Does the past say it all?
Using history to predict change sets in a CMDB

Sarah Nadi, Ric Holt
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
snadi, holt@uwaterloo.ca

Serge Mankovskii
CA Labs Canada
serge.mankovskii@ca.com

***Abstract*—To avoid unnecessary maintenance costs in large IT systems resulting from poorly planned changes, it is essential to manage and control changes to the system and to verify that all items impacted by each change are updated as needed. This paper presents a method of decision support that helps guarantee that each change set (those items to be updated in the change) contains all the software or hardware components impacted by the proposed change. Today, many IT systems are managed by a Configuration Management Database (CMDB), which can be represented as a large graph in which the nodes are configuration items (CIs), such as software applications or servers, and the edges record dependencies between these items.**

In this paper we present a new approach to suggesting change sets based on our conjecture that each new change set is likely to be similar to instances of previous change sets. Accordingly, if the analyst determines that CI x is in a new change set, our method essentially searches for previous change sets, stored in the CMDB, that contain x, and suggests that CIs in those sets (appropriately ranked) should be considered for inclusion in the new change set.

Our model uses *support* and *confidence* measures to estimate how closely nodes x and y are related, based on how often they have appeared together in past change sets. Based on these measures, we implement a prototype that suggests likely items to an analyst who is composing a change set. Based on a history of three years of a particular industrial CMDB, and several filtering techniques, the observed recall and precision values were as high as 69.8% and 88.5% respectively.

***Keywords*-Data mining, change management, maintenance, configuration management.**

I. Introduction

Understanding the effect of a change in an Information Technology (IT) system before applying it is important. For example, if a security patch is to be installed on a server, analysts should ensure that none of the applications hosted by this server will be negatively impacted. In this case, the set of entities to change, i.e. the change set, will include not only the server, but all of the affected applications as well. In order to successfully implement a change without causing costly disruptions to the system, the correct change set should be identified before the change is implemented. Correctly planning changes in an IT system results in a more maintainable system with fewer undetected errors. To assist analysts in doing this, we use historic information to predict change sets.

There has been much work done on predicting the propagation of a change on the source code level through mining software repositories such as CVS [1], [2], [3], [4]. Several studies from the field of Mining Software Repositories have shown that code change history is a good predictor of future changes in the code [5], [6].

In this paper, we are concerned with software (and hardware) that has already been deployed in a complete IT system. In this situation, the software is changed by reconfiguring it, parameterizing it, changing the components it relies on, etc. Such changes are tracked using a Configuration Management Database (CMDB) which logs incidents and change orders as well as tracks configuration items (CIs) and the relationships between them [7]. An analyst usually creates a Change Order to record each change to the system and to log the changed CIs. In this work, we explore whether history is a good predictor of future changes on a system level. That is, if the analyst indicates that a particular CI is going to be changed, can we use historical records to successfully suggest other CIs that should be changed as well?

The Runtime Automated Configuration Engineering (RACE) project is a joint collaboration between the University of Waterloo and CA Labs Canada which aims to provide IT analysts with better decision support tools. In our previous work [8], we have presented DRACA as a Decision support framework for Root cause Analysis and Change impact Analysis. In that work, we presented techniques for root cause analysis. In this work, we present decision support for change impact analysis. Change impact analysis involves identifying entities (in this case CIs) in a system that are likely to be impacted by a proposed change.

Given an initial CI to change, analysts commonly follow the relationship edges stored in the CMDB to determine a list of other CIs that may be impacted. We have been in contact with CA's Global Information Systems (GIS) team that is responsible for managing all of CA's IT system.

We interviewed members of the GIS team, as well as CA customers, and most of them agreed that they need more details about the impact of the change on each CI in the produced list. To do this, DRACA acts as an oracle which when provided a CI that is to be changed, proceeds to suggest other CIs to change as well. DRACA ranks the list of suggested CIs to provide the analyst an indication of whether she should include a suggested CI in her change set. We tested our approach on data provided by CA, and obtained promising results in terms of recall and precision through applying various filtering techniques.

The rest of this paper is organized as follows: Section II gives an overview of a CMDB which is the main repository used in this work. Section III explains the change impact analysis process provided by DRACA. Section IV explains how DRACA's performance will be evaluated. Section V explains the underlying model DRACA uses for suggesting CIs. Section VI presents the empirical work performed to evaluate DRACA. This section explains the structure of the data used, the experiment setup, as well as the results obtained from the base case of our experiment and those from the various filtering techniques we used. Section VII presents related work in the field, and Section VIII discusses possible future directions for this work. Section IX concludes this paper.

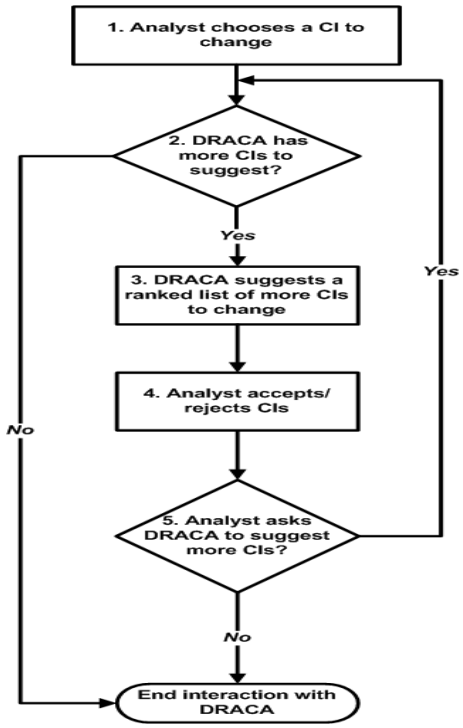

Figure 1. Flowchart of DRACA's change impact analysis process

II. Configuration Management Database (CMDB)

We focus on systems that generally follow ITIL [7] which is a set of standards concerned with providing best practices for business effectiveness and efficiency in the use of information systems. ITIL defines configuration management as "the process of identifying and defining configuration items in a system, recording and reporting the status of configuration items and requests for change, and verifying the completeness and correctness of configuration items".

