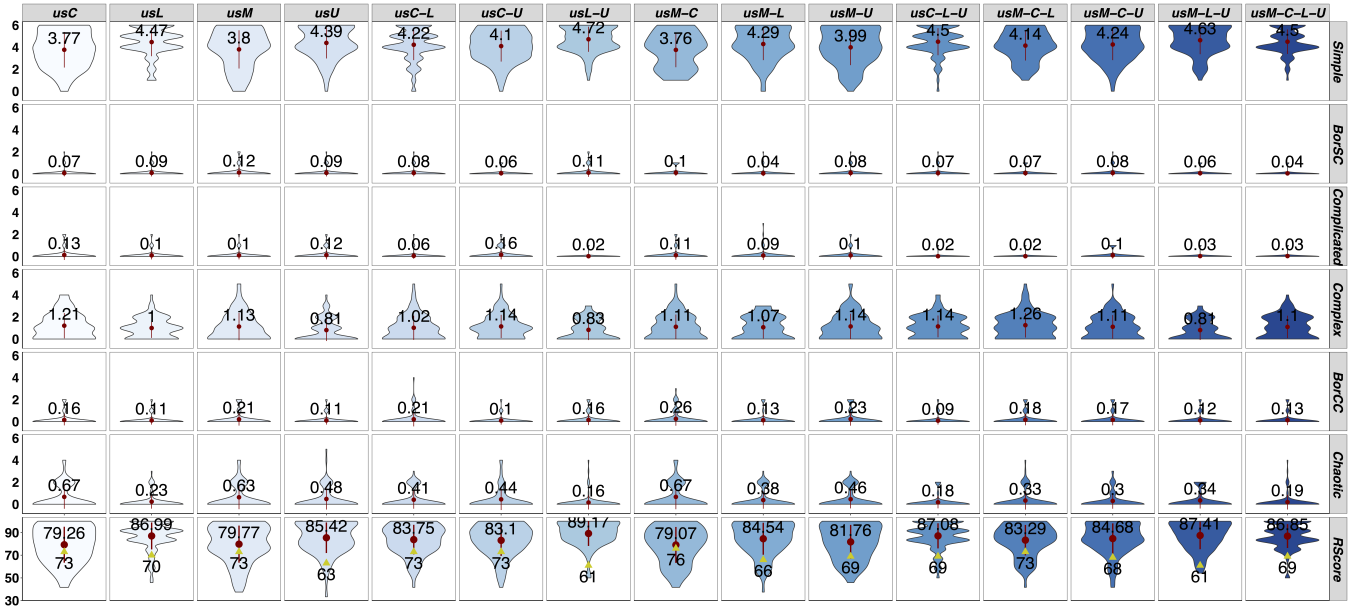

Figure 6: Distributions (in violin plots) of the counts for each Cynefin context (upper 6 rows) and robustness scores (bottom row), under each uncertain situation, across the 90 dispatchers – RQ3. A red dot with a line represents a descriptive statistics (mean±standard deviation). A yellow triangle in *RScore* row shows the number of dispatchers performed less robust than the *Robustestly-behaved dispatcher* under a specific uncertain situation.

that the dispatchers, in general, performed relatively well, under uncertain situations involving both uncertain factors *Loading Time* and *Unloading Time*, which, to an extent, mitigates the negative impact of the other uncertain factors such as *Mass* and *Capacity Factor* on the dispatchers' performance.

In addition, we compared the robustness score of each dispatcher with that of *Robustestly-behaved dispatcher*, under each uncertain situation. The number of dispatchers performed less robust than *Robustestly-behaved dispatcher* are summarized in row *RScore* in Figure 6. One can observe that more than 60 dispatchers performed less robust than *Robustestly-behaved dispatcher* under each uncertain situation. Therefore, ideally when resources allow, all the uncertain situations are recommended to be included in the systemic and comprehensive robustness assessment of a dispatcher. Such information provides elevator engineers insights to design uncertain situation-oriented robustness optimization strategies.

