COACH: COgnitive Analytics for CHange

Sinem Güven[1], Pawel Jasionowski[2], Karin Murthy[1], Krishna Tunga[3], George Stark[4]

[1]IBM T. J. Watson Research Center, Yorktown Heights, NY, USA
[2]IBM Global Services Delivery Center, Wroclaw, Poland
[3]IBM Systems Group, Hopewell Junction, NY, USA
[4]IBM Global Technology Services, Austin, TX, USA

{sguven, kmurthy, ktunga, gstark}@us.ibm.com, pawel.jasionowski@pl.ibm.com

***Abstract* — This paper presents our initial efforts towards building a cognitive analytics framework for change management. We propose a novel predictive algorithm for change risk calculation based on historical change failures, server failures, change triggered incidents as well as expert user input. Our predictive algorithm provides significant improvement over traditional risk assessments in proactively capturing problematic changes when tested with real client account data.**

Keywords—change risk; change management; proactive risk mitigation; change failure elimination; incident reduction

I. INTRODUCTION

Researchers have long investigated different risk prediction methodologies to reduce change failures [1, 2], and thereby eliminate service disruptions caused by changes [3, 4]. Assessing change risk accurately at change creation time is critical for the change approval process, as the risk level determines the level of scrutiny or automation the change would be subject to within the *Change Management* (CM) process. Change risk is typically represented as a numeric value within a range; for example, 1 representing the highest possible risk (Critical), and 5 representing the lowest possible risk (BAU). Although systematic change risk assessment (as opposed to assessing the risk through intuition) has taken its rightful place as a must in the CM process, the level of sophistication of practical assessments remains low, and most such assessments are still in the form of risk surveys. The reason for this is two-fold; first, Change Analysts may resists using a sophisticated risk assessment if it takes up more time than the traditional surveys, and second, they may feel the sophisticated risk assessment may have no added value. After all, both methods produce a risk number and risk mitigation actions, and it may take a while until time-consuming sophisticated risk assessment methods significantly reduce already small change failure rates to justify their usage.

In the IT service management community, there is often consensus around the importance of reducing failed changes to prevent client outages, however, as the number of failed changes are typically fairly small (~1%), it is often difficult to numerically justify focusing research efforts on change failure reduction [5]. Our recent work on exploring change incident relationships [6] revealed that it is often the seemingly successful changes that lead to incidents. Further, our analysis showed that up to 35% of client outages are caused by change (based on our analysis of 160 client account data over a period of 1 year). This explains the discrepancy between the numbers we see around failed changes, and change induced outages.

In our previous work [6, 7], we have established that, despite the unbalanced nature of the change data (~1% failed and 99% successful changes), it is possible to enable predictive model building for pro-active detection of failed and incident causing changes by merging them under the umbrella of *problematic changes*. In this paper, we build on top of and extend that work to propose a framework for cognitive analytics for change. We present the COACH framework, which is designed to provide capabilities to assess and accurately predict change risk (Figure 1). Our COACH Risk Calculator is designed with the Change Analysts' concerns in mind. It automates the change risk assessment to the extent possible to save time, provides a multi-faceted risk to provide insights into the calculated risk and also focuses on not only failed change prevention, but also change induced incident reduction.

The two main analytics components of COACH are *Change Incident Link Discovery* (right side of Figure 1) and *Change Risk Prediction* (left side of Figure 1). *Continuous Learning and Improvement* (bottom of Figure 1) ensures that the analytics learns from observations and improves over time. The next Section provides more details on COACH.