A configuration item (CI) is a component of an IT infrastructure, such as a server or a software application, which is put under the control of the configuration management process. CIs and the relationships between them are stored in the Configuration Management Database (CMDB). The goal of configuration management, apart from accounting for all IT assets and their configuration, is to provide decision support for change management, incident management, problem management, and release management [7]. Accordingly, apart from containing information about CI configuration in the system and their interrelations, the CMDB also keeps track of CI history. This is through storing past incident reports, problem reports, and change orders of CIs. When an analyst views the details of any CI, she can see all the incident, problem, and change reports this CI was involved in.

Change management in ITIL is defined as "the process of controlling changes to the infrastructure or any aspect of services, in a controlled manner, enabling approved changes with minimal disruption". ITIL identifies the need to identify other CIs that will be impacted whenever a change to a specific CI is proposed. In this work, we aim to support analysts as they perform change management.

III. DRACA's Change Impact Analysis Process

The change impact analysis process that we propose is a collaborative interaction between the analyst and the DRACA tool as illustrated in Figure 1. Figure 2 shows the Graphical User Interface (GUI) for our prototype supporting this process.

This process consists of five main steps as shown in both figures. In step 1, the analyst provides DRACA with the CI she wants to change (the initial CI). In our prototype, she enters the name into the 'CI Name' text box. In step 2, the analyst clicks the 'Suggest CIs' button which triggers DRACA to check its model to search for CIs that have previously changed with the initial CI.

If DRACA has no suggestions to make to the analyst (e.g. the indicated CI has not previously changed with any other CIs in DRACA's stored model), then this ends the process of interaction between DRACA and the analyst. However, if DRACA has suggestions to make, it moves to step 3 and provides the analyst with a list of ranked CIs. These CIs are displayed in the 'Suggested CIs (Ranked)' table, and are color coded according to their ranking (red, blue, and green).

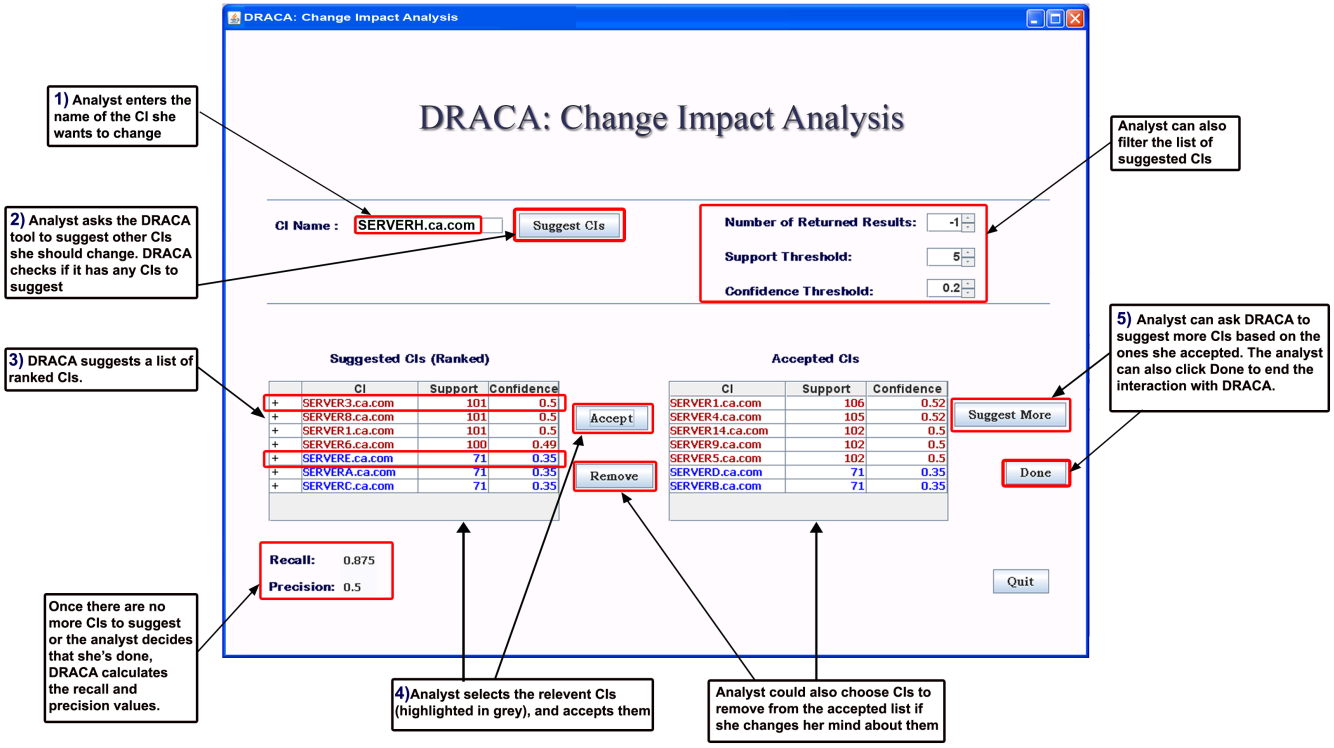

Figure 2. The DRACA prototype tool suggests a list of ranked CIs to the analyst, and allows her to ask for more suggestions based on the CIs she accepts

The analyst, in step 4, then chooses the CIs to include in her change set. The information provided by DRACA in terms of support and confidence guides the analyst to choose the CIs to accept. The accepted CIs (along with the initial CI the analyst provided) form the change set that is being constructed. To accept any of the suggested CIs in Figure 2, the analyst highlights the CI(s) she wants to accept, and clicks on 'Accept'. This moves the selected CIs to the 'Accepted CIs' table.

