# Quantitative Analysis for Non-linear System Performance Data using Case-based Reasoning

Jacky W. Keung[1,2] and Thong Nguyen[3]
[1]National ICT Australia Ltd. (NICTA) Sydney Australia
[2]CSE, UNSW, Sydney Australia
[3]Air Operation Division, Defence Science and Technology Organisation, Australia
Jacky.Keung@nicta.com.au, Thong.Nguyen@dsto.defence.gov.au

## Abstract

*Effective software architecture evaluation methods are essential in today's system development for mission critical systems. We have previously developed MEMS and a set of test statistics for evaluating middleware architectures, which proven an effective assessment of important quality attributes and their characterizations.*

*We have observed it is common that many system performance response data are not of linear nature, where using linear modeling is not feasible in these scenarios for system performance predictions.*

*To provide an alternative quantitative assessment on the system performance using actual runtime datasets, we developed a set of non-linear analysis procedure based on Case-based Reasoning (CBR), a machine learning method widely used in another disciplines of Software Engineering.*

*Experiments were carried out based on actual runtime performance datasets. Results confirm that our non-linear analysis method CBR4MEMS produced accurate performance predictions and outperformed linear approaches.*

*Our approach utilizing CBR to enable performance assessments on non-linear datasets, a major step forward to support software architecture evaluation.*

**Keywords:**
Software Measurement, Case-based Reasoning, Software Architecture Evaluation

## 1. Introduction

In the field of empirical software engineering, experimental data from experiments on software are analyzed using various methods to devise laws and theories, these methods include case studies, surveys and statistical analysis methods. The purpose of this paper is to investigate data analysis strategies for system performance experimental data derived from the middleware architecture evaluation test bed environment for airborne mission critical systems. Linear modeling methods have been used in the previous studies[1, 2], result shows that many system performance datasets under investigation were not of linear nature, therefore alternative non-linear methods are required to fully investigate the nature of the data.

This paper introduces a novel non-linear modeling approach using Case-based Reasoning (CBR) from the artificial intelligence research field. A tailored approach for middleware architecture evaluation combining a sensitivity procedure has been developed in the study, namely CBR4MEMS.

Software systems may response differently due to different characteristics in the middleware architectural configurations and the execution scenarios in the experiments. Therefore systems will behave differently but not necessarily in a linear and predictable fashion. CBR4MEMS utilizes *K-NN* (*k-nearest neighbor*) algorithm to find the similar cases to the target problem in a way that is unique and novel to the data analysis of software architecture evaluation.

Empirical experiments were carried out and show CBR4MEMS performed extremely well in all the experiments among different scenarios, with a minor exception when applied on a particular dataset, detailed will be discussed in Section 5. CBR4MEMS successfully predicts response time of each individual performance metric accurately, and cross validation based on different experiments confirmed this finding.

Section 2 presents an overview of the background of the study and related work. Section 3 describes proposed method and the underlying theory using Case-based Reasoning. Section 4 provides the datasets and experiment procedure. Section 5 presents the result, and section 6 discusses the result and provides further directions. Section 7 concludes the paper.

1530-1362/10 $26.00 © 2010 IEEE
DOI 10.1109/APSEC.2010.47

## 2. Background and Related Work

This study is a continued effort to previous projects on evaluating middleware architecture for Airborne Mission Systems that developed a Method for Evaluating Middleware architectureS (MEMS) [3] [1].

### 2.1 MEMS Evaluation for Middleware

The MEMS evaluation approach provides an important evaluation framework and useful information on the relationship between software architecture [4] and one or more quality attributes to ensure that the system ultimately achieves its quality goals while at the same time supporting its functional requirements [3]. MEMS was developed specifically for the COTS [5] middleware architectures, requiring architects and designers to have considerable knowledge and experience of using middleware. Details are given in Gorton et al.[5].

In general, MEMS evaluation process can be classified into two application stages. The first stage consists of the development of an evaluation plan and comprises the following steps:

- Determine critical quality attributes
- Identify key architecture patterns
- Develop key scenarios
- Define metrics for each quality attribute.

The second stage of MEMS involves 3 steps:

- Prototyping
- Carrying out the experiments
- Analyzing the measurement

Additional details can be found in Liu et al. [1]

The system performance data produced by MEMS helps to determine the suitability of the architecture to meet quality goals of the system. Measurements of performance metrics are collected based on the test scenarios developed in the initial study [1] [3]. Following the evaluation process defined in MEMS, and system performance datasets are then collected from the established test bed platform.

