An In-Vivo Study of the Cognitive Levels Employed by Programmers During Software Maintenance

Tara Kelly  
Limerick Institute of Technology  
Tara.kelly@staffmail.ul.ie

Jim Buckley  
CSIS Dept. University of Limerick  
Jim.Buckley@lero.ie

Abstract
Several researchers have proposed Bloom’s Taxonomy as a framework within which to study the cognitive levels employed by programmers during software comprehension. But a review of empirical studies in this area illustrates that previous work has nearly exclusively focused on the lower cognitive levels of the taxonomy. However, the taxonomy was initially proposed as a ‘cumulative hierarchy’, where less processing occurred at higher levels. This suggests that the focus of current software comprehension literature is appropriate.

Given that there is mixed empirical evidence for this ‘cumulative hierarchy’ property, this work reports on the cognitive levels employed by 6 programmers, involved in in-vivo software maintenance and comprehension. It suggests that the cumulative hierarchy property is true of these contexts, thus adding legitimacy to the focus of the existing literature. However, it notes that processing at the higher cognitive levels does occur and is associated with specific maintenance sub-tasks. As this processing is effort and skill intensive, there is still a need for researchers to explore these higher cognitive levels.

1. Introduction
Software maintenance is typically the longest phase in a successful software system’s lifecycle, stretching from the point of initial deployment until the system is taken out of service through various release iterations [1]. Hence, research has suggested that maintenance can consume up to 90% of total systems’ costs [2] and that half of this effort can be directly related to understanding the software [3]).

Software comprehension can be defined as “the process in which the programmer uses prior knowledge about programming, and information present in the program, to form a dynamic (and static) model of the program which can then be applied to a task” ([4], pg 14). Closely related to software comprehension is the concept of cognitive levels: the degree to which programmers demonstrate cognitive mastery/dexterity over the software systems they work on. Recently researchers have begun probing this aspect of software comprehension using Bloom’s taxonomy [5], [6], [7], [8]. Bloom’s taxonomy is an established pedagogical framework, which states the cognitive dexterities that students may have over learnt material [9]. It classifies cognitive dexterity into a six-tiered hierarchy where, Bloom states, each level builds on the cognitive skills in the preceding levels and thus demonstrates educational advancement.

Bloom’s taxonomy would seem to be highly relevant for informing on software maintenance. For example, large individual, performance differences have been reported of maintenance programmers in empirical studies [10], but there is little work on explaining these differences. Individual cognitive dexterities with respect to material learnt (the software systems programmers work on) would intuitively seem to explain at least part of these differences, providing that all these cognitive dexterities are relevant in the context of software maintenance. Determining the relevance of all these cognitive dexterities for software maintenance is the core focus of the research reported here.

The next section of the paper discusses Bloom’s taxonomy. Section 3 discusses the empirical, software comprehension literature, illustrating its implicit focus on the lower levels of the taxonomy. In section 4, the empirical study is described, and its results presented. Section 5, discusses these results.

2. Bloom’s Taxonomy
Bloom’s taxonomy [9] consists of 6 levels:

1. **Knowledge** deals with the cognitive ability of recall. An example is when a programmer recalls the correct structure of an “if” statement.
2. **Comprehension** uses knowledge to understand a communication and translate it. A programmer documenting an algorithm shows comprehension.
3. **Application** involves making well-based changes to existing communications. An example would be when a programmer makes a change to code.

4. **Analysis** concentrates on breaking a communication down into its (often implicit) component parts, and identifying their relationships. An example would be where a programmer re-documents a system’s architecture.

5. **Synthesis** refers to the ability to build something new from existing Knowledge, Comprehension and Analysis. A relevant example would be the ability to build a new computer application.

6. **Evaluation** refers to the ability to make judgements about ideas, work, material and solutions. An example would be the ability to critically review design alternatives for a software system, in the light of, possibly, conflicting requirements.