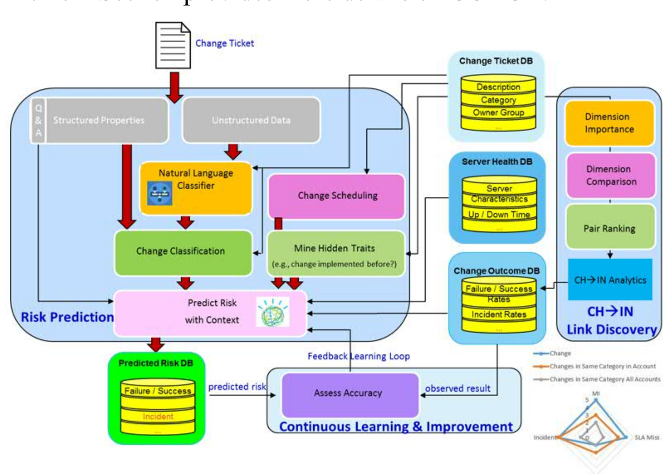

Figure 1. COACH Framework

II. Cognitive Analytics for Change

COACH's *Change Incident Link Discovery* module focuses on discovering relationships between changes and incidents based on a comprehensive analysis of historical change and incident data. Knowing which changes caused incidents in the past is crucial in establishing a data set that accurately reflects the historical risk of various types of changes. In our earlier work [7], we showed that such data is not readily available and that establishing accurate linkage between changes and the incidents that caused them requires sophisticated analytics. We demonstrated how natural language processing and machine learning can be used to extract clues from incident description and resolution text as well as categorical data to statistically link incidents to culprit changes. COACH uses the outcome of this analytics as an important input for change risk prediction (Figure 1).

The *Change Risk Prediction* module forms the basis of our cognitive change risk calculator that can be used at change preparation time to assess the risk of a change. The novelty of COACH lies in its ability to provide an all-encompassing risk context for change not only in terms of its likelihood of *failure*, but also its prospect of causing an *incident*. *Change Risk Prediction* leverages several cognitive analytics techniques to *understand* the change, *reason* about its risk, as well as employ *Continuous Learning and Improvement* to *learn* from past similar change outcomes over time. The rest of this paper will focus on the *Change Risk Prediction* aspect of COACH.

A. *Cognitive Change Risk Calculator*

This section illustrates how COACH provides the fundamental analytics to implement a cognitive change risk calculator that can accurately predict the risk of a change at change creation time.

Whenever a change to the IT infrastructure is required, a Change Analyst needs to assess the potential impact of the proposed change. Information available for assessment consists of, but is not limited to, the change description, details about the affected configuration items (such as a list of servers and their configuration), the team that will perform the change, and answers gathered to risk survey questions about the *probability* of change failure (e.g. how often was this change performed before) and the potential *impact* of change (e.g. how many business-critical processes and applications may be affected). COACH leverages all this information from the change record being created as input to its Change Risk Prediction module.

COACH also leverages additional information about the health of the involved configuration items, and the outcome of past changes performed on those specific configuration items. Moreover, the risk assessment may include how change scheduling affected the outcome of past changes and take into account various other traits.

In addition, COACH uses the Natural Language Classifier (NLC) [8] available on our BlueMix platform to automatically derive the broad category of the change (such as Hardware: Server or Software: Application Server). It then enhances this category further through information extraction from the text to arrive at a more fine-grained category (such as Hardware: Server: Reboot, or Software: Application Sever: Upgrade). Using this fine-grained change classification for the incoming change, COACH can determine similar historical changes and leverage their outcome to arrive at its risk prediction.

An important differentiator of COACH is its ability to provide an all-encompassing risk context in addition to the predicted risk score. This is depicted in the form of a spider chart (Figure 1 right bottom corner) that shows different risk dimensions, as well as a comparison of the proposed change's risk to similar changes across all client accounts in our data warehouse. This gives the Change Analyst additional insights regarding what the risk is related to (failure vs. incident, etc.), how the changes belonging to the same change category typically perform within the Change Analyst's account, as well as how the account's performance compares to other accounts for this type of change. The predicted risks are stored and periodically compared against the observed outcomes. While the predictive algorithm itself will adapt over time based on newly observed change outcomes, it is critical to learn additional more fine-grained context from recent inaccurate risk assessments. In order to do so, the predictive algorithm is complemented by looking at the predicted and observed outcome for similar changes in the same change category. If the algorithm predicts a low risk for a change, but a very similar change with a low predicted risk ended up having a problematic outcome, the assessed risk for the change is increased. More details of COACH's predictive algorithm are covered in Sections III and IV.