Experiments based on the scenarios were executed in the test bed environment at the Defence Science and Technology Organization (DSTO). [3] One key challenge is that a large amount of experimental system runtime data for analysis produced from the test bed. In addition, given the raw datasets were collected in the real time environment, we have observed that external influential factors have had an significant impact to the response times of the systems. Outlying data points were discovered in the previous experiment [2]. Another challenge is that datasets collected are of non-linear nature, applying linear modeling techniques do not allow an accurate prediction of the system performance in the given datasets from DSTO, it is therefore an appropriate non-linear modeling approach must be applied. A non-linear system is a system does not satisfy the superposition property, of which its output is not directly proportional to its input, result in a non-linear trend in the data pattern. For example weather forecast is non-linear in nature, where changes in conditions of external influential factors will produce complex outputs to the weather condition. There are different non-linear methods to simulate and model the problem, in this study we focus on case-based reasoning which is relevant to our situation with a focus on the analysis of the performance datasets.

### 2.2 Case-based Reasoning

Case-based reasoning (CBR) has been widely used in many real world scenarios, particularly in software engineering. Case-based Reasoning also known as Analogy in other areas such as in Software Effort Estimation, a basic human reasoning process used by almost every individual on a daily basis to solve problems based upon similar event(s) that happened in the past. CBR has been used extensively for the purpose of software effort estimation where the posed problem is similar to that of the system performance evaluation for MEMS. In software engineering, for example, Shepperd et al. [6] pioneered the idea of using CBR to improve software effort prediction accuracy, Keung et al. [7] later improved the CBR system by introducing a sensitivity analysis mechanism to ensure only relevant project cases are entered into the CBR system before estimation. In many cases, CBR is suitable for a dataset where a non-linear relationship does not exists

The data analysis approach developed in this work is based on the following principle:

*"Data points that are similar with respect to its characteristics will also be similar with respect to its response time (i.e. performance)"*

This study provides the means of formally testing whether this hypothesis is true for the datasets available by DSTO. In our observation, linear modeling approaches do not provide a means of suitable prediction of performance based on past performance data, and this is largely due to the distribution of the data points observed in the datasets. In many cases, variations in response times are due to various uncontrollable factors outside the scope of the experiment. To capture these non-linear patterns, we propose to use Case-based Reasoning (CBR), which follows 4-stage general case-based reasoning (CBR) process [8], which consisting of:

- *RETRIEVE* the most similar cases or cases to the target problem
- *RESUSE* the past information and solution to solve the new problem
- *REVISE* the proposed solution and to better adapt the target problem
- *RETAIN* the parts of current experience in the case-base for future problem solving

The following diagram (Figure 1) depicts the general cyclical CBR process, showing important steps and interactions in each stage of its application process.

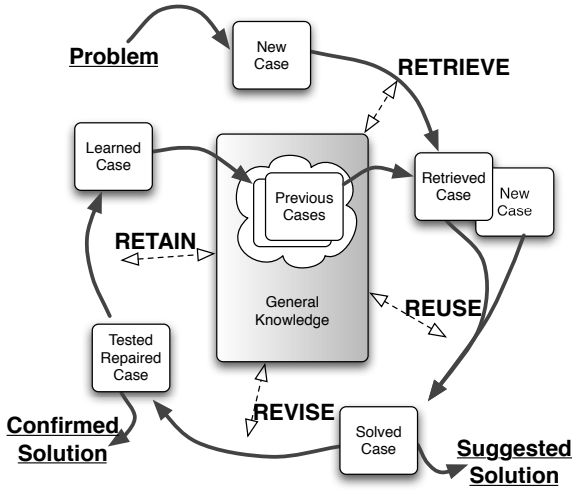



**Figure 1 Anatomy of the CBR Cycle**

### 2.3 K-Nearest Neighbor (KNN)

The similarity between the target new case and each case in the case-base is determined by a similarity measure. Different methods of measuring similarity have been proposed for different measurement contexts. A similarity measure is measuring the closeness or the distance between two objects in an *n*-dimensional Euclidean space. The result is usually presented in a distance matrix (similarity matrix) identifying the similarity among all cases in the database. The Euclidean distance metric is probably the most commonly used in CBR for its numerical distance measures. It is based on the principle of Pythagorean theorem to derive a straight-line distance between two points in *n*-dimensional space.

In general, the Euclidean distance between two points $P = (p_1, p_2, \ldots, p_n)$ and $Q = (q_1, q_2, \ldots, q_n)$ in Euclidean *n*-dimensional space, is defined and calculated as:

$$\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\cdots+(p_n-q_n)^2}=\sqrt{\sum_{i=1}^{n}(p_i-q_i)^2} \quad (1)$$

The Euclidean distance measure is suitable for general problems, particularly when values are of continuous nature. Of course, there are other variants of distance metrics available for other types of data, these include, but are not limited to Jaccard distance for binary distance [9] and Gower distance described by Gower & Legendre [10]. In this study, we only consider the Euclidean distance measure, as the variables we are measuring are continuous.

