Faulty Requirements Made Valuable: On the Role of Data Quality in Deep Learning

Harshitha Challa*, Nan Niu*, and Reese Johnson†
* Department of Electrical Engineering and Computing Systems, University of Cincinnati, USA
† Metropolitan Sewer District of Greater Cincinnati, USA
challava@mail.uc.edu, nan.niu@uc.edu, reese.johnson@cincinnati-oh.gov

***Abstract*—Large collections of data help evolve deep learning into the state-of-the-art in solving many artificial intelligence problems. However, the requirements engineering (RE) community has yet to adapt to such sweeping changes caused exclusively by data. One reason is that the traditional requirements quality like unambiguity becomes less applicable to data, and so do requirements fault detection techniques like inspections. In this paper, we view deep learning as a class of machines whose effects must be evaluated with direct consideration of inherent data quality attributes: accuracy, consistency, currentness, etc. We substantiate this view by altering stationarity of the multivariate time-series data, and by further analyzing how the stationarity changes affect the behavior of a recurrent neural network in the context of predicting combined sewer overflow. Our work sheds light on the active role RE plays in deep learning.**

***Index Terms*—Data quality, stationarity, recurrent neural network, metamorphic testing, smart sewer systems.**

I. Introduction

Requirements faults are common. Basili and Perricone [1] reported that 48% of the failures observed in a medium-scaled software project were "attributed to incorrect or misinterpreted functional specifications or requirements." Perry and Stieg [2] showed that 79.6% of interface faults were due to incomplete or omitted requirements. While not all requirements faults are equal, the summary statistics are comparable in safety and mission critical systems. Lutz [3] analyzed errors in two NASA spacecraft software systems (Voyager and Galileo), and found that "the primary cause of safety-related interface faults is misunderstood hardware interface specifications (67% on Voyager; 48% on Galileo)", and "the primary cause of safety-related functional faults is errors in recognizing (understanding) the requirements (62% on Voyager; 79% on Galileo)."

Faulty requirements are costly. Brooks [4] asserted that "the hardest single part of building a software system is deciding precisely what to build", because "no other part of the work so cripples the resulting system if done wrong." With data support, Boehm [5] suggested that: "Clearly, it pays off to invest effort in finding requirements errors early and correcting them in, say, 1 man-hour rather than waiting to find the error during operations and having to spend 100 man-hours correcting it."

As a community, requirements engineering (RE) has paid substantial attention to characterizing what constitutes a desirable level of quality. Most notably, the IEEE 830 standard [6] defines a good software requirements specification (SRS) to be correct, unambiguous, complete, consistent, ranked for importance and/or stability, verifiable, modifiable, and traceable. In case an SRS lacks these characteristics, systematic and manual reviews (particularly requirements inspections [7, 8, 9, 10, 11]) shall be carried out to identify as many faults as possible.

When building deep learning (DL) systems, however, a new class of requirements becomes crucial: data. DL is data hungry, requiring large collections of data to train a neural network to classify objects, translate languages, or learn to perform other tasks. Because the intended functionality like classification has shared understandings in artificial intelligence (AI), no SRS seems to ever exist for DL yet algorithmic essentials like back-propagation are fairly well developed. An underlying driver of DL becoming the state-of-the-art in practically solving many AI problems, such as computer vision and natural language processing, is the availability of vast volumes of data.

Unlike an SRS that needs to be engineered, data arise naturally: images and the objects in them, natural language documents and the sentiments in them, etc. Even if requirements engineers understand the importance of correctness, unambiguity, completeness, and the like [6], these quality characteristics are less applicable to data, making this part of RE knowledge, including the inspection techniques in the RE toolbox, rather obsolete in developing DL solutions.

In this paper, we argue that data are important requirements for DL and the wisdom of requirements quality shall play an active part in gauging DL's fitness for purpose. Innovative ways, other than inspecting requirements faults, must be sought to cope with data faults. We revisit Jackson's foundational work on the environment and the machine [12], and further elaborate our view on the data implications in building the class of DL machines.

The chief contribution of our paper is a novel vision: Instead of striving for uncovering and subsequently eliminating faulty requirements, we shall recognize the inherent data quality attributes and then embrace the faulty data in new ways. To substantiate the view, we carry out a case study with the Metropolitan Sewer District of Greater Cincinnati (MSDGC) where the prediction in the context of combined sewer overflow is experimented with recurrent neural networks (RNNs). The results demonstrate the value of RNN testing when the stationarity of the time-series data is altered.

The remainder of the paper is organized as follows. Section II provides the background on faulty requirements. Sec-

tion III distills data quality in building DL machines. Section IV describes our MSDGC case study. Section V presents concluding remarks.

II. Requirements Faults and Inspections

Requirements are located in the environment, which is in contrast to the machine to be constructed [12]. Figure 1 depicts the conceptual distinction where environment and machine overlap. Although the machine refers to the software being built (e.g., a library information management system or the controller for NASA's Voyager), the requirements do not directly concern the machine. They concern the environment in which the effects of the machine—once built and deployed—will be observed and evaluated.

While Figure 1 contextualizes requirements as problem theories, a specification bridges problem and solution. An SRS, therefore, documents key issues including functionality (what the software is supposed to do), external interfaces (how the software interacts with people, hardware, and other software), nonfunctional attributes (what the speed, availability, security, etc. considerations are), and design constraints imposed on an implementation (whether any required standards, policies, resource limits, etc. are in effect) [6]. Engineering requirements is often about figuring out these issues.

Specifying requirements, like many other RE tasks, is a human-centric activity and prone to error. Walia and Carver [13] identified 119 requirements errors in their literature review, and grouped them into three types: people errors (arising from fallibilities of the people involved in RE), process errors (arising from selecting inappropriate steps or procedures for achieving the desired RE outcomes), and documentation errors (arising from mistakes in organizing and specifying the requirements, regardless of whether the engineers properly understood the requirements). The review by Anu *et al.* [14] revealed three reasons causing the human errors: slip (lack of attention), lapse (memory-related failure), and mistake (inadequate planning).