B. *Cognitive Underpinnings*

We now describe in more detail how COACH builds on the cognitive capabilities of *understand*, *learn*, and *reason*.

UNDERSTAND. In order to understand and judge the similarity of changes, COACH relies on a comprehensive change taxonomy that we derived based on 2.5 million historical change records for 160 clients.

We initially looked into simply reusing existing categories assigned by Change Analysts as part of the change record creation process. However, the granularity and type of categories varied considerably for different clients and different ticketing systems. While some clients assign detailed categories, such as "Application Shared Services (Database, BI, web and mobile development)", others only use generic categories such as "Application". Initial experiments also revealed that clustering historical changes by their description would not yield meaningful enough clusters. Due to the variety of ways the same change activity can be expressed by different people, historical changes considered to be similar by a subject matter expert did not necessarily end up in the same cluster. For example, changes related to patching the OS of a server can be expressed in many different ways: patch host, apply patch, install KBs, deploy hotfix, etc.

As a consequence, we had to build our own change taxonomy to arrive at meaningful groupings that were neither too broad nor too specific. Figure 2 depicts an excerpt from the three-level taxonomy COACH employs. Note that while we do not represent configuration-related information (such as the operating system of a server) in our taxonomy, this information is used by the predictive algorithm and inferred from the specific configuration items affected by the change.

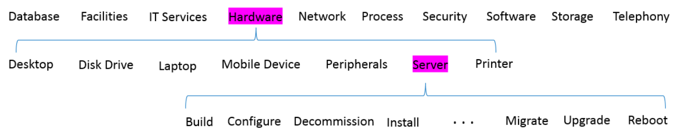

Figure 2. Change Taxonomy

We built rules to map historical changes to the second level of our taxonomy. The rules rely on (1) mapping existing categories where possible (2) exploiting change owner group information to narrow down to likely categories, and (3) extracting information from the change's short description that is specific to a change category. This allowed us to assign approximately 80% of all historical changes to a specific second-level group. We used natural language processing to extract activities from change descriptions in order to, wherever possible, further break-down the change category to the third taxonomy level.

LEARN. Using the second-level taxonomy labels assigned to each change using the rules described above, we trained a state-of-the-art natural language classifier [8] to predict the change category given a change short description. Using this classifier, COACH is able to assign the correct change category with a 96% accuracy. We did not include the third-level change activities into the classification process as many activities are similar across categories. For example, installing a driver is an activity that is common across all hardware sub-categories. As such, including activities in the classification effort makes it unnecessarily hard for the classifier to distinguish between categories. Instead, we separately extract activities from the abstract reusing the techniques built for historical data and use this information to assign a third-level category to each change.

We note here that the accuracy of the initial rule-based labeling was only spot checked and not all labels may be correctly assigned. Thus, despite a 96% accuracy on test data, the real accuracy may be less. Fortunately, the change risk calculator gives users the ability to overwrite the automatically predicted change class. Over time, this provides COACH with additional training data that allows it to retrain and improve the NLC classification accuracy.

COACH also continuously learns new change failure rates, change incident rates, server failure rates by refreshing these statistics through automated analysis of new incoming data to our data warehouse. Change incident rate refresh requires continuously discovering new pairs of changes that led to incidents, from incoming change and incident data. Similarly, server failure rate refresh requires that the underlying problematic server classifier [9] runs continuously as new hardware problems are identified.

COACH initially presents the users with all risk assessment questions, but learns over time which risk assessment questions to eliminate due to low correlation to outcome, or if a question always tends to yield the same answer.