The application of *KNN* in this study allows a more reasonable modeling of the given datasets showing non-linear patterns. Figure 3 shows the dataset of a particular measurement (*Handoff_to_thread_pool*). The *KNN* approach will derive a reasonable estimate based on similar occurrences in the given dataset, rather than based on a linear function of the dataset.

## 3. CBR4MEMS Approach

The general principle of data-intensive CBR is to reuse knowledge in the form of past cases stored in the dataset. An estimate of the response time to complete a task with different number of tracks is made by identifying and comparing one or more similar data points from the training dataset. The general principle of CBR4MEMS approach follows the general CBR cycle is illustrated in Figure 1. In addition, we use a dataset pre-processing step before its application similar to the outlier removal method applied in [2]. The pre-processing step is important, as it will be used to identify and remove influential data points that will have major impact to the overall result. Cases (data points) within a reasonable statistic range are withheld for the analysis. Figure 3 provides an overview of the workflow of our CBR4MEMS approach.

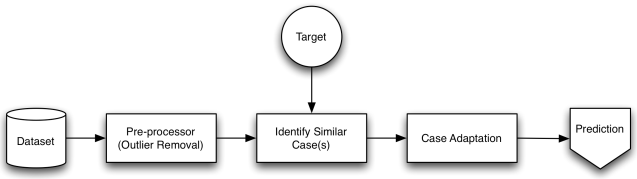



**Figure 2 The CBR4MEMS application process.**

CBR4MEMS approach is developed to supplement the exiting MEMS framework, to provide an advanced quantitative data analysis tool for non-linear datasets in the MEMS evaluation process.

### 3.1 DSTO-CBR Data Pre-processing Step

To minimize the impact of influential outlying data points (observations appear to deviate from the rest of the data) in the dataset, we developed a Leverage Metric (*LM*) in [2] to support sensitivity analysis for the assessment of stability for the dataset. This allows the automatic detection of these abnormal data points and is able to exclude them from the evaluation.

The preliminary assumption for this analysis is that the variation of the data points within a dataset should be within a tolerable range. The tolerable range is arbitrary defined according to the required level of stability of the system. For example at 90% stable, or within 10% error is tolerable for a system.

The leverage metric (*LM*) is based on calculating the residual of each data point to the fitted regression line of the dataset. This indicates the extent to which the data trend is influenced by each individual case. To calculate *LM* for each case *i*, let *E*(*Xi*) be the expected value for the data point *Xi* which is the predicted value of *i* based on the regression model of the entire dataset, *Xi* is the actual observed value from the experiments, then *LMi* is given by:

$$LM_i = X_i - E(X_i) \tag{2}$$

*LMi* is the difference (residual) between the observed data point *Xi* and its expected value *E*(*Xi*) based on all the data points excluding data point Xi, indicating the impact of the specific case *i* on the entire dataset. The expected value *E(Xi)* is a predictive function based on all data excluding *Xi* to predict the outcome of *Xi*. Under the null hypothesis that case *i* is not abnormal, *Xi* will be an estimator of *E(Xi)* and their difference is *LMi*. For the purpose of abnormality detection, the following *z* test provides a mechanism to formally verify whether the value of *Xi* is an abnormal one. For each case *i*, *LMi* can be converted to its standard normal form:

$$Z_i = \frac{LM_i}{S} \tag{3}$$

The *Zi* value provides an indication of the confidence level of a particular data point being abnormal by using the standard empirical rule. The empirical rule or the three sigma rule states that for a normal distribution, almost all values lie within 3 standard deviations of the mean, where about 68% of the values lie within 1 standard deviation of the mean, or about 95% of the values lie within 2 standard deviations of the mean, or 99.7% of the values lie within 3 standard deviations of the mean. In this case, the mean is the expected value of *Xi* derived from the fitted model. Any data point outside the required limit will be removed.

The removal of these outliers is significantly important; inclusion of these data point will have an impact on the overall analysis.

### 3.2 Model Prediction Performance Criteria

An important aspect of any prediction model is to understand how accurate the model is to predict the target under investigation. In CBR for software effort estimation, accuracy is usually defined in terms of mean magnitude of relative error (MMRE), which is the mean of absolute percentage errors. For the sake of the experiment, we adopted the same generic evaluation metric in this study, because the principle and the application procedure of CBR is somewhat similar in this situation. MRE forms the basis for MMRE and is defined as follows:

$$MRE = \frac{|Actual - Prediction|}{Actual} \tag{4}$$

Each prediction in the validation subset produces an MRE*i* value and the mean of all MREs becomes MMRE, an overall measure of prediction performance of the system applied on the entire dataset.