At this point, in step 5, the analyst may ask DRACA to make more suggestions based on the new CIs she added to the change set through the 'Suggest More' button, or she may choose to end the interaction with DRACA through the 'Done' button. If she asks DRACA for more suggestions, then for each newly accepted CI, DRACA searches for other CIs that have changed with it and suggests them to the analyst. DRACA will not, however, suggest any CIs that have been suggested before. This process continues until either DRACA has no more suggestions to make or the analyst decides not to ask DRACA for more suggestions, and clicks 'Done'. Once this process is over, the analyst's interaction with DRACA ends.

If there are CIs that the analyst wants to add to the change set, and which DRACA has missed, the analyst is free to add these CIs to the change set. If this situation occurs, then DRACA was not able to suggest all CIs relevant to the analyst. However, if the analyst does not add any extra CIs after its interaction with DRACA is over, then DRACA found all the relevant CIs. After the analyst chooses the change set, she finalizes the change order which will be stored in the CMDB repository.

IV. Measuring the Performance of DRACA

We need a way to evaluate DRACA's suggested CIs. Ideally, DRACA would suggest all the CIs in the change set without making any errors, but of course this does not often happen in practice. The recall and precision measures from the information retrieval field are appropriate for this type of evaluation. Recall measures the proportion of correct CIs retrieved by the system, while precision measures the proportion of suggested CIs that are correct [9].

We define the *Predicted Set* (P) as the set of all CIs DRACA suggests through the full the iteration process (see Figure 1). We define the *Occurred Set* (O) as the CIs remaining in the change set after excluding the Initial CI provided by the analyst (i.e Change Set - Initial CI). The intersection of the predicted set and the occurred set, called PO, is the common CIs in both sets. For each constructed change set, we then calculate the recall and precision values for the predictions according to the following definitions [1]:

$$Recall = \frac{|PO|}{|O|} \quad (1)$$

$$Precision = \frac{|PO|}{|P|} \tag{2}$$

If no CIs are predicted (i.e., P and thus PO are empty), precision is defined as 1 since there cannot exist any incorrect predictions in an empty set. On the other hand, if the size of the change set is 1, and thus the size of the occurred set is 0, recall is defined as 1 since there are no CIs to predict [1].

In order to have a single measure that indicates the effectiveness of our predictions, we use the F-measure which is based on van Rijsbergen's effectiveness measure which combines recall and precision [9]. The F-measure is calculated according to Equation 3 which gives equal weighting to recall and precision. The ideal F-measure is 1 where both recall and precision are 1.

$$F = 2 * \frac{precision * recall}{precision + recall} \tag{3}$$

V. Model Used

In order to keep track of which CIs changed together in the past, we need a model that gives an indication of which CIs change together frequently versus those which do not. To accomplish this, we use two measures, *support* and *confidence*, from Zimmermann et al. [3].

Given a historical sequence of change sets, we start by counting the number of times each pair of entities (CIs in our case) appeared in a change order. For example, Figure 3 records that CIs B and C have occurred together in 4 change sets. We call this count the *support* between these two CIs. The support is drawn as an undirected edge between the two CIs. These counts are stored in the *support matrix S* as shown below. This example support matrix below corresponds to Figure 3.

$$S = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 5 & 5 & 0 \\ 5 & 10 & 4 \\ 0 & 4 & 8 \end{pmatrix} \end{array}$$

An entry S_{ij} shows how many times i and j changed together during the time period we are mining. For example, since A and B changed 5 times together, entry $S_{AB} = 5$. Noste that S_{ii} is the number of times i changed with itself which is simply the count of how many times i changed in total. Note that the support matrix is symmetric since the number of times A and B changed together is the same as the number of times B and A changed together.

The support matrix S records the actual counts of co-changes. It is also useful to compute relative counts that take the total number of times each of these CIs have changed into consideration. For example, in Figure 3, A and B have changed 5 times together while the total number of times A and B have each changed are 5 and 10 respectively. This means that every time A changed, B changed with it as well while every time B changed, A changed with it only half the time. We call this relative count the *confidence* and calculate it as follows [3]:

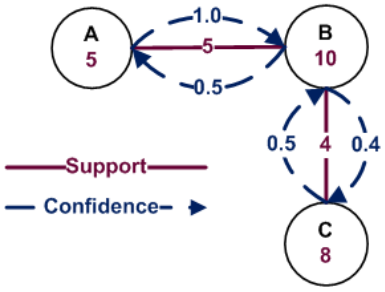

Figure 3. Tracking changes between CIs

$$C_{ij} = \frac{S_{ij}}{S_{ii}} \tag{4}$$

For example, C_{BA} is equal to $\frac{S_{BA}}{S_{BB}} = \frac{5}{10} = 0.5$, which means that 50% of the time that B appeared in a change set, A also appeared in the same change set. The *confidence matrix C* for the example in Figure 3 is given below. Unlike the support matrix, the confidence matrix is not necessarily symmetric.

$$C = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 1.0 & 1.0 & 0 \\ 0.5 & 1.0 & 0.4 \\ 0 & 0.5 & 1.0 \end{pmatrix} \end{array}$$

VI. Empirical Validation

To provide empirical results, we simulated the process described in Section III on a set of industrial data. We use the model presented in Section V to make the suggestions. We propose an initial procedure, our base case, and then propose three improved procedures which ideally filter out items mistakenly suggested by the base case. To evaluate these procedures, we test them with data from three year's use of a CMDB at CA. This section describes the data used, the experiment setup, and the results obtained.