This taxonomy has been translated into 21 languages and has been used world-wide [11]. At one point the taxonomy was the most frequently cited course for educational research [12]. So it can be said that, over the decades, this taxonomy has become the de-facto standard for stating or measuring the cognitive skill which students are to master with respect to their learnt material.

When Bloom initially described this taxonomy, he characterized it as a cumulative hierarchy [9], where processing at lower levels was a pre-requisite for processing at upper levels. Thus, he implied, less processing would occur at higher levels in the taxonomy, suggesting that elevated levels in his taxonomy would be less prevalent.

Over the years, a number of investigations have been carried out into the validity of the taxonomy, with several of these directed specifically at the cumulative hierarchy property [13], [14], [15], [16], [17]. Stoker and Kropp [13] found general support for the cumulative hierarchical nature of the taxonomy, but with the relationship between levels being less emphasized as the hierarchy is ascended. In Madaus et al. [15] and Miller et al. [16] both found that the data was better modeled by a ‘Y’ like structure, the base of the Y being made up of (from the bottom up) Knowledge, Comprehension and Application. Then Analysis was on one arm of the Y and Synthesis and Evaluation (in that order) were on the other arm. But, [15] suggests that the property may have been broken between Analysis and Synthesis, in this instance, because the assessment mechanism differed between the 2 levels (Analysis used multiple choice questions, while Synthesis used open questions). In slightly conflicting work, Hill and McGaw [17] found that the original cumulative hierarchy was a good fit for this data, but only when students’ performance at the Knowledge level was removed from the evaluation.

Apart from this slightly unclear picture of the ‘cumulative hierarchy’ property that these evaluations provide, there are 2 other reasons to believe that its efficacy is an open question:

- All of the evaluations described above were performed on the Stoker and Kropp data-set and so have a narrow basis in data.
- The Stoker and Kropp data-set was obtained from students reading literary texts. The property’s relevance to differing domains remains unclear.

### 3 Software Comprehension Studies

As part of this research, a review of 45 core empirical studies of software comprehension was performed. Each study was assessed for the cognitive levels probed by the researchers. Table 1 summarizes this review and illustrates that the majority of this work probed the lower cognitive levels with relatively few (more recent studies) addressing some of the higher levels. While the lack of Synthesis-level studies can be largely explained by this level’s relevance to new software development, and lesser relevance to maintenance [18], the reasons for limited study of the other 3 elevated levels in the hierarchy is less apparent.

<table>
<thead>
<tr>
<th>Know</th>
<th>Comp</th>
<th>App</th>
<th>Anal</th>
<th>Synth</th>
<th>Eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>34</td>
<td>14</td>
<td>8</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

This emphasis on the lowest 2 levels may be appropriate for maintenance programmers, if processing at the higher levels of the taxonomy is less prevalent in this context. But, as discussed in Section 2, existing research from the educational domain cannot yet provide clear guidance on this ‘cumulative hierarchy’ property. In particular, it is unclear that this characterization can be transferred to the maintenance behaviour of professional programmers.

Table 1 does note that some studies in the software comprehension area have probed more levels of the taxonomy. However, much of this work uses student programmers as participants [6], [7], [8], and employs a verb-table as a means of analyzing the data [5], [8]. This measurement device has been shown to be reductionist and thus somewhat ambiguous in this context [18]. In fact, the only study of this group

---

1 An addendum detailing the individual studies and the cognitive levels they address is available at [22]
which used a (one only) professional programmer [5],
employed the verb-table.

Hence, this work reports on an empirical study, where data generated by 6 professional programmers, during 8 in-vivo maintenance tasks, was analysed using a ‘context-aware analysis schema’ for Bloom’s taxonomy [18]. This allowed the processing at each cognitive level to be quantified giving the strongest evaluation, to date, of the ‘cumulative hierarchy’ property of the taxonomy in the context of professional software maintenance.

4. The Empirical Study

The study addresses the following questions:

• Do professional programmers, involved in software maintenance, carry out processing at each cognitive level of Bloom’s Taxonomy? There is little evidence, as yet, to suggest that programmers process at all levels of the taxonomy.

