Towards a Generic Model for Software Quality Prediction

Zeeshan Ali Rana  
LUMS, DHA  
Lahore, Pakistan  
zeeshanr@lums.edu.pk

Shafay Shamail  
LUMS, DHA  
Lahore, Pakistan  
sshamail@lums.edu.pk

Mian Muhammad Awais  
LUMS, DHA  
Lahore, Pakistan  
awais@lums.edu.pk

ABSTRACT
Various models and techniques have been proposed and applied in literature for software quality prediction. Specificity of each suggested model is one of the impediments in development of a generic model. A few models have been quality factor specific whereas others are software development paradigm specific. The models can even be company specific or domain specific. The amount of work done for software quality prediction compels the researchers to get benefit from the existing models and develop a relatively generic model. Development of a generic model will facilitate the quality managers by letting them focus on how to improve the quality instead of employing time on deciding which technique best suits their scenario. This paper suggests a generic model which takes software as input and predicts a quality factor value using existing models. This approach captures the specificity of existing models in various dimensions (like quality factor, software development paradigm, and software development life cycle phase etc.), and calculates quality factor value based on the model with higher accuracy. Application of the model has been discussed with the help of an example.

Categories and Subject Descriptors
D.2.8 [Software Engineering]: Metrics—software science, product metrics

General Terms
Measurement

Keywords
Prediction, Generic models, Measurement-based prediction, Metrics

1. INTRODUCTION
Software quality prediction helps minimize software costs by allowing the mitigation of risks in early stages of software development process [4, 15, 8]. It further helps in preparation of better resource allocation plans [33, 29] and test plans [33, 13, 23, 19]. These factors help produce a good quality software which results in satisfied customer and healthier return on investment. Various organizations and public departments have been involved in studies related to quality prediction like Commission of the European Communities’ Strategic program for Research in Information Technology [1], Northern Telecom Limited, USA [15], Nortel, USA [14], NASA [18], National Natural Science Foundation of China [31] are a few examples. Earlier studies [13, 20, 33, 21, 22] have viewed quality in different aspects and have been limited to a particular quality factor (for example reliability, maintainability [5]). Moreover the application of those models is very context specific [28, 24]. Details of all these techniques can be found in [24]. Specificity of these approaches is a limitation of software quality prediction study and refrains the community from taking full benefit of the existing work. That is why a very small number of suggested models are used in industry [3]. A generic model based on existing models will help quality prediction endeavor. This paper suggests such a model which is composed of existing models (called component models). Given a software in terms of its collected metrics, the model selects the most appropriate applicable model to predict the desired quality factor value. The proposed model can be extended for as many quality factors as there are models available for. Like an integrated approach suggested by Wagner et al. [28], the proposed model also requires the prior analysis of the important quality factors. Furthermore, it will automate the selection of an appropriate prediction model.

Rest of the paper is organized as follows: In section 2 we discuss the related approaches towards generic models. We present our approach in section 3 and use an example to explain the suggested approach in section 4. In section 5 we discuss the limitations and issues with the suggested approach before concluding the paper in section 6.

2. RELATED WORK
Various studies on product-based quality prediction models have been done [7, 9, 13, 11, 25, 23]. Ganesan et al. employed case-based reasoning to predict design faults [7]. Grosser et al. [9] suggested a technique which was suitable for object oriented (OO) software only. Company specific [17] and domain specific [12] for telecommunication systems) studies have also been presented. Because the work has been done in various dimensions, need of a generic model for software quality has been felt [3, 27, 28, 30]. But the
application and context specific nature of existing models causes difficulty in taking full advantage of the existing work. Fenton et al. [6] have presented a critique of existing models and highlighted their weaknesses [6]. A few models, generic for a certain quality factor such as usability [30], have been suggested. Bouktif et al. [3] have considered the unavailability of large data repositories as an obstacle to generalize, validate and reuse existing models. They have suggested a technique for selecting an appropriate model from a set of existing models. Their approach reused existing models but it was restricted to selection of an appropriate model only. Their major focus was on facilitating a company in adapting object oriented software quality predictors to a particular context. Though Wagner [27] has suggested an approach to reduce the effort to apply a prediction model, some other issues yet need to be addressed. It is difficult to avoid specific behavior of a predictor of software quality. To address this issue models have been divided in different dimensions of specificity. Wagner et al. [28] have argued that software quality models differ along six dimensions namely purpose, view, attribute, phase, technique and abstraction. Rana et al. [24] have identified four dimensions for quality prediction models: software development paradigm (SDP), software development lifecycle (SDLC) phase, quality factor and approach of the model. Both [28] and [24] have two dimensions in common which are attribute (quality factor) and phase (SDLC phase). Dimension SDP [24] is an important information to capture regarding a model. Other issues which hinder the development of a generic model are inconsistent names of software metrics, and non-uniform input to the existing models. For example model by Guo et al. [10] refers to lines of code as TC (Total Code lines) whereas many other models (like [13]) refer to it as LOC. On the other hand, models by Khoshgoftaar et al. [15, 14] refer to TC as ‘Total calls to other modules’. Such inconsistencies make the existing models incomparable. Rana et al. [24] have done unification of metrics nomenclature. Their work paves the way towards development of a generic model.