Similar to any accuracy measures, there have been some criticisms of MMRE measure, in particularly of its unbalanced nature [11]. In order to confirm the results, another approach is to use Pred(25) which is the percentage of predictions that fall within 25 percent of the actual value. The choice of accuracy measure to a large extent depends upon the objectives of those using the prediction system. To remove this concern, in this study we have decided to adopt both MMRE and Pred(25) as prediction performance indicators as both are widely used in empirical evaluation studies in software engineering.

### 3.3 Empirical Evaluation using n-fold cross validation

Cross-validation or rotation estimation is a commonly used technique in statistics for assessing how the results of an analysis procedure will generalize to an independent dataset. The goal in cross-validation is determine how accurately a predictive model will perform in practice, in this case a predictive model for system performance datasets.

*n*-fold cross-validation is also known as leave-one-out cross-validation or jack-knifing commonly known in statistics. *n*-fold cross-validation involves using a single observation from the selected sample dataset as the validation data, and the remaining observations as the training data. This is then repeated, as such, that

each observation in the sample is used once as the validation data.

For example, given 100 samples in a dataset ($X_1$, $X_2$, $X_3$...$X_{100}$), $X_1$ is first removed from the dataset, leaving 99 samples remaining, and then these 99 samples are used to build a predictive model to predict $X_1$, a result is produced namely $R_1$. Once completed, $X_1$ is pushed back into the dataset, similarly $X_2$ is removed from the dataset, its predictive value is computed based on the remaining 99 samples. This process iterates *n*-times (100 times in this case), ensuring each single data point is examined using the remaining data points.

*n*-fold cross-validation examines every single point in the entire dataset using the remaining observations, thus, this is the most accurate technique. The trade-off for such accuracy is that it is usually very expensive to compute, because of the large number of repeated dataset trainings performed.

| Metric | Description |
|---|---|
| *tm_cpu* | CPU utilisation |
| *handoff_to_thread_pool* | Track Handoff Time |
| *update_or_add_track* | Track Add and Update Time |
| *for_each_response_time* | Response Time of for_each() |
| *getTracks_response_time* | Response Time of getTracks() |
| *uftd_SR_time* | Synchronized Read in uftd() |
| *uftd_SW_time* | Synchronized Write in uftd() |
| *all_sync_times* | Sum of all of Synchronized Read and Write (i.e. synchronisation points) |

**Table 1 System performance metrics**

For a broader spectrum of study, we examined the influences of different values of *k* used in the *K-NN* algorithm. To evaluate the influences of *k*(nearest neighbor), we applied both *k*=2 and *k*=3 in each of the experiments. *K* is the number of most similar cases to be selected to produce a solution (i.e. target adaptation); the adaptation strategy used in CBR4MEMS is the average values of all the selected similar data points.

## 4. Performance Data Collection

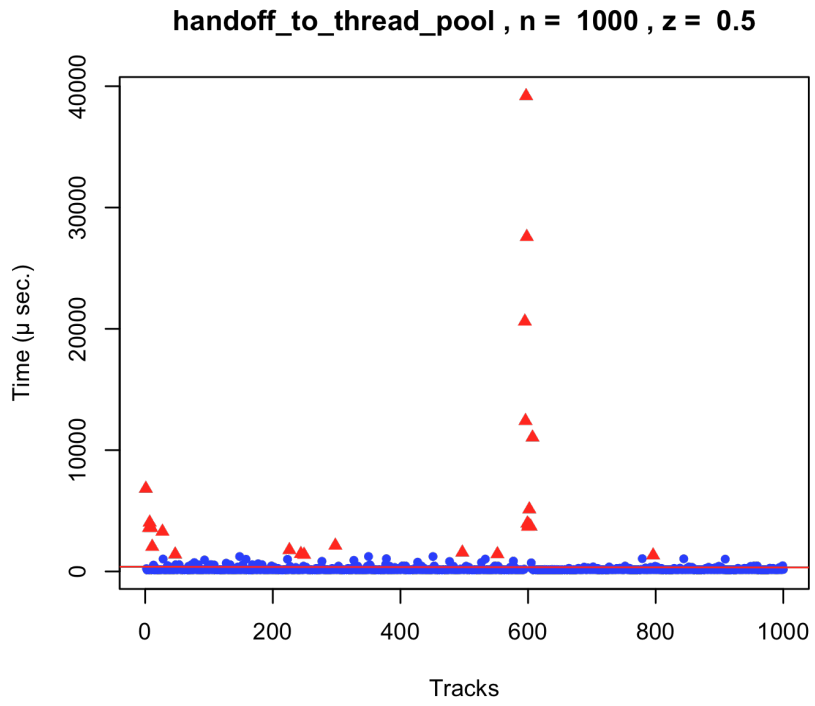
This study continues on the previous study on software architectural evaluation using MEMS. A test bed environment had been setup to collect useful performance runtime data. The core of the measurement plan forms the *test scenario*, as MEMS is a scenario-based approach. The test scenarios are designed in incremental cases. Individual operations are tested first with different workload levels, and then the experiments with combination of the operations are observed. The reason is to identify the performance characteristics of individual operations in order to pinpoint defective operations or software components. Otherwise the complexity of the systems leads to difficulties in monitoring and analyzing the system's performance accurately. This situation is exuberated by the fact that the system is built from COTS middleware.