A. Description of the Data

Format of the Change Order Reports: In the CMDB, a Change Order has several fields including the requester, the assignee, the start date of the change, the description and the change set field (called the 'Configuration items' fields). Of the fields in a change order, we use only the change set field. The advantage of only using this one field is that change sets are easy to extract and easy to understand and, more importantly, as we will show, they may be useful as the basis for predicting future change sets.

Size of the System: The set of industrial data we used to evaluate our technique was provided by CA's Global Information Systems (GIS) team. Internally, CA uses its own CMDB product to manage its internal network and the services it provides to internal or external customers. The GIS team manages the all of CA's internal IT system worldwide. Table I shows the number of change orders containing data in the 'Configuration Items' field per year in the GIS CMDB during the three year period, January 2006 – December 2008, examined in this work.

Year	Number of Change Orders
2006	8,307
2007	9,784
2008	9,214
Average Per Month	758
Total Number of Change Orders Studied	27,305

Table I
NUMBER OF CHANGE ORDERS PER YEAR IN THE GIS SYSTEM

From the three years of data, we determined the following facts about changes in the system. The average size of a change set is 4 CIs. At the end of 2008, there was a total of 37,906 CIs in the system. However, over the examined three year period, only 7,999 CIs were involved in the change reports (about 21% of the total number of CIs). The fact that not all the CIs have changes associated with them is not surprising since the CMDB keeps track of a large variety of CI types. These types range from software applications to hardware such as printers or LCD screens whose changes would not necessarily be logged in the CMDB.

B. Experiment Setup

To apply the change impact analysis process described in Section III, we mined the change orders stored in the CMDB repository to extract the change sets. We use reports from three consecutive years: 2006, 2007 and 2008. We start by constructing the support and confidence matrices from January 2006, and use this knowledge to predict the change sets in February 2006 (i.e. learning period = Jan 2006 and testing period = Feb 2006). We then add the data from February 2006 to our matrices, and attempt to predict the change sets in March 2006 (i.e. learning period = Jan – Feb 2006 and testing period = March 2006), and so on until December 2008.

For each learning period, we calculate the support matrix, and then calculate the confidence matrix according to Equation 4. To calculate the support matrix, *S*, we look at each change order in the specified learning date range, and increment the count of each CI that occurred in this change order as well as incrementing the count corresponding to each pair of CIs in the change order. That is, if a change order has CIs A and B, then S_{AA}, S_{BB}, S_{AB}, and S_{BA} are all incremented.

For each testing period, the aim is to use the model (support and confidence matrices) constructed from the data in the learning period to reproduce the existing change sets given an initial CI in the change set. This technique is commonly used in evaluating research ideas (e.g., [1] and [2]), when deploying the tool in a production system is not feasible. For each change in the testing period, we choose one of the CIs in the change set (Initial CI), and then suggest which other CIs should be changed along with it. To be able to compare results from the different filters we use, we fixed the Initial CI to be the first CI listed in a change set where all change sets are sorted by the CI identifiers. We then suggest a list of CIs that should also be changed based on their corresponding values in the matrices.

To simulate the interaction of the analyst with DRACA, we check which of these suggestions actually exists in the occurred set. Any suggested CI that happens to also exist in the occurred set will be treated as accepted by the analyst. For each accepted CI, DRACA looks for more suggestions. This process continues until no more accepted CIs can be found, or no more suggestions can be found.

For example, consider this change set: $\{x, y, z, w\}$. In this example, the initial CI would be x making the occurred set $O = \{y, z, w\}$. Given x, assume DRACA suggests the following set of CIs: $\{y, r, w\}$. Since y and w are part of the occurred set, DRACA looks for more suggestions given that y and w are accepted. Assume DRACA now suggests $\{l, m\}$. Since none of the suggested CIs are in the occurred set, DRACA stops iterating making the predicted set, $P = \{y, r, w, l, m\}$. The intersection set would be $PO = \{y, w\}$. We can now compute recall and precision based on the sizes of sets: occurred $|O| = 3$, predicted $|P| = 5$ and intersection $|PO| = 2$. This means that for this change set, recall = $|PO|/|O| = 2/3 = 0.667$, and the precision = $|PO|/|P| = 2/5 = 0.4$.

An experiment run consists of reproducing all the change sets from February 2006 to December 2008. For each experiment run, we calculate the average recall and precision values from all the change sets in the testing periods. However, to allow time for the learning process to stabilize, we exclude the first five testing periods, and calculate the average recall and precision for the change sets in July 2006 to December 2008. The F-measure for each experiment run is then calculated based on these average recall and precision values.

