Machine Learning Strategies for Temporal Analysis of Software Clone Evolution using Software Metrics

Jayadeep Pati, Babloo Kumar, Devesh Manjhi, and K. K. Shukla
Department of CSE, IIT(BHU), Varanasi, Uttar Pradesh, India.

Abstract: During software evolution, there is a tendency to duplicate the code, and modify the copy slightly, giving rise to clones. Cloned code fragments adversely affect software quality and maintenance. In this paper, we discuss identification of different types of clone components using Abstract Syntax Tree based approach and also propose models for prediction of the evolution of cloned components in future versions of the software. The primary focus of the paper is modelling of the evolution of clones in a software application. Detection of clones in a large software system is challenging as it depends on the internal design of software modules and methods. Object-oriented metrics like DIT, NOC, WMC, LCOM, and Cyclomatic complexity can be used as good indicators of clone contents. We demonstrate a correlation between clones and various metrics of the source. The first part of our study is to identify the cloned components using Abstract Syntax Tree. The second part is to predict the evolution of cloned components using advanced time series modelling using machine learning approaches. Evaluation of our model is performed using a large open source software system. The assessment includes quantifying the correlation between software metrics and the clone contents in the software.

Keywords: Software Clones, Software Maintenance, Software Metrics, Machine Learning, Time Series Analysis, Exact Match Clones, Near-Miss Clones, Abstract Syntax Tree.

INTRODUCTION

Software evolution is an important aspect in the field of software development. Software evolves due to changes in the requirements, to adopt to changes in the environment and also to improve the software quality. During software evolution, there is always a tendency of code fragment being copied or modified slightly in the same as well as subsequent versions. If we can detect these recurring code fragments and model them, it can be immensely helpful in the software maintenance activities.

The cloned components are considered as bad smell [35] for software as they always increase the maintenance effort. Cloning also enhances the probability of increasing number of bugs in the software application due to the faulty code fragment being repeatedly used in different places. These cloned code fragments have a bad effect on software maintainability and quality of software. It also leads to the poor understandability of the software system as it takes more time to understand the code. Code clones are of different types based on semantics and syntax. The textually similar clones are exact copies of the code fragments. They are called type 1 clones. There also exists code with slight variability in literals, constants, class, layouts, and comments. These are called type 2 clones. Codes with additional statements in addition to type 2 clones are considered as type 3 clones. There also exist clones with a similar functionality called semantic clones or type 4 clones [35].

Detecting the cloned fragments is an important but challenging task. There has been a large number of studies on software clone detection [20, 21, 22, 23, 24]. There exist many clone detection approaches based on types of cloning. They are Line based technique [1, 2, 3, 4], Metric based technique [5, 6, 7, 8], Token based techniques [9, 10, 11], Tree based techniques [13, 14, 16, 17], PDG (Programme dependency graph) based techniques [17, 18, 19] and also Abstract Syntax Tree (AST) based technique [12, 15]. The AST completely captures the whole system information and is a most efficient clone detection approach [35]. In this paper, we have
identified software clones based on an Abstract Syntax Tree (AST) and metric based approach using a popular AST-based Clone extraction tool - CloneDR [12]. A large number of papers have identified clones based on the same version of software systems. The primary work of this paper is to model the clones across different versions of the software systems and to predict the clone evolution pattern.

Object-oriented metrics like DIT, NOC, WMC, LCOM and Cyclomatic complexity are good indicators of code clone existence [25] and can be used as input feature vectors for prediction of clone evolution. This research paper also models the dependency of software clones on the various metrics of the source code. The models are validated on an open source software application ArgoUML [26].

The rest of the paper is organized as follows: Section 2 describes clone detection process. Section 3 presents a description of software metrics and software clones. Section 4 describes the clone evolution prediction. Section 5 describes the data preparation steps. Section 6 presents a description of implementation and result. Section 7 describes the evaluation and interpretation. Section 8 describes the application of the clone evolution modelling. Section 9 concludes the paper and shows direction for future work.