These human errors manifest in an SRS as requirements faults. For example, if the SRS does not reflect the stakeholders' actual needs, then the requirements are incorrect. Other faults are ambiguity, incompleteness, inconsistency, etc. [6]. Because the software built to satisfy faulty requirements would result in failure (e.g., unexpected operation or invalid output), they must be uncovered and addressed to improve the quality of the machine being constructed.

Inspection is among the effective methods to uncover requirements faults. Not only are different inspections introduced (e.g., checklist, scenario-based, or perspective-based reading), but rigorous experiments and replications are also carried out, building a rich body of empirical knowledge [7, 8, 9, 10, 11]. For example, inspectors whose degree is in a field related to software engineering are found to be less effective in identifying requirements faults than inspectors whose degrees are in other fields [10, 11], highlighting the risks of forming a homogenous, biased inspection team.

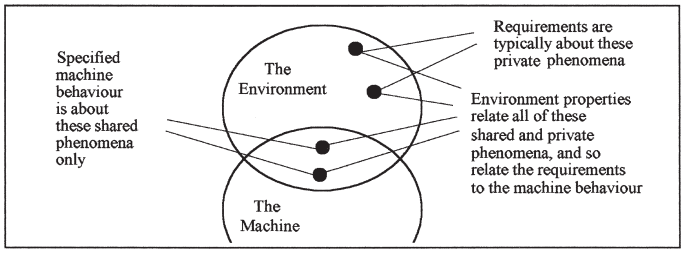


Fig. 1. The environment is the part of the world with which the machine will interact (*adopted from [12]*).

In summary, when the requirements are externalized in an SRS [15], in user stories [16], or in other forms [17], errors manifest in them resulting in faults. These faults, if not detected, would lead to failures of the constructed machine, and if detected late in the software life cycle, would be the most difficult and costly to rectify [4, 5, 18]. Therefore, static reading techniques like inspections, as well as automated tools [19, 20, 21, 22], shall be employed to ensure the requirements artifacts are of high quality on their own.

III. Data Quality and Deep Learning

Compared to requirements artifacts like an SRS or user stories, data in DL development are very much taken for granted. This section revisits Jackson's conceptualization to position data in DL development and to further argue that checking data quality in the same static manner as inspecting SRS is no longer appropriate, motivating our new way of exploiting data quality.

We regard DL as a class of software which can be conceptualized as "machine" in Jackson's model [12]. DL is part of a broader family of machine learning methods that develop algorithms based on sample data known as training data in order to make predictions or decisions without being explicitly programmed to do so [23]. The architectural backbone of DL is artificial neural network, inspired by information processing and distributed communication nodes in biological systems. Compared to traditional machine learning that requires manual feature selections, DL consists of multiple hidden layers to discover intricate structure in large data sets more automatically by using backpropagation to indicate how a machine should change its internal parameters [24].

Clearly, backpropagation, internal parameters, and hidden layers all belong to the private part of the machine. However, DL is to be applied in fields like computer vision, speech recognition, machine translation, playing board games, and medical diagnosis, all of which overlap the "environment" according to Jackson [12]. Take playing board games as an example, no matter how the DL machine is built, its true effects have to be observed and evaluated in the environment, i.e., by competing against human players.

Winning a board game may be an important entertainment goal, other requirements concern safety, security, business, healthcare, etc. All requirements, according to Jackson [12], are typically about the private phenomena of the environment

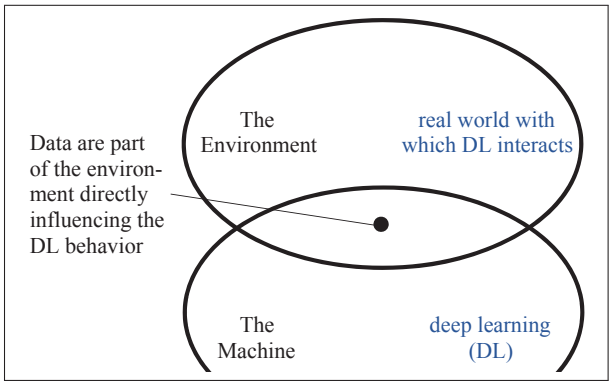


Fig. 2. Data lie in the overlap of the DL machine and its environment.

and can be stated entirely without reference to the machine. For a board game, winning by the rules (achieving a desired state, respecting the timing constraints placed on the moves, etc.) is the requirement, independent of how specific DL machines are trained (e.g., whether reinforcement learning is used to maximize the objective function). The data, or more specifically, the training data of previously played games, exist without any DL machine being built. Therefore, data are *not* in the private part of the machine.

When constructing the DL machine, data are *not* in the private part of the environment, either. This is because data, especially the training data, directly influence the DL behavior. Thus, we position data in the overlap of the DL machine and its environment, as shown in Figure 2. This raises the question of whether data are given (being indicative [12]) as other parts of the environment like the board games and their rules, or data are constructed (being optative [12]) as other parts of the machine like the hidden layers and their nonlinear activations.

We argue the answer is both. While the training data are given, once the constructed DL machine is deployed in the lab or in the field, many more data will be outputted from the machine. These newly generated data, though left the machine, can be used as positive or negative examples for training better DL machines. It is this dynamic consuming-producing relationship of DL that makes data fit in the intersection of the environment and the machine.