Furthermore, to check whether interactions of a higher number of uncertain factors lead to higher (worse) robustness, we analysed the correlation between the number of uncertain factors and the robustness of the dispatchers by comparing the average robustness scores under different t-way interactions, e.g., 2-way vs. 3-way, of uncertain factors. Results indicate that interactions of a higher number of uncertain factors do not necessarily lead to higher (or worse) robustness. For example, the average robustness score of the dispatchers under *usC-L* is higher than that under *usC*, while lower than that under *usL*.

Conclusion for RQ3: Different uncertain factors and their interactions have different extents of impact on the robustness of the dispatchers. Uncertain factors *Capacity Factor*, *Mass* and their interaction have relatively higher impact on the robustness than the others. Moreover, interactions of a higher number of uncertain factors do not necessarily lead to poorer robustness, as we initially expected. **Thus, we recommend prioritizing the assessment of robustness in the following order, based on available resources: 1) *Capacity Factor*, *Mass* and their interaction; 2) interactions among a lower number of uncertain factors; 3) interactions among a higher number of uncertain factors.**

4.2.4 Results of RQ4 - Assessing the effect of the two-way interaction of the uncertain situations and QoS metrics on the robustness of the dispatchers. To study the effect of the two-way interaction, we analyzed the classification results of the 90 dispatchers, under the contexts of each *QoS* metric and uncertain situation. We named each context as a *classification case* for convenience. Due to space limitation, we only present the counts of the *Simple* context for each *classification case* in Table 1. For instance, for the *classification case* of *usC* in terms of *AWT* (expressed as *usC_AWT* in short), 69 dispatchers are classified into the *Simple* context. From the table, we can observe that the counts with respect to *usL_ATD*, *usC-L-U_ATD* and *usM-C-L-U_ATD* are greater than or equal to 80, implying that the total counts of the other contexts (other than *Simple*) are less than 10. Conversely, for cases *usC_LTD*, *usM_LTT* and *usM-C_LTD*, the counts of the *Simple* context are less than 50, which tells that more than 40 dispatchers are classified into the other Cynefin contexts and therefore indicates a relatively poor robustness.

Table 1: Counts of the *Simple* context for each *classification case* across the 90 dispatchers - RQ4. The bold values represent the three largest (in black) and the three smallest (in blue) values among all the counts.

QoS	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
AWT	69	69	73	76	75	71	73	73	76	72	77	76	74	76	76
LWT	57	57	59	62	62	62	61	59	63	59	62	58	59	68	62
ATT	55	64	51	60	59	57	69	52	61	56	65	62	63	71	67
LTT	51	66	**43**	59	50	60	74	50	55	53	58	55	62	59	55
ATD	60	**80**	64	74	76	64	78	58	75	62	**82**	68	71	79	**82**
LTD	**47**	66	52	64	58	55	70	**46**	56	57	61	54	53	64	63

#1 usC #2 usL #3 usM #4 usU #5 usC-L #6 usC-U #7 usL-U #8 usM-C #9 usM-L #10 usM-U #11 usC-L-U #12 usM-C-L #13 usM-C-U #14 usM-L-U #15 usM-C-L-U

Conclusion for RQ4: RuCynefin can effectively evaluate the robustness in terms of a specific *QoS* against a specific uncertain situation; thereby providing feedback on which *QoS* against which uncertain situation should be prioritized for optimization. **We recommend that engineers shall put more focus on the robustness in terms of *LTD* when facing uncertainties caused by *Capacity Factor* itself or its interactions with *Mass*, and the robustness in terms of *LTT* under uncertainties caused by *Mass* only.**

4.3 Threats to Validity

To avoid unrealistic uncertainties, we generate values of passengers' uncertain attributes, e.g., *Mass*, within reasonable intervals around the recommended values from the CIBSE Guide D [9]. In addition, we also controlled the average value of the generated values of all the passengers being the same as the recommended average in the CIBSE Guide D [9] to avoid deviating too much from the standard. We carefully selected intervals around the recommended values in the CIBSE Guide D [9] for each uncertain factor and we confirmed each interval with domain experts. With these intervals, the generated uncertainties are closer to passengers' uncertainties in real life, thus more realistic. Moreover, we chose effective measures *QoS* [9] and *Time List*, which are comparable across all the dispatchers.

