Abstract. In this paper, we focus on the net changes in attributes across versions of OSS and use net class change data (class additions and deletions) as well as refactoring data from a previous study to inform our understanding of how those three systems evolved as they did. While the majority of new attributes were added at levels 1 and 2 of the inheritance, these patterns were not consistent. The research question addresses the evolutionary relationship between classes and attributes as well as the connection between those changes and refactorings. Although some evidence of attributes following patterns conformant with class additions was found, we also identified occurrences of attributes being added unilaterally. A strong correspondence was also found between attribute addition and the refactoring data. Finally, we explore features of a fourth system with seven inheritance levels for similar characteristics.

Keywords. Evolution, OO, attribute, refactoring.

1. Introduction

Software evolution is still an area of software engineering that we know relatively little about. In a recent empirical study of seven Java systems by the authors [14], it was found that approximately 81% of all classes added over the course of the versions studied were added at inheritance level 1 (classes that extend ‘Object’). Only 15% of classes were added to classes at inheritance level 2 and only 4% of classes were added at level 3 and beyond. While we would expect a relatively close correspondence between the addition of the different class features (methods and attributes) in added classes, we cannot discount the possibility that an OO system may show different evolutionary patterns at a higher level (e.g., at class and package) than that at the lower level (i.e., at method and attribute). In this paper, we empirically investigate the evolutionary trends of attributes in three Open-Source Java systems [6, 7]. Inheritance and attribute data was extracted from multiple versions of the 3 systems using the JHawk tool [10]. The research explores the relationship between evolution in attributes, classes and that with refactoring.

2. Related work

In terms of related work, system evolution lies at the heart of the study presented [9, 11, 12]. A study of evolution of an OSS by Capiluppi et al. [1] used the number of folders, files and lines of code to quantify each version of the system. In a further study, Capiluppi and Ramil [2] undertook an empirical analysis of two OSSs (Arla and Mozilla) from an evolutionary perspective. They discovered certain similarities in the evolutionary behaviour of the two systems at a higher level of abstraction. In a study of a large industrial OO system, Cartwright and Shepperd [3] found that inheritance was also used sparingly. In addition, a positive correlation between the Depth of the Inheritance Tree of a Class (DIT) metric of Chidamber and Kemerer (C&K) [5] and the number of user reported problems was found.

3. Study details

The three systems used in this study were chosen from sourceforge.net and were the subject of a refactoring study described in [4].
1. HSQLDB: a Java relational database engine. Comprised 6 versions; it started with 56 classes in version 1 with 358 classes by the final version.
2. JasperReports: a business intelligence and reporting engine. Comprised 12 versions; it started with 818 classes in the version 1, with 1098 classes by the final version.
3. Tyrant: a graphical fantasy adventure game. 45 versions of this system were studied; started with 122 classes in its first version and ended with 273 classes by the final version.

For this study, we used the JHawk [10] tool to extract inheritance and size measures from each version of the three systems: 1) DIT: measures the number of ancestors of a class including ‘Object’ (from which all classes inherit). The DIT metric is that proposed by C&K [5]. The value of DIT for class ‘Object’ at the root of the entire hierarchy as zero; hence, all classes declared at level one implicitly extend only class ‘Object’ 2) Number of Attributes (NOA): measures the number of local variables plus the number of class variables (public, private and protected). The purpose of collecting the DIT was to provide a common basis for comparing net changes in attributes and classes at each level and to allow this relationship to be explored (as well as the possible relationships with refactorings in the systems investigated). The JHawk tool is a general-purpose metrics collection tool capable of collecting a variety of metrics from OO systems. These include C&K metrics as well as general metrics such as lines of code, fan-in, fan-out and other complexity-based metrics.

4. Data analysis

4.1. HSQLDB

The maximum DIT for the HSQLDB system (in any of the versions studied) was 4. Figure 1 shows the net number of attributes added or removed (net changes) from the versions of HSQLDB on an incremental basis. For example, the net number of attributes added at DIT 1 between version 1 and 2 was 377; 46 attributes were added at DIT 2 and only 4 attributes added at DIT 3. It is notable that while 12 classes were added at DIT level 3 throughout the versions studied, only 4 attributes were added in that time. From the same figure, the maximum change of NOA takes place between versions 3 to 4. This trend was also observed in changes of number of methods in [14] suggesting that the system underwent major re-engineering between these two versions. From Figure 1, we see that DIT 1 and 2 is where the vast majority of activity takes place; developers tended to be relatively inactive at deeper levels of the inheritance hierarchy.

![Figure 1. Net changes in NOA (HSQLDB)](image)

The net changes of classes in the 6 versions of HSQLDB are summarized in Table 1. There is a clear trend for classes to be added at shallow levels of the hierarchy and not necessarily in version 1, but between versions 3 and 4. When combined, Figure 1 and Table 1 show that net change in attributes is not always accompanied by a corresponding change in classes.