Being data hungry, DL suffers from the "garbage in, garbage out" problem, and therefore requires high-quality input data to train the neural network for making good classifications and predictions, or learning to perform other tasks well. Data quality, as shown in Figure 3, can be viewed from two angles: inherent or system-dependent [25]. As system-dependent quality attributes, such as availability and traceability [26, 27, 28, 29, 30], rely on the degree to which the machine stores the data and permits the data usage, we focus on the inherent data quality in our current work. The five characteristics listed below refer to the extent the data themselves are amenable to satisfying the DL functionality.

- **Accuracy** concerns how correctly the data represent the true value of the intended attribute of a concept or event in a specific context of use. For instance, if a city's temperature is recorded as "86 celsius", then semantic inaccuracy could be encountered and "86 fahrenheit" ("30 celsius") might be suspected to be the correct value.
- **Completeness** refers to whether the data associated with an entity have values for all expected attributes and in all instances. When constructing a supervised DL machine, missing labels represent a serious incompleteness challenge.
- **Consistency** is defined as attributes being free from contradiction and data being coherent with each other in a specific context. We discuss stationarity, a consistency property of time-series data, in our case study presented in the next section.
- **Credibility** is about whether the data attributes are regarded as true and believable by users. Credibility includes the concept of authenticity (the truthfulness of origins, attributions, commitments) [25].
- **Currentness** is concerned with the degree to which data have attributes that are of the right age. Using image recognition as an example, although landline phones are still around, using only them to train a DL algorithm without considering smartphone samples seems already outdated.

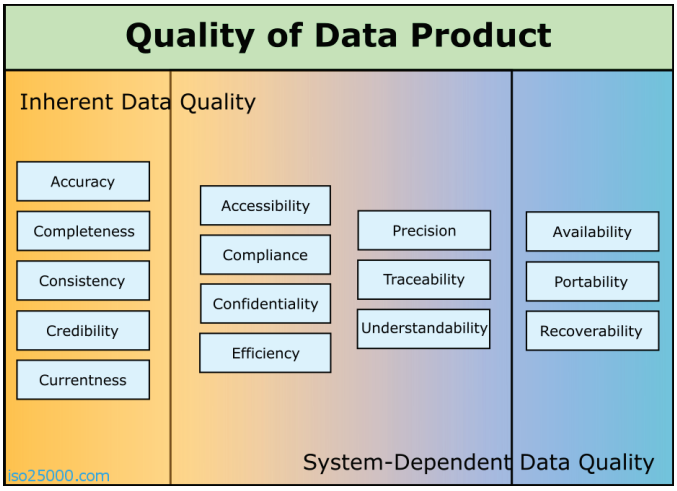


Fig. 3. Inherent data quality includes: accuracy, completeness, consistency, credibility, and currentness (*adopted from [25]*).

These quality characteristics of data have the intrinsic potential to satisfy a DL machine's requirements. The data artifacts are so different from the requirements artifacts. Not only are the desired qualities distinct (e.g., IEEE 830 for SRS [6] versus ISO/IEC 25012 for data [25]), but the approaches toward assuring qualities have to be diversified. In particular, static reading techniques like inspections, which are used to be successfully applied to examine requirements artifacts like an SRS, are no longer effective when data quality is at stake.

Due to the overlapping nature of data in Figure 2, we posit that data quality shall be assessed together with the DL

machine in an intertwined manner, rather than being checked on their own in isolation. Specifically, we propose to apply a new kind of metamorphic testing [31] by focusing on making data faulty along the inherent quality dimensions and then observing whether the DL machine behaves in some expected way.

Metamorphic testing emerged as a technique to alleviate the oracle problem, which refers to the lack of mechanism for checking whether the program under test produces the expected output when executed using a set of test cases [32]. In scientific software, for example, test oracle might be practically unavailable due to reasons like inherent simulation uncertainties and complex floating point operations [33]. Under these circumstances, metamorphic testing can be applied to continuous simulations to derive the desired outcome (oracle) of any simulation being a better fit than its previous round [34].

To illustrate our idea of using metamorphic testing, let us consider a DL machine whose classification accuracy is 90% on phone images. We could intentionally drop the training data's accuracy (one of the inherent data characteristics of Figure 3), e.g., by changing the label of some real phone images to "not phone". This is a concrete operation that we refer to as "metamorphic accuracy drop". Everything else being equal, we would expect the DL trained with the data after a "metamorphic accuracy drop" to behave worse than 90% in phone-image classifications. In this way, we are able to leverage the faulty data that are purposefully altered to compare the DL machine's behaviors before and after the change in data quality.

It is important to note that metamorphic testing machine learning is not new. Murphy *et al.* [35] made one of the first attempts to enumerate six operations a machine learning application's input data could be changed: additive, multiplicative, permutative, invertive, inclusive, and exclusive. Our work differs from these adding, shrinking, or rearranging operations in that we explicitly create *faulty* data based on the inherent quality characteristics, and embed these faulty changes in the DL machine to observe the effects. We next present a case study to offer insights into the faulty change made on the time-series data and the change effects on an RNN.

IV. Case Study

A. Study Context

We collaborate with MSDGC to address one of the most pressing societal problems: combined sewer overflow, or "overflow" for short. Figure 4 illustrates that combined sewer systems manage stormwater runoff, domestic sewage, and industrial wastewater in the same pipe, and when the volume of wastewater in the pipe exceeds the safe capacity (e.g., during heavy rainfall events), overflow occurs causing the untreated water to discharge directly to nearby water bodies. Because overflows can harm human and environmental health, they must be dealt with. For example, the first phase of the Overflow Long Term Control Plan at the Augusta Sanitary District of Maine involved a $12.2-million upgrade to better

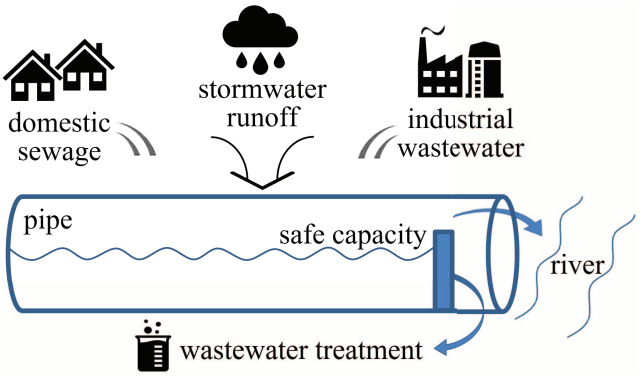


Fig. 4. Combined sewer overflows contain untreated or partially treated human and industrial waste, toxic materials, and debris as well as stormwater.