• If so, is a cumulative hierarchy property associated with their processing? That is, does the amount of processing decrease as the hierarchy is ascended?

4.1 The Empirical Context

The 6 participants in the study were all professional programmers, all male, and ranged in age from 22 to 47 (mean: 35.6 years). They all had at least 3 years industrial programming experience in their current language (mean: 5.6 years) and all except one participant had industrial experience in at least 2 different programming languages (mean: 3 languages).

While one programmer was only employed in his current company for 0.5 years, the mean across participants was 6 years.

The systems these programmers worked on ranged from 1 KLOC to 1.5 MLOC (mean 377 KLOC), and were all either state-sector information systems (tasks 1-4) or commercial software applications (tasks 5-8).

The programmers performed 8 tasks. The 8 tasks they performed were all assigned to them individually by their companies, as part of their daily workload. The first 4 tasks came from the health informatics domain. They were to: export data to another system (tasks 1 and 3 addressed different aspects of this goal), increase the consistency of the current information in the database (task 2) and issue more-targeted renewal forms for disability payments (task 4). Of the other tasks, 3 could be considered functional enhancements to existing applications in the insurance domain (task 5), the travel domain (task 6) and the sales domain (task 8). The last task (task 7) could be considered a configuration enhancement, as it involved adding the ability to select additional printers for an existing application. (Note that participant 1 performed maintenance tasks 1 and 2 and participant 2 performed maintenance tasks 3 and 4. The other participants performed one task each).

4.2 The Empirical Protocol

For a 2-hour period, for each task, programmers were asked to wear a small MP3 recorder and talk as they worked. All but one of the best-practice guidelines of Ericsson and Simons were followed when gathering the think-aloud data from programmers [19]. That is, programmers were asked to say everything that came into their mind as it came into their mind. Programmers were not prompted when they fell silent, as suggested by Ericsson and Simons. This was to improve the ecological validity of the study, by having the researcher leave the room for each 2-hour session. However, overall, programmers fell silent only 6.3% of the time, suggesting that this protocol captured the best approximation of their mental state [20] for the vast majority of the time.

4.3 Data Analysis

A context-aware analysis schema was developed and refined, through a series of pilot studies [18]. In the pilot studies, different samples of the data were classified by 2 independent coders, over several iterations, to develop a coding manual for the schema and to assess its reliability.

This resultant coding manual provides explicit guidance for researchers performing this classification process, to a reasonable level of reliability (Cohen’s Kappa = 0.585) and was structured in accordance with best-practice guidelines [4]. Given the lesser relevance of Synthesis in a software maintenance context, the schema did not provide for the classification of Synthesis level utterances.

4.4 The Results

The complete data-set was then analysed. Table 2 details the number of utterance at each level of the taxonomy for each task. The table illustrates that all developers, in all tasks, worked to some degree, at all 5 assessed levels. This suggests that the taxonomy is relevant for real programmers working on real maintenance tasks.

Evidence for the cumulative hierarchy property was not always as strong in individual cases. However, processing did seem to decrease in tasks 1, 4

---

2 The manual is also available in the addendum [22]
and 6, as the hierarchy was ascended. Indeed, taking all sessions cumulatively this trend is preserved, as can be seen in the bottom rows of Table 2 and Figure 1.

Table 2: Number of utterances at each cognitive level.

<table>
<thead>
<tr>
<th>Task</th>
<th>Know</th>
<th>Comp</th>
<th>Appl</th>
<th>Anal</th>
<th>Eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>324</td>
<td>281</td>
<td>84</td>
<td>43</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>127</td>
<td>200</td>
<td>131</td>
<td>72</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>343</td>
<td>123</td>
<td>180</td>
<td>140</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>206</td>
<td>90</td>
<td>87</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>106</td>
<td>104</td>
<td>59</td>
<td>71</td>
<td>52</td>
</tr>
<tr>
<td>6</td>
<td>148</td>
<td>79</td>
<td>46</td>
<td>38</td>
<td>37</td>
</tr>
<tr>
<td>7</td>
<td>129</td>
<td>40</td>
<td>111</td>
<td>17</td>
<td>45</td>
</tr>
<tr>
<td>8</td>
<td>38</td>
<td>50</td>
<td>57</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>Total</td>
<td>1421</td>
<td>967</td>
<td>755</td>
<td>412</td>
<td>206</td>
</tr>
<tr>
<td>Avg.</td>
<td>178</td>
<td>121</td>
<td>94</td>
<td>52</td>
<td>26</td>
</tr>
</tbody>
</table>