3. OUR APPROACH

The block diagram of suggested prediction model is shown in figure 1. It outputs a set of quality factor values, QFValue, which can be expressed by the following expression:

\[ QFValue = f(\text{Software}, QF, SDP, SDLC \text{Phase}) \]  

Figure 1: Block diagram of suggested model.

Table 1: A snapshot from Unified Metrics Database

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unified Name</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>LOC</td>
<td>TC, LOC</td>
</tr>
<tr>
<td>Total Comments</td>
<td>TCOMM</td>
<td>COM, TCOMM</td>
</tr>
<tr>
<td>Cohesion Metric</td>
<td>CohM</td>
<td>COM, CohM</td>
</tr>
<tr>
<td>Depth of Inheritance Tree of a Class</td>
<td>DIT</td>
<td>DEPTH, DIT</td>
</tr>
</tbody>
</table>

Input Preprocessing: This phase handles the inconsistencies of input metrics’ names with the unified metrics database. The unified metrics database contains the data structures as shown in table 1 and is based on unifications done in [24]. This phase is carried out manually and it is needed so that appropriate models can be selected for this software.

Model Selection: Model selection phase has further five sub-activities. Each model is checked if its input interface has been unified or not. If the input interface is not already unified then ‘Unify Model Inputs’ sub-activity is carried out. Otherwise the search for relevant models is started. For example for a few models shown in figure 3 model Y and Z are relevant models to predict number of errors in object
for that type of software and quality factor. For example, model A which has 90% accuracy for object oriented software needs 12 metrics to do the same job. Assume there is a model A with 90% accuracy for object oriented software and it needs the metric named class hierarchy to predict stability. But this metric is not a part of the input software. Now there is another model B with 85% accuracy but all the required metrics are available. In such a case our selector will be comparing model B with other models which satisfy the constraint and will not consider model A for comparison.

4. AN EXAMPLE

In this section we elaborate the working of our model using dataset kc1 [2]. The dataset originally comprises of 94 attributes which were used by Koru et al. for study in quality prediction [16]. On the basis of other studies on object oriented metrics [32, 26], we have selected eight significant metrics for this study shown in table 3. Suppose that we have the models shown in figure 3. Let Y be a model based on Linear Regression (LR) and Z be a Support Vector Regression (SVR) based model. Their accuracies (calculated using first 100 instances of the dataset kc1) are 69% and 83% respectively. Rest of the information regarding the model remains as it is in figure 3. Now our model is provided with the following information in addition to the list of software whose quality factor value is to be determined:

\[
SDLCPhase = \text{Design} \\
SDP = \text{ObjectOriented} \\
QF = \text{NumberOfErrors}
\]

As shown in figure 2 the first step is ‘Input Preprocessing’. At this step unification of input metrics is done by using ‘Unified Metrics’ database. The unified names are shown in third column of table 3. In the next step the inputs of all models are unified. For example in our case, there are two models Y and B as shown in figure 3 which need their
inputs to be unified. We ‘Unify Model Inputs’ and store back the model in the repository with inputUnifiedFlag set to true. The next step is to ‘Prune Irrelevant Models’. Figure 4 shows that only models Y and Z are relevant models and the model Z having higher accuracy is selected to predict the numberoferrors. Once the best model is selected, the next step is to calculate the QF value for all input software. We applied model Z (SVR) on the dataset kc1 [2] to predict number of errors in the 145 software. We then applied model Y (LR) to the same dataset in order to validate our selection of model on the basis of accuracy. Summary of the results is shown in table 4 where we observed four types of errors namely Mean Absolute Error (MAE), Relative Absolute Error (RAE), Root Mean Squared Error (RMSE) and Root Relative Squared Error (RRSE). The table shows that the error values for model Z are smaller than the error values for model Y. The lower values of MAE and RAE imply that values predicted by model Z were closer to actual values.