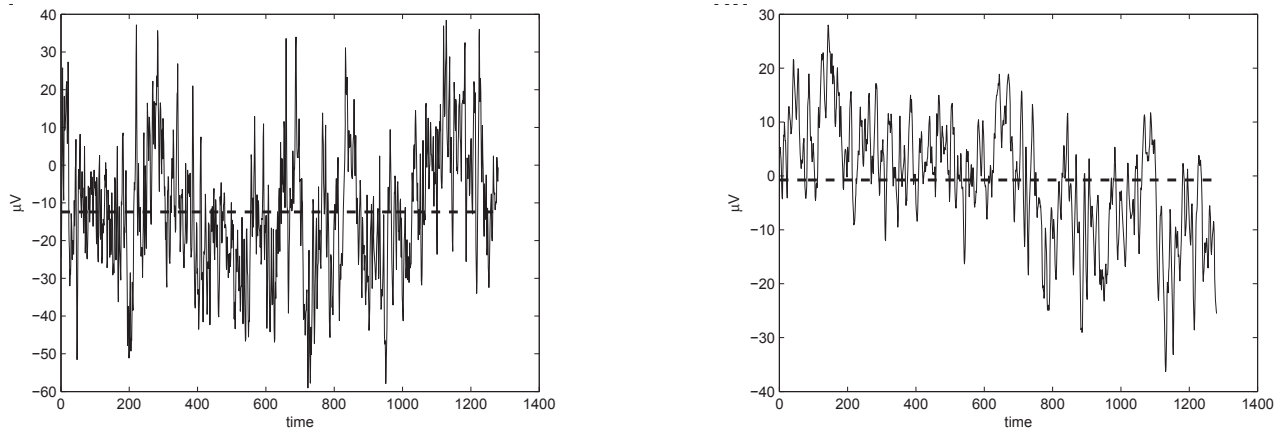
TABLE I
Schematic Overflow Data

Timestep	...	**Flow** (ft^3/sec)	**Velocity** (ft/sec)	**Level** (ft)	...
		...			
Day_1 13:35	...	0.07	0.70	545.21	...
Day_1 13:40	...	0.10	0.09	545.87	...
Day_1 13:45	...	0.09	1.08	546.79	...
		...			

treat excess wet weather flows, resulting in a 70% decrease of untreated overflows [36].

Some municipalities and utilities advance both the physical and cyber infrastructures to address the overflow challenges. MSDGC, for example, deploys the SCADA (supervisory control and data acquisition) system to collect large amounts of data, pioneering the development of a smart sewers network. A SCADA system's fundamental purpose is to communicate data and control commands from a centrally located operator to geographically dispersed remote locations in real time. In 2014, MSDGC began installing sensors throughout its largest watershed, and to date, MSDGC's smart sewer system covers over 150 square miles of its service area, incorporating two major treatment plants, six wet weather storage and treatment facilities, four major interceptor sewers, 164 overflow points, and 32 rain gauges and river level sites [36]. Remote monitoring has improved the maintenance of wet weather facilities and enabled upstream facilities to account for downstream interceptor conditions, increasing overflow capture basin-wide during wet weather.

With SCADA's vast collections, MSDGC explores DL's potential to automatically predict overflow-related data. Our study focuses on the data collected and archived for a specific overflow site in MSDGC's service area. To honor confidentiality, Table I depicts schematically an excerpt where timestep indicates the sequential nature of the time-series data. The middle columns of Table I represent various sensing data. For instance, an inflow manhole sensor can measure the runoff water velocity whereas a gate sensor can be used to measure the sewer level in the pipe.



(a) Stationary **(b) Non-stationary**

Fig. 5. Visualizing weakly stationary and weakly non-stationary time-series data (*adopted from [41]*).

RNN is widely recognized as one of the best DL models for time-series prediction in general [37], and for overflow forecast in particular [38]. RNN uses the internal state to process sequences of inputs, exhibiting temporal dynamic behavior. In [38], an RNN was trained to predict the stormwater runoff in terms of the precipitation and the previous runoff discharge. The experiment on the 34,721-timestep data collected at a combined sewer overflow site near the District of Columbia showed that the prediction accuracy was high when the hidden layer had 50 neurons, which was the maximum number allowed without the RNN running out of memory [38]. Informed by the relevant literature, we chose RNN to perform overflow-related prediction.

B. Stationarity of Multivariate Time-Series Data

The data shown in Table I consist of series of observations, $x_i(t)$; [i=1, 2, …, n; t=1, 2, …, m], made sequentially through time where i indexes the measurements at each timestep t [39]. When $n \geq 2$, there are multiple variables (e.g., flow, velocity, level, etc.) recorded at each observation, making Table I multivariate time-series data. For this kind of data, an important consistency characteristic is *stationarity*. Intuitively, a time series is stationary if the statistical properties of the time series, e.g., the mean and the correlation coefficients, do not change over time.

More formally, a time series is *strictly stationary* if the joint distribution of $X(t_1)$, $X(t_2)$, …, $X(t_n)$ is the same as the joint distribution of $X(t_1+\tau)$, $X(t_2+\tau)$, …, $X(t_n+\tau)$ for all t_1, t_2, …, t_n, τ, where $X(t)$ denotes the random variable at time t [40]. In other words, shifting the time origin by an amount of τ has no effect on the joint distributions, indicating that the statistical properties of the time series are invariant with respect to the window in which the data are analyzed. In practice, it is often useful to define stationarity in a less strict sense, and hence a time series is *weakly stationary* if both the variance and the mean are constant [40].