Incremental changes in parameters will reveal different aspects in different scenarios. The total number of tracks, the number of track writers and the rate that track writers generating loads are parameters controlled in the experiments.

Different performance metrics are collected in Table 1 during the execution of each experiment. Five experiments (different scenarios and parameters) were performed to evaluate architectural attributes and produced performance datasets and these datasets will be used to evaluate the performance of CBR4MEMS.

## 5. Application of CBR4MEMS

Following the procedures described in Section 3 (see Figure 2), each dataset will be first examined for its abnormal data points and followed by the execution of CBR4MEMS, the evaluation of prediction accuracy of CBR4MEMS are calculated using MMRE and PRED25. A typical example of the effect of before and after removal of abnormal data points is illustrated in Figure 3. This section discusses the results of five experiments using CBR4MEMS in detail.



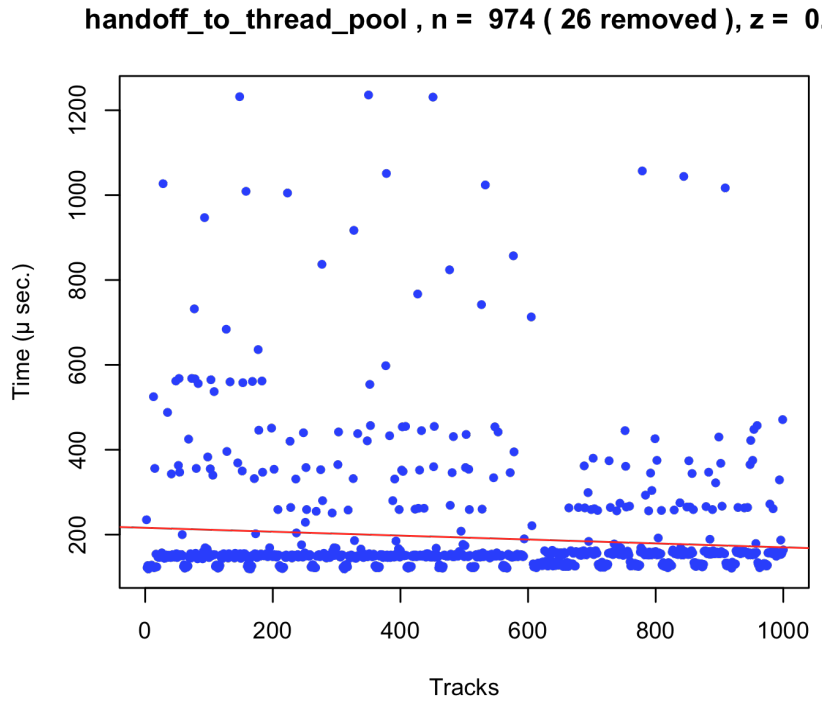




**Figure 3 Graphical scatter plots of dataset *Handoff_to_thread_pool* (Exp. #2). Top graph shows the original dataset, bottom graph shows the effect of 26 outliers removed. Linear regression trend is depicted in red line.**

## 5.1 Experiment 1, 2 and 3

The result of applying CBR4MEMS to Experiment 1 is shown in Table 2.

*Parameters:*
Track Writer: 1
Rate: 1 Track / Sec
Tracks: 1,000
Thread Pool: 10,000

*handoff_to_thread_pool* – A large amount of outliers were removed based on statistics, resulting 556 useful samples (Table 2). Based on MMRE evaluation criteria, it shows at least 98% of the predicted values are accurate, this is confirmed by the measure of *PRED25* using $K$=2 and $K$=3 resulting 0.9928 and 0.9964 respectively. CBR is extremely accurate for this dataset.

*tm_cpu* – *MMRE* and P*RED25* show that CBR4MEMS is able to predict up to 97% of the response time value.

*uftd_SR_time* – 3 data points were removed. MMRE and PRED25 show that CBR is able to accurately predict at least 90%. There is a slight variation of using different values of $K$ in this case, however their difference is not statistically significant.

*uftd_SW_time* – 257 outliers data points were removed, resulting 100% accuracy in *MMRE* and *PRED25*, this is largely due to the small variations in the dataset.

*update_or_add_track* – Both *MMRE* and *PRED25* confirms that CBR is capable of predicting up to 97%.