SOFTWARE CLONE DETECTION

The first phase of our work is to identify the cloned components in the software application. In this paper, we have used an AST based detection approach for the identification of the cloned components. A typical clone detection process involves the following steps:

1. Pre-processing
2. Code Transformation
3. Clone Extraction
4. Match Detection
5. Clone Detection

In the pre-processing steps, the source code is partitioned into disjoint units with granularity level varying from statements, blocks, and functions, etc. to Files. This step removes the unnecessary code elements from the source code and is also important in deciding the domain of comparison. In the next step, we transform the code into an intermediate representation. Now the clones are extracted from the intermediate state. In this paper, we have used an Abstract Syntax Tree based approach for clone extraction. CloneDR is an advanced AST based clone detector. The compiler is used for generating the AST. The compiler performs a comparison among the sub-trees based on some hash function. The similar subtrees based on the hash function are returned as clones [35].

In this paper we have used abstract syntax tree based clone detection using CloneDR, a clone detection tool [12]. It accepts multiple parameters to set a threshold of similarity between clones. It outputs two types of clone sets. After processing, Exact-match clone sets and Near-miss clone sets were obtained.

1. Exact-match Clone Sets (EMCS): It is the count of cases in which some code block has been cloned without any changes [6].
2. Near-miss Clone Sets (NMCS): It is the count of abstractions with parameters for which there are multiple clone instances [6].
Table 1. Software Metrics used for Software Clone Evolution Prediction

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>Weighted Methods per Class</td>
</tr>
<tr>
<td>NSC</td>
<td>Number of Children</td>
</tr>
<tr>
<td>NORM</td>
<td>Number of Overridden Methods</td>
</tr>
<tr>
<td>NOF</td>
<td>Number of Attributes</td>
</tr>
<tr>
<td>NSF</td>
<td>Number of Static Attributes</td>
</tr>
<tr>
<td>NOM</td>
<td>Number of Methods</td>
</tr>
<tr>
<td>NSM</td>
<td>Number of static methods</td>
</tr>
<tr>
<td>NOC</td>
<td>Number of Classes</td>
</tr>
<tr>
<td>NOI</td>
<td>Number of Interfaces</td>
</tr>
<tr>
<td>NOP</td>
<td>Number of Packages</td>
</tr>
<tr>
<td>MLOC</td>
<td>Method Line of Codes</td>
</tr>
</tbody>
</table>

Figure 1 and Figure 2 show the time series representation for EMCS and NMCS respectively for ArgoUML [9].

SOFTWARE METRICS AND SOFTWARE CLONES

Software metric is a standard way of measuring a software system or process. The metrics help to find the degree to which a software system possesses some property. As metrics are a unique way of measuring software systems, there exist a correlation between software metrics and cloned contents. Several metrics for object-oriented systems have been proposed [25], and new metrics are still being introduced. The metrics used in time series analysis of software clone evolution are given in Table 1. The detailed description of software metrics is presented in the paper [29].

Software Metrics Extraction

ArgoUML [26], an open source software application is used for conducting our experiments. We downloaded the source-code without libraries for each of the 19 releases of the ArgoUML software. All the experiments are carried out in the Eclipse environment. We have used the eclipse metrics plugin [27] to extract the various software metrics for each release.

CLONE EVOLUTION PREDICTION

Clone evolution prediction is the primary focus of our paper. In this phase, we apply time series modelling techniques to model the clone number series that we obtained from the previous phase. The model is used for predicting clones in subsequent versions of the software. The model also helps in analyzing the correlation between software metrics on clone evolution. The objective is to build an accurate predictive model for clone evolution by capturing the relationship between the metrics and clones.

Design of Experiments

In this section the detailed experimental procedure for time series modelling of clone evolution is discussed. We have used both univariate and multivariate time series modelling techniques for prediction of software clone evolution using software metrics.

We used the lag length parameter to transform the time series data into available machine learning data. If we have a time series \( y(t) \) and the lag parameter is \( k \), then it treats the first \( k \) lagged values of \( y(t) \) i.e. \( y(t−1), y(t−2), \ldots, y(t−k) \) as explanatory variables. We also add an extra feature named artificial time index to predict the response variable \( y(t) \) [28]. If initially, we had \( n \) instances or observations in dataset then after this transformation we remove the first \( k \) instances with unknown lag-values [28].