To minimize threats related to SiL simulation settings, we configured building and elevator with settings from the CIBSE Guide D [9]. Correspondingly, we used CIBSE modern office lunch peak traffic template from Elevate, which models traffic flow during lunch peak based on extensive traffic surveys conducted by Peters Research Ltd[5]. To reduce the threat related to dispatchers used, we used 90 dispatchers from Orona rather than simulator's built-in dispatcher. Since we only used one traffic template, the generalization of our results might be a threat. However, investigating more templates requires more large-scale experiments, and for one template, we already performed $90 \times (15 \times 10 + 1) = 13590$ simulations, each of which simulated 1409 passengers. We are confident that RuCynefin can be used to evaluate other template specific robustness as well. Additional experiments will be conducted in the future.

5 PRACTICAL USAGES

Identifying dispatchers with poor robustness. We investigated the overall robustness of the 90 dispatchers, which exhibit diverse robustness (Figure 4). A few dispatchers achieved robustness as high as *Robustestly-behaved dispatcher*, while some exhibited extremely poor robustness. Thus, it is critical that elevator engineers categorize dispatchers' robustness with RuCynefin before deploying them, after fixing a bug, or any other changes being performed. In addition, sorting dispatchers with robustness score can help elevator engineers identify dispatchers with poor robustness. This could be very difficult if the categorization is performed by engineers manually.

Facilitating Uncertainty-aware Test Generation and Prioritization. The results of RQ3 (Section 4.2.3) showed that some uncertain factors (e.g., *Mass* and *Capacity Factor*) and their interactions have more impact on the robustness. Consequently, the classification results, together with the robustness scores, can help elevator test engineers to generate or prioritize test cases for testing a version of the dispatcher by focusing on uncertain situations with high impact on the robustness, e.g., *usM-C*.

Assessing the impact of Uncertain Situations on the Degrees of the Robustness of dispatcher. Our work provides a novel method for practitioners, especially elevator designers, to identify various Cynefin contexts, aiming to study how various uncertain factors and their interactions can lead to various degrees of uncertainty in a diverse set of *QoS* attributes in a particular installation of an elevator. Identifying such contexts will help practitioners to systematically understand uncertain factors, which consequently helps engineers to improve the robustness of their dispatchers, e.g., by defining specialized uncertainty handling mechanisms corresponding to each context. Moreover, new methods can be designed to mitigate the impact of uncertainties by moving from a high impacted context (e.g., *Complex*) to a low one (e.g., *Complicated*).

Having the RuCynefin as the Backbone of Uncertainty-wise SiL. Our RuCynefin has the potential to be part of the SiL simulation software (e.g., Elevate) for industrial elevators that can consider uncertainties explicitly during the SiL configuration. More specifically, for instance, Elevate, with the help of our solution, could generate uncertain traffic profiles that might be interesting for elevator designers. After that, Elevate would execute them, classify simulation results into the Cynefin contexts, and returns the classification results together with the robustness scores to elevator designers. They can then be informed to pay special attention to uncertain situations being classified into *Complex* and *Chaotic* contexts. Doing so can facilitate the early detection of uncertain situations, consequently ensuring the development of elevator systems with high robustness.

Moving Uncertain Situations from one Context to Another. We observe that going from one Cynefin context to another is possible as it can be seen that some of our *classification cases* lie on the borders of the contexts. The movement of uncertain situations from one context to another gives us insights into handling and reducing uncertainties and improving robustness of elevators. More specifically, it will help elevator engineers to develop dispatcher that can automatically make suitable scheduling decisions (also called *take actions* in Cynefin framework) according to the categorized Cynefin contexts. Such decisions will be aimed at moving an uncertain situation from a context with high impact on the robustness, e.g., *Chaotic*, to the adjacent context, i.e., *Complex* resulting in improved robustness.

[5] https://peters-research.com

Generalizing RuCynefin to other types of CPS. Similar to elevator system, many types of CPS (e.g., Autonomous Driving Systems (ADSs)) face various uncertainties from their operating environment during their operation. To generalize RuCynefin for assessing the robustness of such systems, first, in terms of *generating uncertain profiles* of RuCynefin, one needs to first identify uncertain factors (e.g., pedestrians and weather for ADSs) and quantify them, such that uncertainties can be introduced systematically into the SiL setting under which the CPS is assessed. Second, regarding *classifying uncertain situations into the Cynefin contexts*, one needs to identify *QoS* metrics. For instance, in the context of ADSs, such *QoS* metrics can be collision probability [35], time to collision, etc. Last, in terms of *calculating robustness score*, the current algorithm of RuCynefin does not need to be changed.