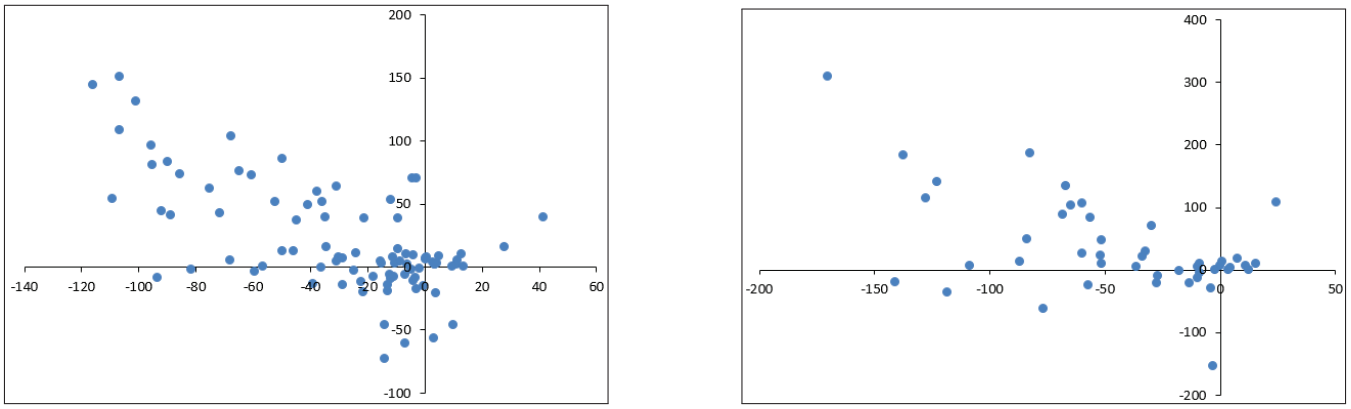
Figure 5 illustrates examples of weakly stationary and weakly non-stationary data where two different variables are plotted in the sub-graphs [41]. The dashed line in the middle represents the mean of each variable. Figure 5a shows that the values of stationary data move irregularly away from the mean but eventually revert to its mean, whereas Figure 5b displays that some trend of the non-stationary data exists. For a multivariate time series, co-integration can be performed to test stationarity [40]. High co-integration level means that one or more linear combinations of multivariate time series is stationary even though individually they may be non-stationary. If the time series are co-integrated, they cannot move too far away from each other [42].

Stationarity, therefore, is intrinsic to time-series data. We believe this attribute is key to consistency shown in Figure 3, as stationarity quantifies the degree to which the data are coherent and free from contradiction. We next describe our experiments to demonstrate how to exploit such an inherent data quality to test RNN in overflow-related forecast.

C. Results and Implications

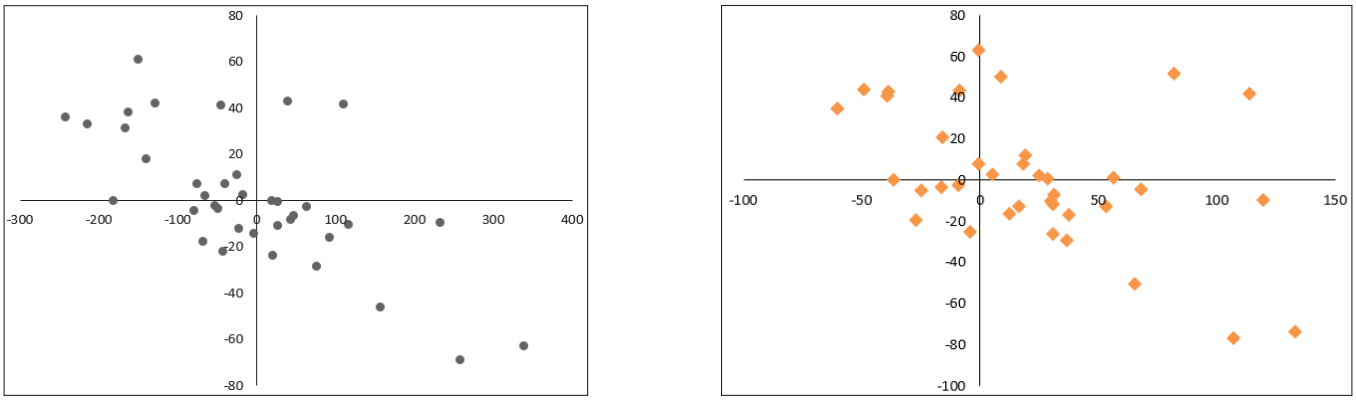
We constructed an RNN machine based on the long short-term memory implementation [43] with three 10-neuron hidden layers and one dense output layer. We used 'ReLU' activation function for each layer, optimized the model using the 'SGD' optimizer, set the learning rate to be 1e-2, and ran the model for 10 epochs with a batch size of 1 to predict one timestep further. For our experiment, 5,000-timestep sequential data were used for training the RNN. We tested the resulting RNN in two rounds: first with the test data of t timesteps, and then with these test data altered in stationarity.