After transforming the time series data using lagged value concept, we used the linear regression and Multi-Layer Perceptron to model the multivariate time series jointly. The algorithms use nonlinear hypothesis for analysis and are much powerful and flexible than conventional methods like ARMA or ARIMA [34].

Machine Learning Algorithm Used For Time Series Analysis

We have used following machine learning algorithms in modelling the clone evolution series.

Linear regression

It gives a linear model with the help of training examples to predict for a given test data. If there are \( m \) features against a target variable, and \( X \) be the feature-vector of order \( m \times n \), it fits the model with a weight vector of order \( m \times 1 \), \( W = (w_1, w_2, \ldots, w_m) \) moreover, a bias term \( b \) to minimize the sum of the square of errors between the actual and predicted values of the target variable.
Let $Y(Actual Value) = (Y_1, Y_2, \ldots, Y_n)$ be the vector of actual values of the target, 
$
\hat{Y}(Predicted Value) = (\hat{Y}_1, \hat{Y}_2, \hat{\ldots}, \hat{Y}_n)
$
be the vector of predicted values, 
\[ \hat{Y} = b + W^T \cdot X \]
For $(W, b)$ giving minimum value of $\|Y - \hat{Y}\|^2$.

**Multi-Layer Perceptron**

Multilayer perceptron (MLP) is an artificial neural network that has one or more hidden layers of neurons between the input layer and output layer \([30]\).

Let $X$, be the input vector comprising of the features of a particular training example.

Let the output vector of the hidden layers be $Y_1, Y_2, Y_3, \ldots, Y_m$
Where, $m=$ number of hidden layers. The final output $Y$ is obtained through the layer-by-layer transfer of the output. If the transfer function used is the logarithmic sigmoid function
\[
f(x) = \frac{1}{1 + e^{-x}}
\]

\[
Y_1 = f(b_1 + w_1^T X)
\]
\[
Y_2 = f(b_2 + w_2^T Y_1)
\]
\[
\ldots \ldots \ldots
\]
\[
Y = f(b_m + w_m^T Y_m)
\]

Where $w_i$ are the weight vectors of the neurons of i-th layer and $b_i$ is the vector of the bias terms for each neuron of i-th layer. The Diagrammatic representation of MLP is given in figure 3.

**DATA COLLECTION**

The analysis was done on 19 versions of ArgoUML ranging from version 0.10.1 to the latest stable release 0.34.0 \([26]\). These span a duration of over 9 years from 2002 to 2011. ArgoUML is a UML diagramming application written in Java and released under the open source Eclipse Public License. It was decided to use this software as our code base because it has been used as an experimental software in many papers \([26]\). It was also part of MSR data challenge, 2006.

**FIRST MODEL**

Here we have modelled the clone sets individually using linear regression. If a clone-set time series is denoted by $Y_t$ then The model comprises of $Y_{t-1}, Y_{t-2}, Y_{t-3}, Y_{t-4}$ feature vectors for predicting $Y_t$ (EMCS or NMCS)
Where $Y_{t-1}, Y_{t-2}, Y_{t-3}, Y_{t-4} = lagged$ variables.

Here only lagged values are used for predicting the clone series.
Second Model

Here we build our predictive model by jointly modelling the relationship between the pair \((p, q)\) of a clone-set and each software metrics using linear regression.

\[ p \in Y_{t-1}, Y_{t-2}, Y_{t-3}, Y_{t-4} = \text{Lagged variables} \]

\[ q \in x1|x2|x3|...|x11 = \text{Numerical values for 11 Software Metrics extracted.} \]

Here lagged value and each software metrics value individually is used for joint modelling of clone series. In this model, we also got the correlation between each software metrics and the clone content.