Finally, the predictive algorithm, described in Sections III and IV, continuously learns as new changes are performed and new data becomes available.

REASON. Section III and IV describe in detail how COACH reasons about the risk of a change building on the understanding and learning described in this section.

III. RISK PREDICTION ALGORITHM

This Section provides a description of how we determined the various inputs to the COACH Risk Calculator, and also presents details of its underlying risk prediction algorithm. Unlike traditional change risk calculators, our risk prediction is *cognitive (see Section II)*: it *understands* the change, *learns* from a dynamic data set and outcomes, and *reasons* with a data-driven statistical approach.

A. Predictive Algorithm Design Considerations.

During the predictive algorithm design process, we tried various machine learning models, however, we decided to go with a statistical algorithm instead due to the following restrictions:

a) *Unbalanced Data:* The number of problematic changes (even with the addition of changes that cause incidents) remain fairly low compared to the successful change volume (exact numbers cannot be disclosed due to confidentiality). The heavily unbalanced nature of the data makes it unideal for machine learning models. Although there is a lot of research on building predictive models on unbalanced data [10, 11], proposed techniques mainly work on static models built out of training data sets and updated manually. Because COACH needs an underlying predictive algorithm that learns over time as new change, incident and server data become available, we were not able to leverage these techniques.

b) *Access Restrictions:* We had certain data access restrictions across different geographies, which meant that we may not have access to historical change descriptions at risk calculation time to compare them to our incoming change description. As COACH needs to be available to any client account, the data access restriction further limited the selection of machine learning algorithms to a small set. With the change features available to us at change creation time, it was not possible to create a meaningful predictive model through machine learning algorithms.

B. Determining Potential Predictors

Next, we defined a set of potential predictors, based on the different pieces of information available to us at change creation time. We used IBM SPSS Statistics software to perform Chi-square tests on the potential predictors to see if the perceived relationship between the predictors and the change closure code (*successful* or *problematic*) is due to chance, or if there is a true association between the predictors and the outcomes. For example, when we tested the effect of survey risk on change outcome, the p-value = .000 indicated that there's a 0% chance to find the observed (or a larger) degree of association between the variables if they're perfectly independent in the population. In other words, survey risk significantly impacts the change outcome, and thus, is an important predictor for change risk. Similarly, we tested several other predictors using the same method. In the next few sub-sections, we only report on those predictors which were observed to have a strong association (p = .000.) to the outcome (change closure code).

C. *Predictor: Risk Assessment Survey*

A risk assessment survey typically comprises Probability and Impact questions. For example: *How many times have you implemented this type of change? How complex is the implementation? What is the potential business impact in case of failure?* Each question is rated, for example, from 1 (the highest) to 5 (the lowest) by the Change Analyst, and the probability and impact values are aggregated through a formula to calculate the final risk through a look-up of a risk matrix, such as the one in Table 1.

Probability \ Impact	1	2	3	4
1	1	1	2	3
2	1	2	3	4
3	2	3	4	4

Table 1. Change Risk Matrix

Let us assume that the risk is initially calculated with the traditional survey method and is denoted as *survey_risk*. Before designing a predictive algorithm, we first wanted to test the effectiveness of the traditional survey method by comparing the survey_risk with the change outcome. To perform this test, we built a dataset based on one client's data (26,600 change records) from our data warehouse that comprised *successful* changes (with no incident), and *problematic* changes {changes that failed, were backed out, had issues, or caused an incident). We decided to focus on one client with the most number of problematic changes to have enough representation of problem cases. Also, because the definition of risk is not consistent across all client accounts (risk level 1 can mean the lowest for one account and highest for another), we did not want to impair the analysis by comparing mapped risk ratings.