To alter stationarity, we apply a linear transformation to the "flow" value at each timestep, i.e., we set $x_{flow}(t_m) = [\alpha \cdot x_{flow}(t_{m-1}) + \beta \cdot x_{flow}(t_m)] \,/\, \gamma$ while keeping the other two attributes' values intact ("velocity" and "level"). We call this operation "metamorphic stationarity change" as changing the variance of only one variable in a multivariate time series will impact the stationarity of the data. We then analyze how the change in stationarity measured by co-integration, which we calculate by using Python's statsmodels library [44], affects



(a) **(b)**

Fig. 6. Change in co-integration (x-axis) and the corresponding change in prediction error (y-axis) after the "metamorphic stationarity change" (α=2, β=1, γ=3) is applied to t timesteps of test data, which results in N tests (points in the plot): **(a)** t=300, N=90, and **(b)** t=600, N=45.



(a) **(b)**

Fig. 7. Change in co-integration (x-axis) and the corresponding change in prediction error (y-axis) after the "metamorphic stationarity change" (α=2, β=1, γ=3) is applied to 300 timesteps of test data, which results in 37 tests (points in the plot): **(a)** RNN trained with 4-month, more current data, and **(b)** RNN trained with 2-month, less current data.

the RNN's prediction at t. To quantify prediction performance, we aggregate the error: (value from the test data) − (value from RNN's prediction), for all three variables in our data.

Figure 6 shows the results. From these plots, a general trend can be inferred: When co-integration value increases meaning that the data become more stationary, the less prediction errors the RNN makes. This trend is in line with correlation-based multivariate time-series analysis where lesser stationarity leads to lower classification accuracy [41]. In addition, Figure 6 shows that stationarity decreases and prediction error increases scale in correspondence with t, e.g., the greatest prediction error jump in Figure 6a and Figure 6b is 150.17 and 309.08 respectively. To our surprise, such a scaling factor is not observed for stationarity improvement: the maximal co-integration in Figure 6a and Figure 6b is 41.61 and 24.59 respectively, indicating our current way of operating the "metamorphic stationarity change" tends to make the data consistency degree worse, rather than better.

Our results demonstrate the feasibility of intentionally manipulating the inherent data quality characteristics in engineering DL machines. In particular, by changing stationarity, we are able to empirically observe a desired RNN trend. Although preliminary, our findings show that requirements engineers shall not remain passive about data quality in DL. Our intertwined treatment of data quality requirements and DL machine construction has a couple of practical implications. First, RNN machine builders shall consider adding a data preprocessing component to stationarize the multivariate time-series items *before* feeding them into the RNN for processing. However, caution should be exercised as over-stationarizing (e.g., redundant first-order differencing) could hurt classification accuracy [41].

The second implications concerns RNN end users such as the MSDGC engineers. When training or retraining the RNN, the users shall select the data representing the problem context more stably. To investigate this, we perform another

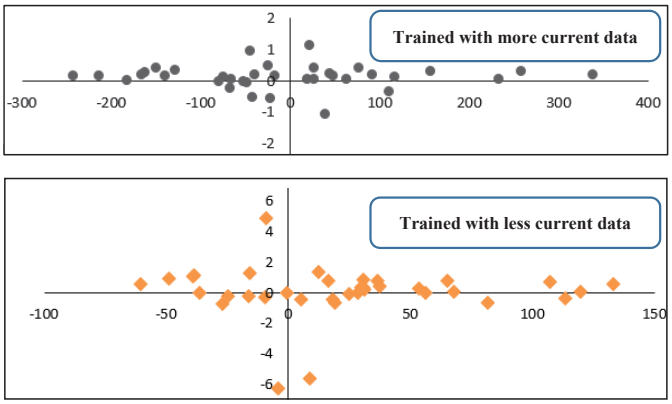


Fig. 8. Ratio of prediction-error change (y) over co-integration change (x) of Figure 7.

experiment where a collection of 300 timesteps of test data is used (e.g., May–June). We then train two RNN variants: RNN_{mc} with 4-month, more current data (e.g., Jan–April) and RNN_{lc} with 2-month, less current data (e.g., Jan–Feb). Figure 7 shows the results of testing the two RNNs when our "metamorphic stationarity change" operator is applied. While the same general trend of "less stationary, more error" holds, RNN_{lc} is more noisy. Figure 8 further compares the $\frac{y}{x}$ ratio of these two machines. A majority of the RNN_{mc} results are bound within a threshold range of $[-1, 1]$, whereas a wider range is exhibited by RNN_{lc}. In practical settings, such thresholds may be used to inform the MSDGC engineers about the extent to which the RNN machine behaves beyond the limits and hence human intervention would be required.

D. Threats to Validity

The construct validity of our case study analysis can be affected by the metrics that we used to measure stationarity and prediction error. Following Chatfield [40], we calculate co-integration values to assess the stationarity of MSDGC's multivariate time-series data. Similarly, to account for the inherent quality of the entire data rather than an individual attribute, we report the error by aggregating the difference between the RNN prediction and the test data of all the variables including the one to which "metamorphic stationarity change" is applied.

The internal validity can be influenced by the way that we implement the DL machine as well as the "metamorphic stationarity change" operation. The validity threats particularly concern our calibration of DL hyperparameters for the machine learning process, our choice of specific parameters used in the linear transformation, and our selection of training and test data in the experiments. In addition, though our intention is to account for sensor malfunctioning or other failure situations, the decision of linearly transforming only "flow" values may impose certain threats to internal validity.

The results may not generalize beyond the multivariate time-series data and the prediction context of combined sewer overflow, potentially hurting external validity. Although strong parallels can be drawn between our "metamorphic stationarity change" over the multivariate time-series data and approaches like DeepTest [45] that blur image data to detect erroneous behaviors of the DL machines, it is interesting future work to explore in which application areas data-driven metamorphic testing is suitable, and in what other domains it may not be desired.

V. Concluding Remarks

This paper presents our view that DL is a class of machines requiring special attention to the data quality requirements. Unlike stakeholder needs expressed in natural languages or requirements models, we argue that data for DL lie in the intersection of machine and environment. Thus, data are both indicative and optative [12]. The inherent quality characteristics of data are no longer amenable to be checked by using static reading techniques like inspections. Rather, we show how the wisdom of uncovering requirements faults can be transformed to a more integrated and dynamic mode. In particular, we define a novel set of operators based on metamorphic testing [31] to directly alter data quality, and illustrate our vision with RNN testing under "metamorphic stationarity change" of MSDGC's time-series data.