Third Model

In this model, we include all the software metrics together along with lagged value for modelling of clone series. For all the models, we used lagged feature vectors up to size 4 as it gives a more accurate prediction and minimum error. The model for predicting the response variable i.e. Clone series (EMCS or NMCS) includes 4 lags of each software metric and lagged feature vectors. The diagrammatic representation of model 1, model 2 and model 3 is given in figure 4, 5, and 6 respectively.

EVALUATION AND INTERPRETATION

The models are evaluated based on the RMSE (Root Mean Square Error), MAE (Mean Absolute Error) and MAPE (Mean Absolute Percentage Error). RMSE, MAE, MAPE are calculated using following formulae.

1. 

\[ MAE = \frac{1}{n} \sum_{t=1}^{N} |(A(t) - F(t))| \]

2. 

\[ RMSE = \sqrt{\frac{1}{n} \sum_{t=1}^{N} (A(t) - F(t))^2} \]

3. 

\[ MAPE = \frac{1}{n} \sum_{t=1}^{N} \left| \frac{A(t) - F(t)}{A(t)} \right| \times 100 \]

Here: A (t): Actual Value, F (t): Predicted Value, N: Number of Terms.

The models are evaluated both on the Training data and Test Data. The Model which gives minimum error for Test Data is to be selected.
Evaluation of First Model

In this model only lagged value of clone series are used for prediction. The results after applying Linear Regression are given in Table 2 and Table 3.

Evaluation of Second Model

Here along with the lagged values of the clone series the individual metric values are also considered for modelling. This Model also determines the correlation between software metrics and Clone sets. Here we have 11 individual models for each metrics along with lagged values. Figure 7 and Figure 8 represent the Bar Chart representation of all MAPE for all the models for EMCS and NMCS respectively. The result shown is for test data. Here we have used linear regression for modelling.

Evaluation of Third Model

In this model, we consider all the metrics at once along with lagged values of clone series. Here we have applied both Linear Regression and Multilayer Perceptron for modelling. The result after joint modelling of clone number series with linear regression is given in Table 4 (for Test Data only). The result after joint modelling with multilayer perceptron is shown in table 5 (for Test Data only).

Interpretation of the Models

In this section, we give a comparative analysis of the predictive performance of all the models. We found that after including the software metric values, the errors are reduced drastically. We also found that after applying multilayer perceptron to our combined model 3 the error further reduces. So the inclusion of software metrics and nonlinear modelling is the most appropriate approach for prediction of software clone evolution. From the second model, we found that there
exists a correlation between software metrics and software clones. Figure 9 shows the comparison between Model 1 (Univariate modelling) and Model (Multivariate Modelling) using Linear Regression. Figure 10 shows a comparison of a nonlinear and linear model for Model 3 (Multivariate Model).

**APPLICATION OF THE MODEL**

The clone evolution prediction has a broad application in the field of software maintenance as it can reduce the effort for maintenance activities. The software testing and maintenance are the most expensive parts in the software engineering [31]. An early idea about clone evolution can result in reduction of maintenance and testing effort which in turn lowers the cost. Another application is in the field of software evolution. Many hypotheses have been proposed on how software changes and grows [32]. Clone evolution prediction will be helpful in the posterior validation of many software evolution hypotheses as given [33].

**CONCLUSION AND FUTURE WORK**

The paper presents a method for proper monitoring and predicting the evolution of clone numbers across different versions of the open source software applications. It is helpful in predicting effort for software maintenance. It is also useful for validating some software evolution hypotheses.

The paper also includes software metrics along with the lagged values for modelling the clone evolution. From the result, we found that software metrics are a good indicator of the existence of software clones in any software application. The paper also gives a comparative analysis of modelling software clone evolution. We found Multivariate Nonlinear Model as the most suitable model for predicting the clone evolution.

In the future, we can have more detailed analysis of the effect of individual software metrics on software clones. This will give a clearer picture of clones in software. We can also model the increasing and decreasing temporal patterns of the software clone evolution using advanced modelling techniques.

**REFERENCES**


[28] A. Rossen, “On the predictive content of nonlinear transformations of lagged autoregression residuals and time