6 RELATED WORK

Cynefin Framework. Since the Cynefin framework was proposed [32], it has been applied in diverse contexts, such as project management [30, 42], enterprises management [31], economical problems analysis [19], healthcare [36, 50], and biomedical research [29]. In addition, Cynefin has been explored in information science and software engineering. For example, McLeod et al. [40] conducted an exploratory evaluation on using Cynefin as a research tool for identifying issues and practical strategies for managing electronic records. O'Connor et al. [44] applied Cynefin to software process management to detect important factors that have direct and significant impact on software development processes. Ilieva et al. [27] proposed a vision for applying Cynefin to assist AI bots in performing tasks. To increase the security of IoT systems, Selgert et al. [46] applied Cynefin in DevOps to help find situations that can be optimized, e.g., those under which a system falls into the Chaotic context, in the DevOps process during the design phase. Differently, RuCynefin is a novel application of the Cynefin framework in the context of assessing the robustness of industrial elevator systems.

Uncertainty-aware Software Engineering. Uncertainty is inevitable in software engineering and has been widely studied. Existing work focus on studying uncertainty in, e.g., requirements and architecture decisions [16, 34, 56, 59, 60], software development [26], software testing such as test planning [60], test model evolution [58], validation of deep learning model predictions [13], and self-adaptation problem of self-adaptive system [15, 37, 38]. In addition, uncertainty quantification for ADS testing [12] and discovering unknowns via online testing [28] were also investigated. With the increasing integration of software in complex CPS, uncertainty testing of CPS has attracted more attention and various techniques have been applied, e.g., search-based approaches [3, 33, 48, 49] and model-based testing methods [10, 11, 14]. However, recent studies focused mostly on generating [10, 11, 41, 49, 49, 57] and optimizing [3, 37, 38, 51] uncertainty-related test cases. In contrast, we focus on assessing the robustness of CPS, in the context of elevator systems, against uncertainties. Moreover, RuCynefin can be used to generate uncertainty-aware tests, which considers all possible interactions among uncertain factors – an aspect hasn't been considered by these related works. In addition, RuCynefin is equipped with a classification algorithm, which can be used to prioritize uncertainty-aware tests, i.e., uncertain situations being classified into, e.g., *Chaotic* context are high uncertain tests with more impacts on the system.

Elevator Uncertainty. Uncertainties in elevator systems have been investigated from different aspects with diverse techniques, such as 1) evaluating passengers' average traveling time under environmental uncertainties with Monte Carlo simulations [1]; 2) utilizing the Bayesian network [22] to model software uncertainty [8] with the aim of designing dispatching optimization algorithm; 3) applying Fuzzy theory to minimize waiting time [18], optimize energy consumption [17], measure passengers' satisfaction [55], etc., in the presence of passengers' uncertain waiting time [55] and calls [17, 18]; and 4) employing transfer learning to predict passengers' waiting times under uncertainties [54]. In addition, optimizing the efficiency of the friction crane brake system under braking force uncertainties also being investigated in [52]. The above mentioned works mainly focused on optimization under uncertainties, e.g., [8, 17, 18, 52]. However, the robustness of the dispatcher against passengers' uncertainties has been sufficiently investigated, which is a key indicator determining passengers' satisfaction.

Testing Elevator Systems. Various techniques have been applied to test elevator systems, e.g., machine learning [4], metamorphic testing [5, 6], and model-based development [43]. However, the literature mostly focuses on testing elevator systems' functionalities such as braking [25, 39, 45, 53], and controllers [43]. Very few works focus on testing elevator systems' performance, and among those, we noticed one work from [2], which is about testing the performance of elevator group control algorithms. In contrast, our work focuses on testing the robustness of elevator systems against passengers' uncertainties.