Our study explores systematic ways of assessing data quality in DL development. However, low-quality data can be caused by components such as data acquisition/sensing. This may require re-scoping of the environment and the machine so that stakeholder needs and desires could be better satisfied. Nevertheless, our work exploits faults in software and systems engineering to create new ways of taking advantage of these faults in RE tasks. For instance, by surveying the modeling mistakes in the literature [46, 47], we develop a novel mutation-driven approach to tracing safety requirement and state-based design models [48].

Our future work includes performing more experiments to lend strength to the preliminary findings reported here. We are also intrigued by the interdependencies and tradeoffs of the inherent data quality attributes. Indeed, our experimental results reported in Figures 7 and 8 has linked time-series stationarity with data currentness, suggesting future work may help assess multiple data quality requirements for DL. As the role of RE in AI reshapes and expands [49], the synergy of data qualities and nonfunctional requirements (e.g., robustness [50]) is worth exploring.

Acknowledgement

We thank the anonymous reviewers for the valuable and constructive feedback. We also thank Hemanth Gudaparthi for comments on earlier drafts of this paper. The work is funded in part by the U.S. National Science Foundation (Award CCF 1350487).

References

[1] V. R. Basili and B. T. Perricone, "Software errors and complexity: an empirical investigation," *Communications of the ACM*, vol. 27, no. 1, pp. 42–52, January 1984.

[2] D. E. Perry and C. S. Stieg, "Software faults in evolving a large, real-time system: a case study," in *European Software Engineering Conference (ESEC)*, Garmisch-Partenkirchen, Germany, September 1993, pp. 48–67.

[3] R. R. Lutz, "Analyzing software requirements errors in safety-critical, embedded systems," in *International Symposium on Requirements Engineering (RE)*, San Diego, CA, USA, January 1993, pp. 126–133.

[4] F. P. Brooks, *The Mythical Man-Month*. Addison-Wesley, 1975.

[5] B. W. Boehm, "Software engineering," *IEEE Transactions on Computers*, vol. 25, no. 12, pp. 1226–1241, December 1976.

[6] IEEE Standard Board, "IEEE recommended practice for software requirements specifications," https://standards.ieee.org/standard/830-1998.html, Last accessed: July 2020.

[7] G. M. Schneider, J. Martin, and W.-T. Tsai, "An experimental study of fault detection in user requirements documents," *ACM Transactions on Software Engineering and Methodology*, vol. 1, no. 2, pp. 188–204, April 1992.

[8] A. A. Porter and L. G. Votta, "Comparing detection methods for software requirements inspections: a replication using professional subjects," *Empirical Software Engineering*, vol. 3, no. 4, pp. 355–379, December 1998.

[9] B. Regnell, P. Runeson, and T. Thelin, "Are the perspectives really different? further experimentation on scenario-based reading of requirements," *Empirical Software Engineering*, vol. 5, no. 4, pp. 331–356, December 2000.

[10] J. C. Carver, N. Nagappan, and A. Page, "The impact of educational background on the effectiveness of requirements inspections: an empirical study," *IEEE Transactions on Software Engineering*, vol. 34, no. 6, pp. 800–812, November/December 2008.

[11] Ö. Albayrak and J. C. Carver, "Investigation of individual factors impacting the effectiveness of requirements inspections: a replicated experiment," *Empirical Software Engineering*, vol. 19, no. 1, pp. 241–266, February 2014.

[12] M. Jackson, "The meaning of requirements," *Annals of Software Engineering*, vol. 3, no. 1, pp. 5–21, January 1997.

[13] G. S. Walia and J. C. Carver, "A systematic literature review to identify and classify software requirement errors," *Information & Software Technology*, vol. 51, no. 7, pp. 1087–1109, July 2009.

[14] V. K. Anu, W. Hu, J. C. Carver, G. S. Walia, and G. Bradshaw, "Development of a human error taxonomy for software requirements: a systematic literature review," *Information & Software Technology*, vol. 103, pp. 112–124, November 2018.

[15] N. Niu, J. Savolainen, Z. Niu, M. Jin, and J.-R. C. Cheng, "A systems approach to product line requirements reuse," *IEEE Systems Journal*, vol. 8, no. 3, pp. 827–836, September 2014.

[16] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, "Forging high-quality user stories: towards a discipline for agile requirements," in *International Requirements Engineering Conference (RE)*, Ottawa, Canada, August 2015, pp. 126–135.

[17] N. Niu, S. Brinkkemper, X. Franch, J. Partanen, and J. Savolainen, "Requirements engineering and continuous deployment," *IEEE Software*, vol. 35, no. 2, pp. 86–90, March/April 2018.

[18] N. Niu, A. Y. Lopez, and J.-R. C. Cheng, "Using soft systems methodology to improve requirements practices: an exploratory case study," *Requirements Engineering*, vol. 5, no. 6, pp. 487–495, December 2011.

[19] S. Reddivari, Z. Chen, and N. Niu, "ReCVisu: a tool for clustering-based visual exploration of requirements," in *International Requirements Engineering Conference (RE)*, Chicago, IL, USA, September 2012, pp. 327–328.

[20] N. Niu, S. Reddivari, and Z. Chen, "Keeping requirements on track via visual analytics," in *International Requirements Engineering Conference (RE)*, Rio de Janeiro, Brazil, July 2013, pp. 205–214.

[21] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, "Improving agile requirements: the quality user story framework and tool," *Requirements Engineering*, vol. 21, no. 3, pp. 383–403, September 2016.