Once the dataset is built, we inspected the changes and compared their survey_risk ratings to their observed outcomes to calculate:

a) successful recall (i.e., how many of the successful test changes were correctly classified as 'low risk or medium risk' by the survey_risk)

b) problematic recall (i.e., how many of the problematic test changes were correctly classified as 'high risk' by the survey_risk)

Table 2 provides a summary of survey_risk effectiveness. When we look at the recall rates, we see the survey does a decent job of calculating the risk of successful changes (48% low risk, 52% medium risk and no high risk). However, the survey falls short when capturing problematic changes (88% of problematic changes have medium risk, and none has high risk). In general, the survey seems to assign medium risk irrespective of whether the change is actually problematic or successful. The high number of successful changes classified correctly (100%) by survey_risk confirms the association Chi-square test indicated. However, since our focus is to identify problematic changes (while not over-predicting risk), we confirmed the need for a more sophisticated risk calculation approach.

Table 2. Survey Risk Effectiveness

Survey Risk	Successful Recall	Problematic Recall
Low	48%	12%
Med	52%	88%
High	0%	0%

D. *Predictor: Change Failure Rate*

In order to calculate change failure rates, we rely on at least one-years' worth of historical data from our data warehouse, which comprises 160 client accounts' operational IT data. By considering a fine-grained change category (see Section II B) and change closure code (*successful*, *failed*, *issues*, *backed-out*), we can compute the percentage of failed changes in each category. Our definition of a "failed change" comprises any change with {*failed*, *issues*, and *backed-out}* closure codes.

E. *Predictor: Change Incident Rate*

If change closure codes readily indicated whether a change led to an incident, we could have used the same logic as in Section III D to calculate the percentage of changes that cause incidents. Unfortunately, this is usually not the case as it may not be apparent at change closure time whether the change caused an incident, and if it becomes apparent a while later through root cause analysis, it is not practical for Change Implementers to go back and update closed change records due to time constraints. Instead, we rely on COACH's Change–Incident Link Discovery analytics [7] to mine such change – incident relationships. Once we have the change – incident pairs, we compute what percentage of changes cause incidents in each change category. Our definition of "changes that cause incidents" comprises any change that led to incident(s) and has one of {*successful*, *failed*, *issues*, *backed-out*} closure codes.

F. *Predictor: Server Failure Rate and Server Properties*

Server failure rates are important to consider when predicting change risk as they indicate how likely it is that the change running on a server may fail due to the server's current health issues. Server failure rates are determined by statistical modeling and simulation techniques used to study server incidents, utilization, and configurations to classify them into two groups: problematic and non-problematic. Moreover, the simulation component of these capabilities propose improvement actions for problematic servers along with quantifiable benefits for each (what-if scenarios) [9].

Server properties, such as operating system, manufacturer, environment (production vs. development) are also available to us through our client account data warehouse, and will be used to model similar server behavior when we do not have explicit knowledge about the server on which a change would be implemented.

With our predictors identified and confirmed, we can now design a predictive algorithm to calculate change risk based on survey risk, change failure rate, change incident rate and predicted server failure rate.

G. Formalizing Change Risk Prediction Algorithm

Our risk prediction approach takes survey_risk as its basis, and provides an *adjustment function* to increase or lower the risk as determined by COACH. The reason behind this design decision was to leverage the effectiveness of the survey_risk on successful changes to avoid over predicting the actual risk. The rest of this Section formally expresses our algorithm.

i) Change Failure Rate: Based on historical change data, we can define all possible change categories (Section II B).

Let $c_j, j = 1, 2,..., n$ be all available change categories.

We define the number $Pfc(c_j)$ as a probability of failed change inside the category c_j, i.e. the number of all failed changes in the category divided by the number of all changes in the category.

Let $m_c = 1/n\ \Sigma\ Pfc\ (c_j)$ be an average of $Pfc(c_j)$.

Moreover, we define $s_c{}^2 = 1/n\ \Sigma\ (Pfc(c_j) - m)^2$ where s_c is the standard deviation of $Pfc(c_j)$.