Of 1000 samples given in this experiment, CBR performed extremely well in modeling the relationship between each of different metrics and their correspondent response times. Empirical results show that applying different $k$ values (number of nearest neighbors) does not significantly changes the result.

Experiment 2 and 3 (in Table 3 and Table 4 respectively) are somewhat similar to experiment 1, only the rate of input has changed to 10 tracks/second. Similar to the results produced in Experiment 1, CBR performed extremely well in this experiment.

Figure 3 shows the effect of data preprocessing. Noticeably only 26 outlying data points were removed in the sample subset of *handoff_to_thread_pool*. Figure 3 is a typical data illustration of the datasets in the experiments, given the extreme outlier has a value of close to 40,000 micro second, which is far distant apart to the main samples collected in the range between 100 to 400. Without having these extreme outliers removed it would be difficult to produce any sensible conclusion.

The *handoff_to_thread_pool* subset in the second experiment, CBR is able to predict between 84 to 87 percents using $K$ = 2 and 3 respectively (Table 3). As a contrast, a linear regression line is plotted using a red line in the bottom of Figure 3. Using the $k$-nearest neighbor approach we can estimate the outcome based on individual similarity in characteristics, which is not, restricted by the linear trend of a linear model, therefore the result is relevant and accurate. *tm_cpu, uftd_SR_time, uftd_SW_time and update_or_add_track* produced similar accurate results confirming CBR4MEMS's robustness in this experiment.

## 5.2 Experiment 4 and 5

In experiment 4 (see Table 5), the track writer input has been increased to 5 and the track rate has been increased to 50/sec, resulting 5,000 samples captured in each repeat.

Results show that the predictions have been affected by the sample size, rendering its predictability for *handoff_to_thread_pool, uftd_SR_time and all_sync_times*. In the *handoff_to_thread_pool* subset, given only 52% of the samples are remained after data preprocessing, the *MMRE* predictability is only 32%, nevertheless better than using linear regression in our previous experiments. It seems due to large amount of variations in the sample dataset, an accurate predictive model cannot be derived in this particular case. Further investigations show that the there are regions of data are clearly deviated from the main stream, causing unpredictability. However, other three subsets (*tm_cpu, uftd_SW_time and update_or_add_track*) are giving consistently good prediction results.

Experiment 5 combines read, write and update operations in the MEMS evaluation framework, resulting a variation of number of responses. The overall results of 6 assessments show consistently good accuracies in the range between 90% to 97% based on MMRE, and these results are further confirmed by PRED25 indicating its outperformed prediction accuracy. The result also shows the influence of the value $K$ used in the *KNN* algorithm is not significant in this experiment.

**Experiment #1 (id: s1_e1)**

| Performance Metric | Original Size | Reduced (z=0.5) | K=2, MMRE | K=2, PRED25 | K=3, MMRE | K=3, PRED25 |
|---|---|---|---|---|---|---|
| handoff_to_thread_pool | 1000 | 556 | 0.0227 | 0.9928 | 0.0216 | 0.9964 |
| tm_cpu | 29 | 26 | 0.0319 | 1.0000 | 0.0523 | 1.0000 |
| uftd_SR_time | 1000 | 997 | 0.1067 | 0.8716 | 0.1055 | 0.9218 |
| uftd_SW_time | 1000 | 743 | 0.0000 | 1.0000 | 0.0000 | 1.0000 |
| update_or_add_track | 1000 | 972 | 0.0344 | 1.0000 | 0.0325 | 1.0000 |

**Table 2 CBR4MEMS Results of Experiment 1**

**Experiment #2 (id: s1_e2)**

| Performance Metric | Original Size | Reduced Size | K=2, MMRE | K=2, PRED25 | K=3, MMRE | K=3, PRED25 |
|---|---|---|---|---|---|---|
| handoff_to_thread_pool | 1000 | 974 | 0.1335 | 0.8380 | 0.1696 | 0.7998 |
| tm_cpu | 108 | 68 | 0.0034 | 1.0000 | 0.0083 | 1.0000 |
| uftd_SR_time | 1000 | 989 | 0.1147 | 0.7988 | 0.1084 | 0.8756 |
| uftd_SW_time | 1000 | 642 | 0.0000 | 1.0000 | 0.0000 | 1.0000 |
| update_or_add_track | 1000 | 971 | 0.0525 | 0.9938 | 0.0535 | 0.9949 |