[22] V. K. Anu, G. S. Walia, G. L. Bradshaw, W. Hu, and J. C. Carver, "Usefulness of a human error identification tool for requirements inspection: an experience report," in *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, Essen, Germany, February-March 2017, pp. 370–377.

[23] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[24] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[25] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), "ISO/IEC 25012: Software engineering – software product quality requirements and evaluation (SQuaRE) – data quality model," https://www.iso.org/standard/35736.html, Last accessed: July 2020.

[26] A. Mahmoud and N. Niu, "Supporting requirements to code traceability through refactoring," *Requirements Engineering*, vol. 19, no. 3, pp. 309–329, September 2014.

[27] N. Niu, W. Wang, and A. Gupta, "Gray links in the use of requirements traceability," in *International Symposium on Foundations of Software Engineering (FSE)*, Seattle, WA, USA, November 2016, pp. 384–395.

[28] W. Wang, A. Gupta, N. Niu, L. D. Xu, J.-R. C. Cheng, and Z. Niu, "Automatically tracing dependability requirements via term-based relevance feedback," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 342–349, January 2018.

[29] N. Niu, W. Wang, A. Gupta, M. Assarandarban, L. D. Xu, J. Savolainen, and J.-R. C. Cheng, "Requirements socio-technical graphs for managing practitioners' traceability questions," *IEEE Transactions on Computational Social Systems*, vol. 5, no. 4, pp. 1152–1162, December 2018.

[30] W. Wang, N. Niu, M. Alenazi, J. Savolainen, Z. Niu, J.-R. C. Cheng, and L. D. Xu, "Complementarity in requirements tracing," *IEEE Transactions on Cybernetics*, vol. 50, no. 4, pp. 1395–1404, April 2020.

[31] S. Segura, G. Fraser, A. B. Sánchez, and A. R. Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, September 2016.

[32] X. Lin, M. Simon, and N. Niu, "Hierarchical metamorphic relations for testing scientific software," in *International Workshop on Software Engineering for Science (SE4Science)*, Gothenburg, Sweden, June 2018, pp. 1–8.

[33] Z. Peng, X. Lin, and N. Niu, "Unit tests of scientific software: a study on SWMM," in *International Conference on Computational Science (ICCS)*, Amsterdam, The Netherlands, June 2020, pp. 413–427.

[34] X. Lin, M. Simon, and N. Niu, "Exploratory metamorphic testing for scientific software," *Computing in Science and Engineering*, vol. 22, no. 2, pp. 78–87, March/April 2020.

[35] C. Murphy, G. E. Kaiser, L. Hu, and L. Wu, "Properties of machine learning applications for use in metamorphic testing," in *International Conference on Software Engineering and Knowledge Engineering (SEKE)*, San Francisco, CA, USA, July 2008, pp. 867–872.

[36] United States Environmental Protection Agency, "National Pollutant Discharge Elimination System (NPDES)," https://www.epa.gov/npdes/combined-sewer-overflows-csos, Last accessed: July 2020.

[37] J. Schmidhuber, "Deep learning in neural networks: an overview," *Neural Networks*, vol. 61, pp. 85–117, January 2015.

[38] N. Zhang, "Urban stormwater runoff prediction using recurrent neural networks," in *International Symposium on Neural Networks (ISNN)*, Guilin, China, May-June 2011, pp. 610–619.

[39] A. Tucker, S. Swift, and X. Liu, "Variable grouping in multivariate time series via correlation," *IEEE Transactions on Systems, Man, and Cybernetics (Part B)*, vol. 31, no. 2, pp. 235–245, April 2001.

[40] C. Chatfield, *The Analysis of Time Series: An Introduction*. Chapman and Hall/CRC, 2003.

[41] K. Yang and C. Shahabi, "On the stationarity of multivariate time series for correlation-based data analysis," in *International Conference on Data Mining (ICDM)*, Houston, TX, USA, November 2005, pp. 805–808.

[42] D. A. Dickey, D. W. Jansen, and D. L. Thornton, "A primer on cointegration with an application to money and income," *Economic Research*, vol. 73, no. 2, pp. 58–78, March/April 1991.

[43] F. Chollet, "Keras: The Python Deep Learning Library," https://keras.io/layers/recurrent/, last accessed: July 2020.

[44] S. Seabold and J. Perktold, "Statsmodels: econometric and statistical modeling with Python," in *Python in Science Conference (SciPy)*, Austin, TX, USA, June-July 2010, pp. 92–96.

[45] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: automated testing of deep-neural-network-driven autonomous cars," in *International Conference on Software Engineering (ICSE)*, Gothenburg, Sweden, May-June 2018, pp. 303–314.

[46] M. Alenazi, N. Niu, and J. Savolainen, "SysML modeling mistakes and their impacts on requirements," in *International Model-Driven Requirements Engineering Workshop (MoDRE)*, Jeju Island, South Korea, September 2019, pp. 14–23.

[47] W. Wang, N. Niu, M. Alenazi, and L. D. Xu, "In-place traceability for automated production systems: a survey of PLC and SysML tools," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3155–3162,

June 2019.
[48] M. Alenazi, N. Niu, and J. Savolainen, "A novel approach to tracing safety requirements and state-based design models," in *International Conference on Software Engineering (ICSE)*, Seoul, South Korea, May 2020, pp. 848–860.
[49] F. Dalpiaz and N. Niu, "Requirements engineering in the days of artificial intelligence," *IEEE Software*, vol. 37, no. 4, pp. 7–10, July/August 2020.
[50] H. Gudaparthi, R. Johnson, H. Challa, and N. Niu, "Deep learning for smart sewer systems: assessing nonfunctional requirements," in *International Conference on Software Engineering (ICSE): Software Engineering in Society (SEIS)*, Seoul, South Korea, May 2020, pp. 35–38.