7 CONCLUSION AND FUTURE WORK

To systematically investigate the robustness of industrial elevator dispatchers against various passengers' uncertainties, in this paper, we proposed RuCynefin, which classifies uncertain situations into the Cynefin contexts, and further calculates robustness scores. To evaluate RuCynefin, we conducted experiments with an industrial case study of 90 elevator dispatchers from Orona – an industrial elevator developer in Europe. We also provided insights on the practical usages of RuCynefin. In the future, we plan to further study uncertain situations in the vertical transportation domain and beyond, and devise novel methods to deal with uncertain situations automatically, aiming to improve the overall robustness of software systems against uncertainties.

ACKNOWLEDGMENTS

All authors are supported by the Adeptness project funded by the European Union's Horizon 2020 programme under Grant Agreement no. 871319. Tao Yue and Shaukat Ali are also supported by the Co-evolver project (Project #286898) funded by the research council of Norway. Aitor Arrieta is part of the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1519-22), supported by the Department of Education, Universities and Research of the Basque Country. We thank Orona for providing us with industrial elevator dispatchers to perform this case study. The results and discussions do not, in any way, represent the quality of Orona's elevators.

REFERENCES

[1] Lutfi Al-Sharif, OF Abdel Aal, and AM Abu Alqumsan. 2013. Evaluating the elevator passenger average travelling time under incoming traffic conditions using analytical formulae and the Monte Carlo method. *Elevator World* 61, 6 (2013), 110–123.

[2] Lutfi Al-Sharif, Zaid Jaber, Jamal Hamdan, and Anas Riyal. 2016. Evaluating the performance of elevator group control algorithms using a three-element new paradigm. *Building services engineering research and technology* 37, 5 (2016), 597–613. https://doi.org/10.1177/0143624416652182

[3] Shaukat Ali, Hong Lu, Shuai Wang, Tao Yue, and Man Zhang. 2017. Uncertainty-wise testing of cyber-physical systems. In *Advances in Computers*. Vol. 107. Elsevier, 23–94. https://doi.org/10.1016/bs.adcom.2017.06.001

[4] Aitor Arrieta, Jon Ayerdi, Miren Illarramendi, Aitor Agirre, Goiuria Sagardui, and Maite Arratibel. 2021. Using Machine Learning to Build Test Oracles: an Industrial Case Study on Elevators Dispatching Algorithms. In *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*. 30–39. https://doi.org/10.1109/AST52587.2021.00012

[5] Jon Ayerdi, Sergio Segura, Aitor Arrieta, Goiuria Sagardui, and Maite Arratibel. 2020. QoS-aware Metamorphic Testing: An Elevation Case Study. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 104–114. https://doi.org/10.1109/ISSRE5003.2020.00019

[6] Jon Ayerdi, Valerio Terragni, Aitor Arrieta, Paolo Tonella, Goiuria Sagardui, and Maite Arratibel. 2021. Generating metamorphic relations for cyber-physical systems with genetic programming: an industrial case study. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1264–1274.

[7] Josiah A. B. S. Dykstra and Stephen R. Orr. 2016. Acting in the unknown: the cynefin framework for managing cybersecurity risk in dynamic decision making. In *2016 International Conference on Cyber Conflict (CyCon U.S.)*. 1–6. https://doi.org/10.1109/CYCONUS.2016.7836616

[8] Yerzhigit Bapin, Kanat Alimanov, and Vasilios Zarikas. 2020. Camera-driven probabilistic algorithm for multi-elevator systems. *Energies* 13, 23 (2020), 6161. https://doi.org/10.3390/en13236161

[9] Gina Barney. 2010. *Transportation systems in buildings : CIBSE Guide D: 2010*. London: Chartered Institution of Building Services Engineers.

[10] Matteo Camilli, Carlo Bellettini, Angelo Gargantini, and Patrizia Scandurra. 2018. Online Model-Based Testing under Uncertainty. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. 36–46. https://doi.org/10.1109/ISSRE.2018.00015

[11] Matteo Camilli, Angelo Gargantini, Patrizia Scandurra, and Catia Trubiani. 2021. Uncertainty-aware Exploration in Model-based Testing. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. 71–81. https://doi.org/10.1109/ICST49551.2021.00019

[12] Ferhat Ozgur Catak, Tao Yue, and Shaukat Ali. 2021. Prediction surface uncertainty quantification in object detection models for autonomous driving. In *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*. IEEE, 93–100.