Now we define the function $cfr(x_c)$. The value x_c represents how problematic the change is. The number $x_c \in [0, 1]$ is defined based on the category of the performed change. We compute x_c as a probability of change failure in at least one category of c_j if and only if the user defines that the change belongs to category c_j. Then:

$$cfr(x_c) = 0,\ x_c \in [0, m_c]$$

$$cfr(x_c) = 1 - exp[-(x_c - m_c)^2/(2s_c{}^2)],\ x_c \in (m_c, 1]$$

This means that $\mathrm{cfr}(x_c)$ will only increase risk if the probability of failure for the selected change category is higher than the average failure rate across all change categories. The advantage of using a function of change failure probability instead of simply using the change probability is that we do not unnecessarily increase change risk when the probability of failure is low (less than average).

ii) Change Incident Rate: The Change incident rate function is defined in exactly the same way as the change failure rate function (see above).

Let $cir(x_i)$ be the change incident function.

Again, the value x_i represents how problematic the change is based on the category of the performed change. We compute x_i as a probability of change causing incident in at least one category of c_j if and only if the user defines that the change belongs to category c_j.

iii) Server Failure Parameter: Next, we define the problematic and non-problematic server parameter. Let h be the name of the server specified by the Change Analyst (if more than one server is specified, we perform the described action for all servers and choose the maximum result parameter). Let x_h denote the problematic score for the server h. m_s *and* s_s denote the mean and standard deviation of the problematic score for all the servers in our data warehouse combined. We define the function $hfr(h, x_h)$ as follows:

a) when the server h is identified as non-problematic in our data warehouse, then $hfr(h, x_h) = 0$;

b) when the server h is identified as problematic in our data warehouse, then:

$$hfr(h, x_h) = 0,\ x_h \in [0, m_s + s_s]$$

$$hfr(h, x_h) = 0.5 + p,\ x_h \in (m_s + s_s, 1]$$

where $p = 1 - cfr(m_c + 3s_c)$.

If the server's information is not available in our data warehouse, then we compute the $hfr(h, x_h)$ value based on the percent of problematic servers with the same attributes, such as OS {provider, name, version}, purpose and environment.

Now, we can define the risk adjustment function:

$$ra(h, x_c, x_i, x_h) = cfr(x_c) + cir(x_i) + hfr(h, x_h)$$

Based on the change category distribution, we observe less changes in higher risk categories. This is intuitively correct from the change management process point of view as most changes are of medium (risk = 3) difficulty. To accommodate this observation, we use the 3-sigma concept to adjust the risk:

Let m and s represent the mean and standard deviation of the adjusted risk score, $ra(h, x_c, x_i, x_h)$ computed above for all the changes in our data warehouse and let t_s be defined as:

if survey_risk > 3 then $t_s = 1$

if survey_risk = 3 then $t_s = (m + s)$

if survey_risk = 2 then $t_s = (m + 2s)$

if survey_risk = 1 then $t_s = 0$

Finally, the change risk can be defined as follows:

$$risk = survey\_risk + t_s \cdot ra(h, x_c, x_i, x_h)$$

The resulting risk rating is determined by taking the risk to be the nearest positive integer of the risk parameter. Finally, as our algorithm is made up of distinct risk functions, COACH can provide specific insights about the change risk to Change Analysts by indicating if they should expect change failure vs. incident vs. potential server failure as part of the change they are implementing.

IV. PREDICTION ALGORITHM VALIDATION

In order to calculate recall precision for our predictive algorithm, we used the same dataset we built for testing survey risk effectiveness, and compared the changes' predicted outcomes to their observed outcomes. As seen in Table 3, using our predictive algorithm, there is significant improvement in our ability to proactively detect problematic changes (0% → 78%), at the expense of slightly increased false positives (0% → 38% high risk for successful changes). We argue that successful changes end up being successful due to correct actions having been taken at change preparation time. This means that 38% of the successful changes can be inherently risky, but ended up being successful due to the risk being mitigated, and thus our prediction has the potential of being even more accurate than results depicted in Table 3.

