Design Driven Code Complexity Metrics

Omar Masmali
Department of Computer Science
The University of Texas
El Paso Texas USA
oamasmani@miners.utep.edu

Omar Badreddin
Department of Computer Science
The University of Texas
El Paso Texas USA
obadreddin@utep.edu

Background
Software complexity is an important indicator for software quality and sustainability. Software complexity is a good approximation for software comprehensibility and maintenance. Many complexity measures have been developed by researchers to identify and measure the degree of complexity. These complexity metrics suffer from some fundamental limitations. First, these metrics often ignore the uniqueness of software and operate under the assumption that a single metric will be applicable to all or most software systems. Moreover, these metrics do not evolve over time to appropriately consider the evolving code base size and its indented lifetime. One important goal of software engineering is to deliver software systems that can be sustainably maintained for extended period of time. My research investigates novel code complexity metrics that are derived from software designs. The proposed approach ensures that the derived metrics are uniquely tailored for the software under development and are unique for each software module. Furthermore, these metrics can dynamically evolve throughout the codebase life cycle.

Proposed Approach
The proposed approach is formulated by assigning a complexity rating for each element within the class. These elements include attributes, methods and associations. First, we specified the complexity rate for attribute type and visibility to obtain the first metric which is attribute complexity. Then, we estimate the method complexity metric by appointing a complexity rate for method visibility, return type, and parameters. After that, we assign a complexity rate for the incoming and outgoing associations to get the association complexity metric. Finally, by summing all attributes, methods, and associations complexities, we can estimate the class complexity as shown Table 1.

Table 1: Design Driven Code Complexity Metrics

<table>
<thead>
<tr>
<th>Name</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute Complexity</td>
<td>$Att_{comp} = (V_{Att.} \times CoRate) + (T_{Att.} \times CoRate)$</td>
</tr>
<tr>
<td>Method Complexity</td>
<td>$Method_{comp} = (V_{Meth.} \times CoRate) + (R_{Meth.} \times CoRate) + \left(\sum_{i=1}^{N}(P_{Meth.} \times CoRate)\right)$</td>
</tr>
<tr>
<td>Association Complexity</td>
<td>$Asso_{comp} = \left(\sum_{i=1}^{N}(IN_{As.} \times CoRate)\right) + \left(\sum_{i=1}^{N}(OUT_{As.} \times CoRate)\right)$</td>
</tr>
<tr>
<td>Class Complexity</td>
<td>$Class_{comp} = \left(\sum_{i=1}^{N}Att_{comp}\right) + \left(\sum_{i=1}^{N}Method_{comp}\right) + Asso_{comp}$</td>
</tr>
</tbody>
</table>
**Preliminary results**

After applying the proposed approach on opensource systems, we observed the high correlation between LOC and class complexity. The correlation was between 0.76 and 0.91 with an average 0.85. According to these findings, we can see the strong relationship between class complexity and the lines of code for that class. Moreover, we evaluated the proposed metrics against the nine weyuker’s properties in order to validate the usefulness and correctness of the measures. We found that the proposed measure satisfied by the properties.