C. Experimental Results

We ran the above procedure on our three year data-set. We first present the results from applying this procedure to a base case which suggests each CI that has occurred in the past with a CI accepted by the analyst. We then present the results from running the procedure while applying three filtering techniques each of which remove some of the suggestions made in the base case. These techniques apply

Support Threshold	Recall	Precision	F-measure
0 (Base Case)	0.9423	0.0983	0.1780
10	0.7973	0.4674	0.5893
20	0.7488	0.6679	0.7061
30	0.7215	0.7572	0.7389
40	0.7032	0.7925	0.7452
50	0.6900	0.8307	0.7539
60	0.6793	0.8706	0.7635
70	0.6709	0.8866	0.7638
80	0.6627	0.8995	0.7635
90	0.6558	0.9095	0.7621
100	0.650	0.9208	0.7621
110	0.6451	0.9295	0.7616

Table II
RECALL/PRECISION/F-MEASURE WITH INCREASING SUPPORT THRESHOLDS (FILTER 1)

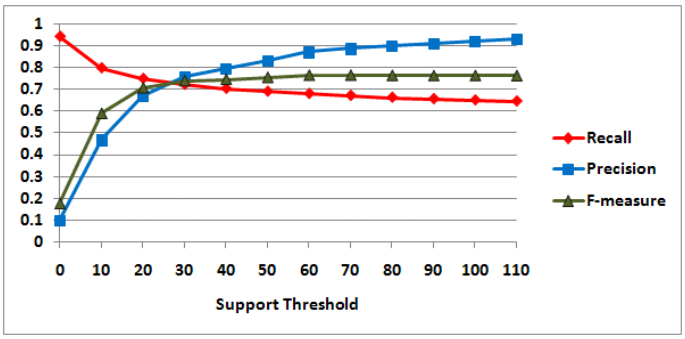

Figure 4. Recall/Precision/F-measure with increasing support thresholds (Filter 1)

a support threshold, a confidence threshold, and exponential forgetting. We compare the performance of each of these filters to the base case.

Base Case

The base case begins by determining which items have occurred together in a change set in the past. It then predicts members of a change set as follows. When an item x is known to be in the current set (as determined by an analyst), the base case suggests each item y that has previously occurred in a change set with x. That is, S_{xy} is greater than zero. For example, using the support matrix in Section V, given that CI B is in the change set, DRACA would also suggest A and C since they have non-zero entries in the support matrix.

This approach is simple and seems to be promising. When we ran our experimental procedure on this approach, it produced a recall of 0.9423. However its precision was only 0.0983, which is so low as to be of doubtful utility. This produces an F-measure of 0.178. We conclude that the base case makes too many suggestions, thereby producing high recall but too low precision. Consequently, we proceed to apply filters to prune out some of the base case's suggestions, hoping to gain precision at ideally a reasonable cost in decreased recall.

Filter 1: Support Threshold

We use the support matrix to count how many times each pair of CIs changed together. We use these counts as our first filter to refine the base case as follows. When an item x is known to be in the change set (as determined by an analyst), Filter 1 suggests each item y that has previously occurred in a change set with x more than t times, where t is a parameter called the support threshold. Otherwise, y is not suggested. For example, two CIs that have changed together 10 times in the past is stronger evidence that they may change together in the future when compared to two CIs that have changed only 5 times together in the past. Our expectation is that such a threshold will eliminate enough suggested CIs to increase precision and thereby to improve upon the base case. Table II shows the effect of varying the support threshold. The graph in Figure 4 shows the plot of these results.

Larger values of the threshold prune out more suggestions that would have been made by the base case. If the threshold is zero, no pruning takes place and this procedure devolves to the base case; this can be seen in the first row of Table II. As can be seen, as the support threshold increases, the precision increases with an accompanying decrease in recall. This, in turn, causes the F-measure to improve to about 0.76 at a threshold of about 70, and then starts decreasing. At that point the recall is approximately 67% and the precision is approximately 89%.

Although these values seem to indicate that this filter is potentially useful in practice, in fact it cannot be expected to continue to work well. This is because the counts of pair occurrences (i.e., support) continues to grow with time, while the support threshold remains constant. Eventually a problem will arise, in that the increasing support counts will allow more and more suggestions, so precision will be degraded.

Filter 2: Confidence Threshold

Filter 1 improved the F-measure of our predictions, but it, unfortunately, depends on the length of time the system has been running. The second filter (confidence threshold) is much like the first filter but avoids this problem by using relative counts. When an item x is known to be in the change set (as determined by an analyst), this procedure suggests each item y such that y's confidence with respect to x is greater u, where u is called the confidence threshold.

To illustrate how Filter 2 works, consider the example confidence matrix we presented earlier in which C_{BA} = 0.5 and C_{BC} = 0.4. If B is known to be in a change set, and if the confidence threshold is 0.2, then both A and C will be suggested as likely members of the change set because both C_{BA} and C_{BC} exceed 0.2. However, if the confidence threshold is 0.45, only A will be suggested.

Confidence Threshold	Recall	Precision	F-measure
0 (Base Case)	0.942	0.098	0.178
0.1	0.924	0.320	0.476
0.2	0.899	0.399	0.552
0.3	0.872	0.476	0.616
0.4	0.850	0.529	0.652
0.5	0.816	0.625	0.707
0.6	0.791	0.682	0.732
0.7	0.753	0.768	0.760
0.8	0.718	0.837	0.773
0.82	0.712	0.850	0.775
0.84	0.710	0.857	0.776
0.86	0.695	0.887	0.779
0.88	0.678	0.899	0.773
0.9	0.666	0.914	0.770
0.92	0.658	0.924	0.769
0.94	0.650	0.932	0.766
0.96	0.644	0.938	0.764
0.98	0.638	0.942	0.761
1.0	0.594	1	0.745

Table III
RECALL/PRECISION/F-MEASURE WITH INCREASING CONFIDENCE THRESHOLDS (FILTER 2)