[13] Ferhat Ozgur Catak, Tao Yue, and Shaukat Ali. 2022. Uncertainty-aware prediction validator in deep learning models for cyber-physical system data. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 4 (2022), 1–31.

[14] Amrita Chatterjee and Hassan Reza. 2020. Toward Modeling and Verification of Uncertainty in Cyber-Physical Systems. In *2020 IEEE International Conference on Electro Information Technology (EIT)*. 568–576. https://doi.org/10.1109/EIT48999.2020.9208273

[15] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. 2011. Taming uncertainty in self-adaptive software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 234–244.

[16] Naeem Esfahani, Kaveh Razavi, and Sam Malek. 2012. Dealing with uncertainty in early software architecture. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. 1–4.

[17] Joaquín Fernández, Pablo Cortés, José Guadix, and Jesús Muñuzuri. 2013. Dynamic fuzzy logic elevator group control system for energy optimization. *International Journal of Information Technology & Decision Making* 12, 03 (2013), 591–617. https://doi.org/10.1142/S0219622013500223

[18] J Fernandez, Pablo Cortés, Jesús Muñuzuri, and José Guadix. 2013. Dynamic fuzzy logic elevator group control system with relative waiting time consideration. *IEEE transactions on industrial electronics* 61, 9 (2013), 4912–4919. https://doi.org/10.1109/TIE.2013.2289867

[19] Davide Fierro, Stefano Putino, and Lucio Tirone. 2018. The Cynefin framework and technical competencies: a new guideline to act in the complexity. In *INCOSE International Symposium*, Vol. 28. Wiley Online Library, 532–552. https://doi.org/10.1002/j.2334-5837.2018.00498.x

[20] Simon French. 2013. Cynefin, statistics and decision analysis. *Journal of the Operational Research Society* 64, 4 (2013), 547–561.

[21] Simon French. 2015. Cynefin: uncertainty, small worlds and scenarios. *Journal of the Operational Research Society* 66, 10 (2015), 1635–1645. https://doi.org/10.1057/jors.2015.21

[22] Nir Friedman, Dan Geiger, and Moises Goldszmidt. 1997. Bayesian network classifiers. *Machine learning* 29, 2 (1997), 131–163.

[23] Aitor Gartziandia, Aitor Arrieta, Jon Ayerdi, Miren Illarramendi, Aitor Agirre, Goiuria Sagardui, and Maite Arratibel. 2022. Machine learning-based test oracles for performance testing of cyber-physical systems: An industrial case study on elevators dispatching algorithms. *J Softw Evol Proc* (2022).

[24] Aitor Gartziandia, Jon Ayerdi, Aitor Arrieta, Shaukat Ali, Tao Yue, Aitor Agirre, Goiuria Sagardui, and Maite Arratibel. 2021. Microservices for Continuous Deployment, Monitoring and Validation in Cyber-Physical Systems: an Industrial Case Study for Elevators Systems. In *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 46–53.

[25] Dimitrios Giagopoulos, Iraklis Chatziparasidis, and Nickolas S Sapidis. 2018. Dynamic and structural integrity analysis of a complete elevator system through a Mixed Computational-Experimental Finite Element Methodology. *Engineering Structures* 160 (2018), 473–487.

[26] Hamdy Ibrahim, Behrouz H Far, Armin Eberlein, and Yousef Daradkeh. 2009. Uncertainty management in software engineering: Past, present, and future. In *2009 Canadian Conference on Electrical and Computer Engineering*. IEEE, 7–12.

[27] Roumiana Ilieva, Kiril Anguelov, and Mario Nikolov. 2018. A Cynefin Framework for Agile Decision Making of AI BOTS. In *2018 International Conference on High Technology for Sustainable Development (HiTech)*. IEEE, 1–4. https://doi.org/10.1109/HiTech.2018.8566411

[28] Ruihua Ji, Zhong Li, Shouyu Chen, Minxue Pan, Tian Zhang, Shaukat Ali, Tao Yue, and Xuandong Li. 2018. Uncovering unknown system behaviors in uncertain networks with model and search-based testing. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 204–214.