Table 3. Comparison of Survey Risk (top) and Predicted Risk (bottom) Performance

Survey Risk	Successful Recall	Problematic Recall
Low	48%	12%
Med	52%	88%
High	0%	0%

Predicted Risk	Successful Recall	Problematic Recall
Low	30%	8%
Med	32%	14%
High	38%	78%

As our change categories improve over time through cognitive learning, we will have more granularity around how change risk is calculated. At the same time, as more changes with known outcomes accumulate in our data warehouse, our change failure rates, change incident rates, and server failure rates are automatically re-learned, keeping the underlying predictive algorithm data up to date. Finally, comparison of the predicted risk with recently observed outcomes of similar changes will ensure that change risk is consistent with the expected outcome. Such features of the COACH framework will, in turn, enable more accurate risk prediction.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we presented the COACH framework and focused on its change risk prediction capability. We described our predictive algorithm design considerations and restrictions, as well as the data that was available to us at predictive algorithm design phase. We also explained how our predictors were selected and demonstrated that our predictive algorithm is significantly more accurate than traditional survey based risk assessments in proactively identifying problematic changes.

Our work only scratched the surface of the effort required to build a cognitive framework for change management. There are several areas where we need to improve our system. First, we have access to root cause analysis (RCA) and service level agreement (SLA) data in our warehouse. As the next step, we will integrate SLA miss and Major Incident (MI) likelihood parameters to our predictive algorithm. As SLA and MI data points are rare (just like failed changes), our current predictive algorithm design provides the right base for integration of such predictors. Further, we would like to explore our RCA data to be able to provide "reasons" for problematic changes in addition to highlighting their risk. This would enable Change Analysts to take appropriate actions to mitigate the indicated risk. Our journey in improved change risk classification will certainly continue as more data and more client accounts become available in our data warehouse. Our goal is to get to the right level of granularity for change categories to be able to easily compare changes across all accounts without losing the categories' inherent risk. Finally, we will test the effectiveness of our Continuous Learning and Improvement loop as the COACH Risk Calculator continues to accumulate more and more assessed changes.

REFERENCES

[1] Bianchin et al., "Similarity Metric for Risk Assessment in IT Change Plans," In IEEE CNSM, Niagra Falls, ON, Oct. 2010.

[2] S. Güven, C. Barbu, D. Husemann, D. Wiesmann, "Change Risk Expert," In IFIP/IEEE NOMS, Maui, Hawaii, 2012.

[3] Wickboldt et al., "Improving IT Change Mgmt Processes with Automated Risk Assessment," In IEEE DSOM, Oct. 2009.S.

[4] Hagen, M. Seibold, A. Kemper, "Efficient Verification of IT Change Ops.," In IFIP/IEEE NOMS, Maui, Hawaii, 2012.

[5] J. Druebert. "Changes, Incidents & Unintended Consequences," In Insight on IT Service Management, 2010.

[6] S. Güven and K. Murthy, "Understanding the Role of Change in Incident Prevention" In IEEE/IFIP CNSM, Montreal, 2016.

[7] S. Güven et al., "Towards establishing causality between Change and Incident", In IEEE/IFIP NOMS, Turkey, 2016.

[8] https://console.ng.bluemix.net/catalog/services/natural-language-classifier

[9] J. Bogojeska, D. Lanyi, I. Giurgiu, G. Stark, D. Wiesmann. "Classifying Server Behavior and Predicting Impact of Modernization Actions". In IFIP/IEEE CNSM, Zurich, Switzerland, 2013.

[10] A.-D. Lipitakis and S. Kotsiantis, "A hybrid Machine Learning methodology for imbalanced datasets," in IEEE IISA, Crete, Greece, 2014.

[11] F. Bulut, "Performance evaluations of supervised learners on imbalanced datasets," in Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT), 2016.