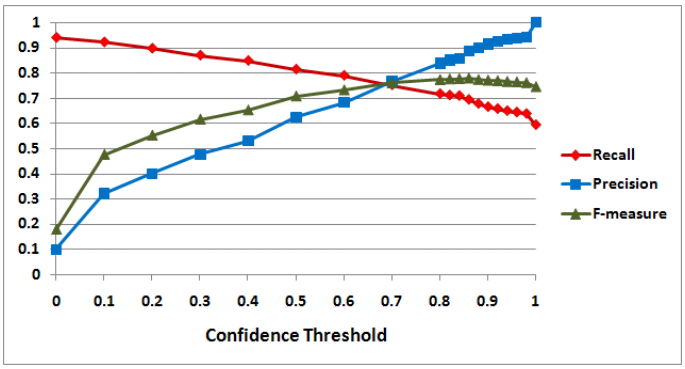

Figure 5. Recall/Precision/F-measure with increasing confidence thresholds (Filter 2)

Table III shows the effect of varying the confidence threshold on the recall and precision levels, and accordingly on the F-measure. Figure 5 shows the plot of these results. As in the case of the previous filter (support threshold), larger values of the threshold prune out more of the suggestions that would have been made by the base case. Similarly, if the threshold u is zero, this procedure devolves to the base case; this can be seen in the first row of Table III.

The table and figure show that as the confidence threshold is raised, the F-measure keeps improving until a threshold of about 0.8 is reached and then starts falling. To find a more exact value for which the F-measure maximizes, we tested all confidence thresholds form 0.8 – 1.0 in intervals of 0.02. This showed that the F-measure reached a maximum value of 0.779 at a threshold of about 0.86, and then started falling. At that maximum value, recall was about 70% and precision was about 89%. These values are somewhat better than those from the previous filter, but more important is the expectation that this second filter will keep producing good recall and precision with the passage of time. These high values indicate that this filter is potentially useful in practice.

Filter 3: Exponential Forgetting

Our third filter is based on the idea that more recent information should count for more. Since we are dealing with dynamic systems, the relationship between CIs may change over time or the way they depend on each other may change. Therefore, more recent change sets should reflect the current state of the system better than change sets that took place a year ago, for example. When applying exponential forgetting, we use the concept of half-life to indicate how fast the forgetting occurs. The half-life measures after how much time will we only remember half the amount of any information. This determines the rate at which we forget previous information. A shorter half-life means quicker forgetting of information, while a longer half-life means retaining any learned information for a longer period of time.

As previously explained, in order to calculate the support matrix, we count the number of times CIs appear in the change reports. If no exponential forgetting is used, then every time a CI appears in a change report, we increment its count by 1 regardless of the time this change occurred at. Let us call the amount we increment by, in this case 1, the *impact* of the new piece of information. With exponential forgetting, the impact of a new piece of information will depend on the current time and on the time this piece of information occurred at. Given the half life, λ, we use the following formula to calculate the impact, I_{t_0} at the present time t_{now} of a change set that occurred at time t_0:

$$I_{t_0} = 2^{-\frac{(t_{now}-t_0)}{\lambda}} \tag{5}$$

In our work, we update the support and confidence matrices every month. Therefore, when predicting the change sets for April for example, we assume that t_0 is March 31st. The impact of the data learned from previous months in the support and confidence matrices is adjusted accordingly. After we are done predicting the change sets in April, the new t_0 time becomes April 30th, and we add the change sets in April to our matrices, and adjust their impact accordingly.

Before applying exponential forgetting, we filter using our best results from Filters 1 and 2. This is done by applying Filter 2 (which produced better results than Filter 1) with its confidence threshold set to 0.86 (the best setting of that threshold). Table IV and Figure 6 show the results of exponential forgetting with that threshold active.

As shown, varying the half-life only slightly improved results. The results suggest that the best half-life is 12 months with an F-measure of approximately 0.7803.

Half-life	Recall	Precision	F-measure
3 months	0.7074	0.8623	0.7772
6 months	0.6987	0.8782	0.7782
9 months	0.6982	0.8828	0.7798
12 months	0.6980	0.8847	0.7803
15 months	0.6965	0.8858	0.7798
18 months	0.7176	0.8453	0.7763
21 months	0.6961	0.8867	0.7799
24 months	0.6960	0.8868	0.7799
27 months	0.6965	0.8858	0.7798
30 months	0.6960	0.8868	0.7799
33 months	0.6959	0.8869	0.7799
36 months	0.6959	0.8869	0.7799
∞	0.6945	0.8869	0.7790

Table IV
RECALL/PRECISION/F-MEASURE WITHE A CONFIDENCE THRESHOLD OF 0.86 AND INCREASING HALF-LIFE, λ (FILTER 3)

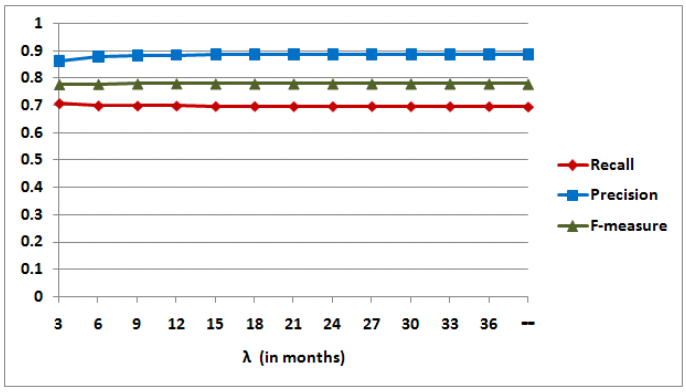

Figure 6. Recall/Precision/F-measure with a confidence threshold of 0.86 and increasing half-life, λ (Filter 3)

D. Discussion of Results

The obtained results seems promising since they indicate that the change set data is highly predictive, with high values of recall and precision. However, a question that arose during these experiments is this: Why are the values of recall, precision and the F-measure so high? These values are higher than figures from experiments such as predicting change sets in source code updating [1], [2], [10]. This question became more intriguing with the bottom row of Table III which documents the situation when the confidence threshold is 1.0. This threshold implies that the procedure will make absolutely no suggestions. In other words, the procedure will predict that the change set contains nothing but the item originally provided by the analyst. As the table shows, with this threshold, the F-measure (0.745) is reasonably high.