These processing differences between the cognitive levels were found to be statistically significant at the 5% level using the Friedman test (X^2 = 11.75) [21]. The decreasing trend was also found to be strongly significant using a Page L test (p = 0.001, L = 411), supporting the cumulative hierarchy property for this specific domain (where cumulative hierarchy is characterized as decreasing activity as the hierarchy is ascended).

4.5 Validity

The results from this study should be viewed with some caution, given the small sample size, and the self-selecting population. However, it should be noted that this is the most comprehensive data set to-date in the study of cognitive level usage by professional programmers involved in a range of real-world software maintenance contexts.

The schema itself is also a threat to the validity of this study. By its nature it is quite difficult to use and, even though reasonable reliability was achieved between coders, this leaves a certain amount of categorization open to interpretation.

In addition, there was no measure of ‘correctness’ for the utterances that the programmers made. For example, if a programmer commented on the efficiency of the code, then his utterances were coded as Evaluation, even if his comments were incorrect. Indeed, given the scale of the software systems being maintained and our lack of familiarity with these systems, it would have been impossible to determine the correctness of each utterance. However, this element of correctness would seem to report fairly directly on the difficulty of mastering (rather than just processing at) each of the cognitive levels.

5. Discussion

The findings of this study support the assertion that professional programmers work at 5 levels of Bloom’s taxonomy during maintenance. Also, their activity at upper levels does seem to decrease significantly. This suggests that the focus of existing software literature is largely correct.

![Figure 1: Cognitive level processing across all tasks](image)

But there are 2 significant caveats. Because we could not capture information on the correctness of their utterances, we can only comment on the amount of processing programmers did. Hence, it may be that programmers are weaker at higher levels. Indeed, this is likely given Bloom’s assertion that the more elevated the level, the more difficult that cognitive dexterity is to learn [9]. If so, efforts should be made to further study these levels and to facilitate programmers’ acquisition of cognitive mastery with respect to these levels, possibly through the development of new supportive software tools.

A 2nd caveat is that in-depth qualitative analysis found that specific maintenance tasks were associated with specific (elevated) cognitive levels:

- After changes had been made to the code-base there was a rise in Evaluation utterances, as the programmers seemed to mentally assess the quality of their changes.
- In changes which seemed distributed over de-localised pieces of code, Analysis processing increased. For example, in Task 5 where a new test region was created for testing the entire system, 10% of all categorised utterances were at the Analysis level.
- Likewise, elevated levels of Analysis utterances seemed temporally linked to Application. This would suggest that making changes to the code may prompt a programmer to analyze how the change would impact on de-localized elements.
Given these seeming relationships between specific software maintenance sub-tasks and the elevated levels of the hierarchy, it would still be appropriate to study the elevated levels of taxonomy, in the context of software maintenance.

5. Conclusion
This research has shown, through an empirical study, that professional programmers seem to work at all cognitive levels assessed, and that their processing supported the cumulative hierarchy property. However, the empirical study is of a small scale and should be buttressed by additional studies which also probe the correctness of programmers’ processing at each level. In addition, it should be noted that qualitative evidence was found to suggest that specific maintenance subtasks require processing at elevated levels of the hierarchy, making them a valid focus for software comprehension studies in the future.

Acknowledgements
We would like to thank our participants and Lero (supported by Science Foundation Ireland grant number 03/CE2/I303_1).

References