5. DISCUSSION

Currently the proposed model is handling structural and object oriented development paradigms only, but it is extendible to other paradigms. It can also be extended for as many quality factors as there have been models available for. The model is usable at any stage of software life cycle provided the models repository contains models applicable in that phase. Our model caters for the specificity of a component predictor by taking three control inputs (Quality Factor, Software Development Life Cycle Phase, and Software Development Paradigm), which contribute towards selection of a predictor. We are using the product-based existing models unlike Boukhtif et al. [3] who are using classification based models.

Accuracy as selection criterion: Any model can be plugged into the suggested model provided its accuracy does not fall below the minimum threshold $\alpha$. Li et al.[17] have argued that accuracy is not a good criterion for model selection. They have asserted that in context of product testing prioritization, model selection should be done considering the specificity of the predictors. Although this concern applies to our selection criteria as well but we are restricting ourselves to accuracy for the time being.

Generality: Integrated approach to predict software quality by Wagner et al. [28] is a significant contribution since it provides the ways to describe generic quality characteristics and how to adapt to different contexts. Our paper suggests a model which is more general than the integrated approach by Wagner et al. [28] since it does not require each organization to develop a base model before including purpose models. The purpose models are specific models which are derived from the base model in the light of quality goals [28]. Our idea is to have a predictor which requires minimum effort from quality engineer in predicting quality of software. As shown in section 4, our model only needs the software metrics and will do the rest of the work including model selection and QF calculation.

6. CONCLUSIONS

In this paper we have suggested an approach which takes benefits from the existing work. The approach caters for the specific nature of component models and is extendible as well as scalable. Extendible in the sense that it can be used for more than one paradigms as well as for different SDLC phases. Moreover it can have the capability to predict more than one quality factors. A new component model for another quality factor can be added to the generic framework at any time. The only restriction on the inclusion of a component model is that the accuracy of the model can not fall below a threshold $\alpha$ which is set to 65% in this paper. The paper shows working of the model through an example using 145 data instances. The example depicts that the model selected based on its accuracy was better selection in this case. The paper identifies some issues which still need to be addressed. The first issue is whether a software should be provided in the form of metrics or as source code. The suggested approach took software in the form of metrics but faced problems in using existing models as discussed in section 3.2. So taking software in the form of metrics can affect the accuracy of the prediction. In the second case the suggested model first needs to calculate the software metrics of its choice. This might be time consuming but while predicting software quality, accuracy is the most important factor. So time taken to build and use the model can be compromised as long as it guarantees higher accuracy. In future our plan is to explore the tradeoffs between selection of the above mentioned two choices.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Name in dataset</th>
<th>Unified Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coupling Between Objects</td>
<td>COUPLING_BETWEEN_OBJECTS</td>
<td>CRO</td>
</tr>
<tr>
<td>Depth of Inheritance Tree of a class</td>
<td>DEPTH</td>
<td>DIT</td>
</tr>
<tr>
<td>Lack of Cohesion in Methods</td>
<td>LACK_OF_COHESION_OF_METHODS</td>
<td>LCOM</td>
</tr>
<tr>
<td>Number of Children</td>
<td>NUM_OF_CHILDREN</td>
<td>NOC</td>
</tr>
<tr>
<td>Dependence on an descendant</td>
<td>DEP_ON_CHILD</td>
<td>DEP_ON_CHILD</td>
</tr>
<tr>
<td>Count of calls by higher modules</td>
<td>FAN_IN</td>
<td>FAN_IN</td>
</tr>
<tr>
<td>Response For Class</td>
<td>RESPONSE_FOR_CLASS</td>
<td>RFC</td>
</tr>
<tr>
<td>Weighted Methods per Class</td>
<td>WEIGHTED_METHODS_PER_CLASS</td>
<td>WMC</td>
</tr>
</tbody>
</table>

Table 3: Object-Oriented Metrics Used

<table>
<thead>
<tr>
<th>Metric</th>
<th>Model Z (SVR)</th>
<th>Model Y (LR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAE</td>
<td>4.4669</td>
<td>5.5415</td>
</tr>
<tr>
<td>RAE</td>
<td>73.9543%</td>
<td>91.7451%</td>
</tr>
<tr>
<td>RMSE</td>
<td>10.6442</td>
<td>10.5929</td>
</tr>
<tr>
<td>MAE</td>
<td>97.4832%</td>
<td>97.0131%</td>
</tr>
</tbody>
</table>

Table 4: Summary of Results
7. ACKNOWLEDGMENTS
We would like to thank Malik Jahan Khan and Numan Sheikh for their support during this work. We also thank Higher Education Commission (HEC) of Pakistan and Lahore University of Management Sciences (LUMS) for funding this research.

8. REFERENCES