Both the high recall (0.594) and high precision (1.0) contribute to this high F-measure. The perfect precision can be explained as follows. Since precision measures the percentage of suggestions made that were correct, making no suggestions at all means making no mistakes at all which produces a perfect precision of 1.0. This maximal possible value of precision helps make the F-measure be high.

However, it was the corresponding high recall value that came as a surprise. Achieving perfect precision is usually accompanied by a very low recall rate, but this was not what we observed. The corresponding observed high recall value (0.594) has the following explanation. In the analyzed data, we found out that 16,294 change orders out of the total of 27,305 change orders studied contained only 1 CI in their change set. This means that roughly 59% of the change sets in the data contain exactly one CI. Each such CI will be selected by the analyst as the initial CI leaving the occurred set empty. This means that the occurred set is empty about 59% of the time. If the occurred set is empty, recall is 1.0 by definition [1]. However, if the occurred set is not empty, and no suggestions are made, then recall is 0. Since 59% of the change sets have empty occurred sets, it follows that the average recall of all change sets is about 59% in the case in which the procedure makes no suggestions.

In our experiment, we determined that a value of 0.86 for the confidence threshold u produces the maximum F-measure value (0.779). We interpret this as follows. This u value (0.86) is close enough to 1.0 so it suggests no CIs most of the time which matches the change sets of size 1. Additionally, when it does make suggestions, it suggests only high frequency pairs that have a high chance of being correct. This explains why the experiment has such simultaneous high recall and precision results.

One interpretation of this situation is as follows. Filter 2 learned to perform well (as shown by its recall, precision and F-measure values) and it accomplished this by making no suggestions in many cases. Another interpretation would question the convention of defining precision to be 1.0 in the case of an empty suggestion set as this definition seems to inflate the value of precision. Another interpretation or approach would ignore all singleton change sets and would re-run the experiments using only change sets of size at least two. The authors favor the first interpretation, but recognize that the other interpretations have merit. Regardless of the interpretation, it appears that Filter 2 may adapt reasonably well to other historical data of change sets that either do or do not contain many singleton change sets; future work may confirm this position.

Figure 7 compares the recall-precision curves of the first two filters. These curves are based on the precision and recall columns of Tables II and III. The top left point of these two curves corresponds to the base case. As the figure shows, the curve for Filter 1 (support threshold) lies below that of Filter 2 (confidence threshold). This indicates that in all cases Filter 2 outperformed Filter 1. Since Filter 2 produced the best results in our experiments, we used confidence to rank the set of suggested CIs that are displayed to the analyst.

Applying exponential forgetting with the optimal confidence threshold of 0.86 slightly improved results (raising

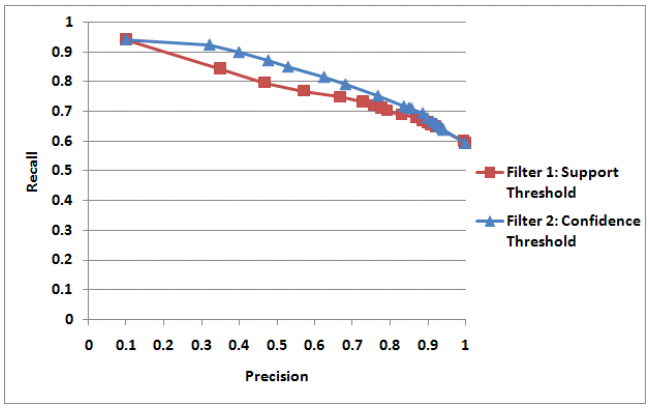

Figure 7. Comparing recall and precision for Filter 1 (support threshold) and Filter 2 (confidence threshold)

the F-measure from 77.9% to 78.03% with a half life $\lambda =$ 12 months). We believe that this is due to the nature of the system data we are analyzing with is more or less stable in the sense that once a change set occurs, it is likely that it will occur again in the future. With a more dynamic system, we expect that varying the half-life would produce more significant variances in the results.

E. Threats to Validity

Although our results seem promising, we still cannot conclude that our findings will apply to different systems. We presented our results from analysis of one system, which is the system used by CA's GIS team. We reviewed one other system, but it did not use change sets so we were not able to apply our technique to it. Ideally, in the future we may be able to analyze more systems which use change sets, but this may be challenging as it is not easy to gain access to industrial systems. Analyzing additional systems would allow us to better evaluate the utility of various filter.

VII. Related Work

There has been much work done on how to predict the propagation of change of one software entity to another at the source code level (e.g., [11], [12]). Our work is analogous to the work done by Hassan et al. [1] where they perform similar analysis to ours but on the level of the source code. They study four main heuristics to predict change sets, historical co-changes being one of them. They use pruning techniques to improve the recall and precision values while combining their proposed heuristics in different ways. Without pruning, the heuristics produced high recall values, but very low precision values. With a hybrid technique that combines historical co-changes with file structure information and pruning, an average recall of 0.51 and an average precision of 0.49 was achieved.

Zimmermann et al. [2] present their tool, ROSE, which guides programmers during the change process for updating source code by suggesting other program entities that previous programmers have changed in the same situation. This is done by mining the CVS repository to discover which program entities were changed together. Rules are then produced based on the entities that change together frequently, and are assigned a support and confidence level as previously described above [3]. Their results show that for stable systems, ROSE obtained a precision of 0.44 and a recall of 0.28.