**Table 3 CBR4MEMS Results of Experiment 2**

**Experiment #3 (id: s1_e3)**

| Performance Metric | Original Size | Reduced Size | K=2, MMRE | K=2, PRED25 | K=3, MMRE | K=3, PRED25 |
|---|---|---|---|---|---|---|
| handoff_to_thread_pool | 1000 | 556 | 0.0227 | 0.9928 | 0.0216 | 0.9964 |
| tm_cpu | 29 | 26 | 0.0319 | 1.0000 | 0.0523 | 1.0000 |
| uftd_SR_time | 1000 | 997 | 0.1067 | 0.8716 | 0.1055 | 0.9218 |
| uftd_SW_time | 1000 | 743 | 0.0000 | 1.0000 | 0.0000 | 1.0000 |
| update_or_add_track | 1000 | 972 | 0.0344 | 1.0000 | 0.0325 | 1.0000 |

**Table 4 CBR4MEMS Results of Experiment 3**

**Experiment #4 (id: s2_e1)**

| Performance Metric | Original Size | Reduced Size | K=2, MMRE | K=2, PRED25 | K=3, MMRE | K=3, PRED25 |
|---|---|---|---|---|---|---|
| handoff_to_thread_pool | 5000 | 2646 | 0.6726 | 0.6448 | 0.7083 | 0.7166 |
| tm_cpu | 92 | 24 | 0.0082 | 1.0000 | 0.0373 | 1.0000 |
| uftd_SR_time | 5000 | 4983 | 0.2014 | 0.8378 | 0.1570 | 0.8320 |
| uftd_SW_time | 5000 | 4978 | 0.1673 | 0.5306 | 0.1457 | 0.7965 |
| update_or_add_track | 5000 | 4891 | 0.0940 | 0.9183 | 0.0650 | 0.9667 |
| all_sync_times | 10000 | 9983 | 0.7111 | 0.0175 | 0.5049 | 0.1695 |

**Table 5 CBR4MEMS Results of Experiment 4**

**Experiment #5 (id: s3_e1)**

| Performance Metric | Original Size | Reduced Size | K=2, MMRE | K=2, PRED25 | K=3, MMRE | K=3, PRED25 |
|---|---|---|---|---|---|---|
| handoff_to_thread_pool | 2564 | 2563 | 0.05413 | 0.9902 | 0.04495 | 0.9945 |
| tm_cpu | 285 | 131 | 0.0967 | 0.9026 | 0.1134 | 0.8896 |
| uftd_SR_time | 2564 | 1679 | 0 | 1 | 0 | 1 |
| uftd_SW_time | 2564 | 2083 | 0.02797 | 1 | 0.02669 | 1 |
| update_or_add_track | 2564 | 1486 | 0.02379 | 1 | 0.02376 | 1 |
| all_sync_times | 5128 | 3342 | 0.07361 | 1 | 0.06882 | 1 |

**Table 6 CBR4MEMS Results of Experiment 5**

## 6. Discussions

We have demonstrated the proposed CBR4MEMS approach for the data analysis of runtime experimental data. Unlike the previous study using a set of linear modelling techniques, we have applied CBR a machine learning technique and can be used for predictions on non-linear datasets.

The overall result in the experiments is encouraging and it shows that the CBR4MEMS approach is capable of modelling and predicting response times under different scenarios. This provides a valuable tool in addition to the set of linear modelling techniques reported in the previous studies [2] [1, 3]. In all 5 experiments using different scenarios, we demonstrated CBR4MEMS outperformed linear regression in terms of modelling capability as well as prediction accuracy.

Although CBR4MEMS delivers exceptionally good prediction accuracy, based on the evaluation criteria MMRE and PRED25, we are able to confirm that CBR4MEMS is a suitable approach for the MEMS scenario-based evaluation for middleware architecture runtime performance datasets derived from various scenarios and experiments.

The characteristics of raw datasets also play an important role in the data analysis. We have found different levels of outliers exist in various different datasets across different scenarios and experiments. We have observed that after outlier removal, the reduced subset is relatively smaller than the original sample size. This is certainly alarming, the implication here is that there are insufficient qualified data points on which to base estimate, or there are other external influential factors heavily influencing the results produced by the test bed system. Possibilities include but not limited to network bandwidth, disk storage, other device I/Os as well as other shared loading applications can also cause significant delays in the response time of the system.

From a statistics point of view, the outliers are removed based on their relevancy to the main stream of data, a data point largely deviated from the main group of data will warrant its status being flagged as an outlier. However, the sensitivity of this analysis depends on the acceptable limit given by the user.

There are limitations we have observed using CBR4MEMS. For example in experiment 4, the proposed approach can predict up to 32% for the *handoff_to_thread_pool* component, this is largely due to the characteristics of the given dataset. Similar to any other modelling approaches, when a dataset displays random-like characteristics, their predictable relationship cannot be reasonably modeled. The suitability of the selected approach is important, if a dataset displays a linear characteristic then linear approaches would be appropriate. On contrary, if a dataset has a non-linear characteristic or cannot be reasonably modeled by a linear method, then non-linear approaches may be applied as an alternative. In this study, we have introduced a non-linear approach based on CBR to solve some of the problems that cannot be easily modeled by using a linear method, such as linear regression.