[29] Gerd Kempermann. 2017. Cynefin as reference framework to facilitate insight and decision-making in complex contexts of biomedical research. *Frontiers in neuroscience* (2017), 634. https://doi.org/10.3389/fnins.2017.00634

[30] Vladimir I Khabarov and Vladimir V Volodin. 2020. UNCERTAINTY AS THE KEY FACTOR TO SELECT PROJECT MANAGEMENT TOOLS. *PalArch's Journal of Archaeology of Egypt/Egyptology* 17, 10 (2020), 2397–2407.

[31] Gladius David Kulothungan, Charles Oham, and Puspa Jirel. 2019. Managing uncertainty in small and micro social enterprises: A Cynefin-Bricolage framework. In *Handbook of Research on Value Creation for Small and Micro Social Enterprises*. IGI Global, 327–348.

[32] Cynthia F Kurtz and David J Snowden. 2003. The new dynamics of strategy: Sense-making in a complex and complicated world. *IBM systems journal* 42, 3 (2003), 462–483.

[33] Jaekwon Lee, Seung Yeob Shin, Shiva Nejati, Lionel C Briand, and Yago Isasi Parache. 2020. Schedulability Analysis of Real-Time Systems with Uncertain Worst-Case Execution Times. *arXiv preprint arXiv:2007.10490* (2020).

[34] Emmanuel Letier, David Stefan, and Earl T Barr. 2014. Uncertainty, risk, and information value in software requirements and architecture. In *Proceedings of the 36th International Conference on Software Engineering*. 883–894.

[35] Chengjie Lu, Yize Shi, Huihui Zhang, Man Zhang, Tiexin Wang, Tao Yue, and Shaukat Ali. 2022. Learning Configurations of Operating Environment of Autonomous Vehicles to Maximize their Collisions. *IEEE Transactions on Software Engineering* (2022).

[36] Christian Lunghi and Francesca Baroni. 2019. Cynefin framework for evidence-informed clinical reasoning and decision-making. *J Am Osteopath Assoc* 119, 5 (2019), 312–321.

[37] Tao Ma, Shaukat Ali, and Tao Yue. 2021. Testing self-healing cyber-physical systems under uncertainty with reinforcement learning: an empirical study. *Empirical Software Engineering* 26, 3 (2021), 1–54. https://doi.org/10.1007/s10664-021-09941-z

[38] Tao Ma, Shaukat Ali, Tao Yue, and Maged Elaasar. 2019. Testing self-healing cyber-physical systems under uncertainty: a fragility-oriented approach. *Software Quality Journal* 27, 2 (2019), 615–649. https://doi.org/10.1007/s11219-018-9437-3

[39] Xiaolong Ma, Gen Pan, Peng Zhang, Qing Xu, Xi Shi, Zeliang Xiao, and Yunting Han. 2021. Experimental evaluation of braking pad materials used for high-speed elevator. *Wear* (2021), 203872. https://doi.org/10.1016/j.wear.2021.203872

[40] Julie McLeod and Sue Childs. 2013. The Cynefin framework: A tool for analyzing qualitative data in information science? *Library & Information Science Research* 35, 4 (2013), 299–309.

[41] Claudio Menghi, Shiva Nejati, Khouloud Gaaloul, and Lionel C Briand. 2019. Generating automated and online test oracles for simulink models with continuous and uncertain behaviors. In *Proceedings of the 2019 27th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 27–38. https://doi.org/10.1145/3338906.3338920

[42] Andreas Nachbagauer. 2021. Managing complexity in projects: Extending the Cynefin framework. *Project Leadership and Society* 2 (2021), 100017. https://doi.org/10.1016/j.plas.2021.100017

[43] Carlos Fernando Nicolas, Iban Ayestaran, Imanol Martinez, and Patricia Franco. 2016. Model-Based Development of an FPGA Encoder Simulator for Real-Time Testing of Elevator Controllers. In *2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC)*. 53–60. https://doi.org/10.1109/ISORC.