In related work, Canfora et al. [4] predict change sets by mining textual descriptions of change requests rather than considering entities that changed at the same time only. That is, when a change request is received, related change requests are retrieved based on the similarity of the textual descriptions of both reports, and accordingly, the changed files in the retrieved reports are recommended according to the rankings.

The goal of the above related work, as well as other similar work [10], [13], is to predict possible source code changes and change impacts. On a service or system level, there has been less work dedicated to identifying the effect of a change. Work in this direction includes that by Kumar et al. [14] which present an ontology which helps in identifying and quantifying the impact of changes. Similarly, de Boer [15] uses heuristics based on the semantic meanings of the relationships between components in an architecture to identify the ripple effect of a change. However, there is no other work, to our knowledge, that applies data mining techniques and uses historical data to predict change propagation on the system level.

VIII. Future Work

In this paper, we used simple heuristics to investigate if the nature of changes stored in a CMDB is predictive or not. Since our results seem promising, we intend to expand on this work using more of the available information. As shown in Section VI-A, there are several input fields in a change order that we might use in the future to derive additional information to help predict change impact. For example, looking at the description of the change along with the CIs which changed can allow us to have a classification of the different types of changes, and to predict which CIs to change based on the nature of the change. Additionally, looking at which analysts perform which changes can allow us to recommend the best analyst to perform the current change.

We also plan to add more decision support to our tool. For example, DRACA could not only suggest other CIs to change, but also the best time to implement this change based on the availability schedule of all CIs involved in the change set. Additionally, to reduce the probability of a change causing a failure sometime later in the system, we could bundle change orders with incident reports in the CMDB to identify which changes induced incidents, similar to the work in [16].

IX. Conclusion

Proactively identifying CIs that will be impacted by a change can prevent costly outages. On the long run, this makes an IT system more maintainable. Through examining previous change sets in CA's CMDB, we provided a decision-support technique for creating change sets. DRACA suggests the CIs that should be included in a change set, and also provides a ranking of these CIs based on their pattern of recurrence in the past. By modeling this recurrence in support and confidence matrices, and by applying different filters, we were able to predict change sets with a combination of recall and precision as high as 69.8% and 88.5% respectively.

We presented the effect of different filters we applied to the suggested set of CIs. These include a support threshold, a confidence threshold and exponential forgetting. The confidence threshold seemed to be the most effective threshold and greatly improved results. Although we mine our data out of the CMDB repository, this work can be applied on any repository with a different format as long as change sets are recorded.

Our results are promising and show that the change sets are highly repetitive. Our next step is to use additional sources of data such as calendar information for example to be able to provide more advice and decision-support to the analyst.

Acknowledgments

This research is supported by a research grant from CA Canada Inc, and is partly funded by OCE and NSERC.

We would like to thank CA customers for their valuable contributions. Many thanks to the CA GIS team for sharing their data with us. We would also like to thank Ian Davis and Mike Godfrey for their feedback on our ideas.

References

[1] A. E. Hassan and R. C. Holt, "Predicting change propagation in software systems," in *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 284–293.

[2] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 563–572.

[3] T. Zimmermann, S. Diehl, and A. Zeller, "How history justifies system architecture (or not)," in *IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution*. Washington, DC, USA: IEEE Computer Society, 2003, p. 73.

[4] G. Canfora and L. Cerulo, "Impact analysis by mining software and change request repositories," in *METRICS '05: Proceedings of the 11th IEEE International Software Metrics Symposium*. Washington, DC, USA: IEEE Computer Society, 2005, p. 29.

[5] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *ICSM '98: Proceedings of the International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 1998, p. 190.

[6] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions Software Engineering*, vol. 26, no. 7, pp. 653–661, 2000.

[7] "Office of government commerce (ogc), ed.: Service support," *IT Infrastructure Library (ITIL)*, 2000.

[8] S. Nadi, R. Holt, I. Davis, and S. Mankovskii, "Draca: Decision support for root cause analysis and change impact analysis for cmdbs," in *CASCON '09: Proceedings of the 19th Centre of Advanced Studies Conference*, Toronto, Canada, November 2009.

[9] C. van Rijsbergen and P. D, "Information retrieval," 1979.

[10] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004.

[11] L. C. Briand, J. Wuest, and H. Lounis, "Using coupling measurement for impact analysis in object-oriented systems," in *ICSM '99: Proceedings of the IEEE International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 475 – 482.

[12] M. Petrenko and V. Rajlich, "Variable granularity for improving precision of impact analysis," in *ICPC '09: Proceedings of the 17th IEEE International Conference on Program Comprehension*, Vancouver, BC, Canada, 2009, pp. 10 – 19.

[13] A. R. Sharafat and L. Tahvildari, "A probabilistic approach to predict changes in object-oriented software systems," in *CSMR '07: Proceedings of the 11th European Conference on Software Maintenance and Reengineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 27–38.

[14] A. Kumar, P. Raghavan, J. Ramanathan, and R. Ramnath, "Enterprise interaction ontology for change impact analysis of complex systems," in *APSCC '08: Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 303–309.

[15] F. de Boer, M. Bonsangue, L. Groenewegen, A. Stam, S. Stevens, and L. Van Der Torre, "Change impact analysis of enterprise architectures," in *Information Reuse and Integration, Conf, 2005. IRI-2005 IEEE International Conference on.*, 2005, pp. 177–181.

[16] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*. New York, NY, USA: ACM, 2005, pp. 1–5.