We also suggest the following for future MEMS architecture evaluation studies:

1. The system under evaluation should be deployed on dedicated hosting environment and running on real-time operating systems. This helps to isolate external influential factors in the measurements.
2. Sensitivity analysis should be repeated using various degrees of z values to ensure a suitable value is applied. As the system under test is running on a COTS middleware, the internal behavior of the middle ware is a black box to the evaluators. The assumptions need to be confirmed with regards to the behaviors that may have an impact on the performance and scalability.
3. The requirements of worse case scenarios should be identified for evaluation. This can also provide more precise rating scale definition that leads to more accurate evaluation results.

## 7. Conclusion

CBR4MEMS is a non-linear procedure developed specifically for MEMS, the DSTO runtime test bed performance datasets were used in the study. It was developed to assess the non-linear nature of the datasets produced from the test bed environment generated by MEMS, where the application of linear modelling is simply out of question. In addition, a sensitivity analysis procedure was used to filter abnormal outlying data points that causing significant influences to the evaluation. Together CBR4MEMS is a robust and complete automation technique for assessing runtime performance datas based on MEMS.

Empirical observations in the study show that a large number of outlying data points exist in the datasets provided by DSTO. The removal of these outliers is necessary to ensure a valid evaluation can be carried out without bias. However, the removal of large amount of outliers will cause the available samples to be significantly reduced. This is not desirable, and subject to further investigation into what was the cause of these large amounts of outliers in the datasets.

Empirical experiments conducted on various scenarios and experimental datasets, based on the performance evaluation criteria of MMRE and PRED25, the model prediction results are satisfactory and we are confident that CBR4MEMS is an useful technique for evaluating runtime performance datasets. In most of the cases the accuracy of 80~95% can be achieved using the proposed approach. Empirical validations were carried out using *n*-fold cross validation also confirms its usefulness in the evaluation of non-linear datasets.

In this research study, we have provided a novel and robust approach adopting case-based reasoning method (CBR) from the machine-learning domain. The approach used in the previous study was a linear modelling approach, limiting some of its use on all the measures given in the DSTO datasets.

Further research is necessary to improve the overall efficiency and performance of CBR4MEMS in the future. CBR4MEMS will construct an *NxN* distance matrix each time, as the sample size (*N*) increases the required amount computation power will also increase.

The ability to evaluate and predict the runtime performance based on the simulated datasets for software architectural evaluation developed in this study is of great practical significance. The CBR4MEMS approach is a novel and robust approach that improves the performance and reliability of performance prediction systems, thus supporting important architectural decision making on mission critical software systems.

## 8. Acknowledgement

NICTA (National ICT Australia Ltd.) is funded through the Australian Government's Backing Australia's Ability Initiative, in part through the Australian Research Council.

## 10. References

[1] Y. Liu, *et al*., "MEMS: A method for evaluating middle architectures," presented at the International Conference on Quality of Software Architecture (QoSA), 2006.

[2] J. Keung, *et al*., "A Statistical Method for Middleware System Architecture Evaluation," in *Australian Software Engineering Conference 2010* Auckland, New Zealand, 2010.

[3] Y. Liu, *et al*., "Quality Assessment of Mission Critical Middleware System Using MEMS," in *International Conference on Quality Software (QSIC)*, Jeju, Korea, 2009.

[4] L. Bass, *et al*., *Software Architecture in Practice*: Addison-Wesley, 2003.

[5] I. Gorton, *et al*., "Rigorous evaluation of COTS middleware technology," *Computer,* vol. 36, pp. 50-55, 2003.

[6] M. Shepperd, *et al*., "Effort estimation using analogy," in *International Conference on Software Engineering*, Berlin, Germany, 1996.

[7] J. Keung, *et al*., "Analogy-X: Providing Statistical Inferences to Analogy-based Software Cost Estimation," *IEEE Transactions on Software Engineering,* vol. 34, pp. 471-484, 2008.

[8] A. Aamodt and E. Plaza, "Case-based Reasoning: Foundational issues, methodological variations, and system approaches," *Artificial Intellegence Communications,* vol. 7, p. 59, 1994.

[9] P. N. Tan, *et al*., *Introduction to Data Mining*: Addison-Wesley, 2006.

[10] J. C. Gower and P. Legendre, "Metric and Euclidean properties of dissimilarity coefficients. ," *Journal of classification,* vol. 5, 1986.

[11] T. Foss, *et al*., "A Simulation Study of the Model Evaluation Criterion MMRE," *IEEE Transactions on Software Engineering,* vol. 29, pp. 985-995, 2003.