2016.17

[44] Rory V O'Connor and Marion Lepmets. 2015. Exploring the use of the cynefin framework to inform software development approach decisions. In *Proceedings of the 2015 International Conference on Software and System Process*. 97–101.

[45] Qifeng Peng, Zhongxing Li, Hong Yuan, Guojian Huang, Shanqing Li, and Xueli Sun. 2018. A model-based unloaded test method for analysis of braking capacity of elevator brake. *Advances in Materials Science and Engineering* 2018 (2018). https://doi.org/10.1155/2018/8047490

[46] Franklin Selgert. 2020. Cynefin framework, devops and secure IoT. In *International Conference on Computer Safety, Reliability, and Security*. Springer, 255–265. https://doi.org/10.1007/978-3-030-55583-2_19

[47] Saeed Shalbafan, Elyssebeth Leigh, Julien Pollack, and Shankar Sankaran. 2018. Decision-making in project portfolio management: using the Cynefin framework to understand the impact of complexity. *International Research Network on Organizing by Projects* (2018).

[48] Seung Yeob Shin, Shiva Nejati, Mehrdad Sabetzadeh, Lionel C Briand, and Frank Zimmer. 2018. Test case prioritization for acceptance testing of cyber physical systems: a multi-objective search-based approach. In *Proceedings of the 27th acm sigsoft international symposium on software testing and analysis*. 49–60. https://doi.org/10.1145/3213846.3213852

[49] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. 2022. Efficient Online Testing for DNN-Enabled Systems using Surrogate-Assisted and Many-Objective Optimization. In *Proceedings of the 44th International Conference on Software Engineering (ICSE'22)*. ACM.

[50] Eric K Van Beurden, Annie M Kia, Avigdor Zask, Uta Dietrich, and Lauren Rose. 2013. Making sense in a complex landscape: how the Cynefin Framework from Complex Adaptive Systems Theory can inform health promotion practice. *Health promotion international* 28, 1 (2013), 73–83.

[51] Neil Walkinshaw and Gordon Fraser. 2017. Uncertainty-Driven Black-Box Test Data Generation. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 253–263. https://doi.org/10.1109/ICST.2017.30

[52] Piotr Wolszczak, Pawel Lonkwic, Americo Cunha, Grzegorz Litak, and Szymon Molski. 2019. Robust optimization and uncertainty quantification in the nonlinear mechanics of an elevator brake system. *Meccanica* 54, 7 (2019), 1057–1069.

[53] Aizhong Wu, Xi Shi, Lin Weng, and Dingyu Hu. 2020. Thermo-mechanical modeling and transient analysis of frictional braking of elevator safety gear. *Journal of Thermal Stresses* 43, 12 (2020), 1467–1486. https://doi.org/10.1080/01495739.2020.1820921

[54] Qinghua Xu, Shaukat Ali, Tao Yue, and Maite Arratibel. 2022. Uncertainty- Aware Transfer Learning to Evolve Digital Twins for Industrial Elevators. In *Proceedings of the 30th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*.

[55] Dong Mei Yan and Li Liu. 2014. Research on elevator group control uncertainty. In *Applied Mechanics and Materials*, Vol. 602. Trans Tech Publ, 874–877.

[56] Huihui Zhang, Man Zhang, Tao Yue, Shaukat Ali, and Yan Li. 2020. Uncertainty-wise requirements prioritization with search. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 1 (2020), 1–54.

[57] Man Zhang, Shaukat Ali, and Tao Yue. 2019. Uncertainty-wise test case generation and minimization for cyber-physical systems. *Journal of Systems and Software* 153 (2019), 1–21.

[58] Man Zhang, Shaukat Ali, Tao Yue, and Roland Norgre. 2017. Uncertainty-wise evolution of test ready models. *Information and Software Technology* 87 (2017), 140–159.

[59] Man Zhang, Tao Yue, Shaukat Ali, Bran Selic, Oscar Okariz, Roland Norgre, and Karmele Intxausti. 2018. Specifying uncertainty in use case models. *Journal of Systems and Software* 144 (2018), 573–603.

[60] Hadar Ziv, Debra Richardson, and René Klösch. 1997. The uncertainty principle in software engineering. In *submitted to Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*.