<table>
<thead>
<tr>
<th>Version</th>
<th>DIT 1</th>
<th>DIT 2</th>
<th>DIT 3</th>
<th>DIT 4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>50</td>
<td>21</td>
<td>3</td>
<td>0</td>
<td>74</td>
</tr>
<tr>
<td>2-3</td>
<td>10</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>3-4</td>
<td>133</td>
<td>34</td>
<td>8</td>
<td>1</td>
<td>176</td>
</tr>
<tr>
<td>4-5</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5-6</td>
<td>33</td>
<td>4</td>
<td>-1</td>
<td>-1</td>
<td>35</td>
</tr>
</tbody>
</table>

4.1.1. HSQLDB refactoring

One feature of the evolution of a system that may help to explain the trend in Figure 1 is that of refactoring [8]. Developers should refactor ‘mercilessly’ [8] and apply various types of refactoring as good practice. However, all empirical evidence to date suggests that only
simple refactorings are undertaken frequently. For example, the study by Counsell et al. [4] found that the majority of refactorings were simple renaming of methods and fields. More ‘complex’ refactorings such as those related to inheritance were found to be applied less frequently. In this paper, we want to explore whether patterns in refactorings follow those of attributes or classes (or neither). In other words, when we add large numbers of each – does refactoring effort increase accordingly? Figure 2 shows the trend in attribute-based refactorings applied to the HSQLDB system in the first four versions. Refactoring data was extracted using an automated tool, details of which were first reported in [4]. (We note that when the tool was run, version 4 was the latest available version of HSQLDB.) Fifteen refactorings were extracted by the tool including:

1) Move Field. ‘A field is, or will be, used by another class more than the class in which it is defined. Create a new field in the target class, and change all its users’ [8].
2) Pull Up Field. ‘Two subclasses have the same field. Move the field to the superclass’ [8].
3) Push Down Field: ‘A field is used only by some subclasses. Move the field to those subclasses’ [8].
4) Rename Field: this refactoring is applied to make the meaning of a field clearer. It is also often undertaken after a field has been moved or pulled up/pushed down to reflect its new role [8].

It is noticeable from Figure 2 that the single ‘peak’ of refactorings (at version 3) occurred at the same time as the single ‘peak’ of net additions of attributes to the HSQLDB system shown in Figure 1. The highest number of refactorings was for the Rename Field and Move Field refactorings, suggesting (when also considering Figure 1) that classes were not necessarily ‘pulled up’ or ‘pushed down’ the inheritance hierarchy. This further implies that, in keeping with the trends described in [14], systems evolved through addition of new classes at DIT level 1 and 2 and not necessarily through the manipulation of the inheritance hierarchy. We also note a coincidence of peaks of net changes in attributes (Figure 1) and Rename Field refactorings.

4.2. JasperReports

The maximum DIT for the JasperReports system in any of the versions was 5. Figure 3 shows the net change in attributes through the versions studied. Between version 10 and 11, there was a movement of attributes from DIT 2 to DIT 1. From Figure 3, we again see a strong tendency for NOA at DIT level 1 to fluctuate.

It is noticeable from Figure 2 that the single ‘peak’ of refactorings (at version 3) occurred at the same time as the single ‘peak’ of net additions of attributes to the HSQLDB system shown in Figure 1. The highest number of refactorings was for the Rename Field and Move Field refactorings, suggesting (when also considering Figure 1) that classes were not necessarily ‘pulled up’ or ‘pushed down’ the inheritance hierarchy. This further implies that, in keeping with the trends described in [14], systems evolved through addition of new classes at DIT level 1 and 2 and not necessarily through the manipulation of the inheritance hierarchy. We also note a coincidence of peaks of net changes in attributes (Figure 1) and Rename Field refactorings.
The maximum change (58) occurred between versions 6-7. Table 2 shows the number of net changes of classes at the different levels of DIT.

In keeping with the HSQLDB system, there appears to be a lack of addition of classes in earlier versions of the system. One plausible theory for that lack of addition of classes is that there is a time ‘lag’ between when a system is first released and the signs of decay. That decay is accompanied by a concerted re-engineering effort.

<table>
<thead>
<tr>
<th>Version</th>
<th>DIT 1</th>
<th>DIT 2</th>
<th>DIT 3</th>
<th>DIT 4</th>
<th>DIT 5</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>2-3</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>3-4</td>
<td>36</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>43</td>
</tr>
<tr>
<td>4-5</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>5-6</td>
<td>11</td>
<td>-6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>6-7</td>
<td>35</td>
<td>18</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>58</td>
</tr>
<tr>
<td>7-8</td>
<td>43</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47</td>
</tr>
<tr>
<td>8-9</td>
<td>33</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>37</td>
</tr>
<tr>
<td>9-10</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>11</td>
</tr>
<tr>
<td>10-11</td>
<td>-1</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>11-12</td>
<td>33</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>44</td>
</tr>
</tbody>
</table>

4.2.1. JasperReports refactoring

The same four refactorings for the first three versions of JasperReports are shown in Figure 5. (We note that when the refactoring tool was run, version 3 was the latest available version for JasperReports.) Only two of the four refactorings are non-zero. No evidence of either of the ‘Pull Up Field’ or ‘Push Down Field’ refactorings were found in any of the versions of this system.

The fact that there was also peak of added attributes coinciding with that in Figure 4 supports the hypothesis that significant effort was applied to the system at this point and that refactoring effort coincided with that effort. Again, this gives us an insight into the question as to whether developers do refactor and ‘when’ they refactor.

4.3. Tyrant

Figure 6 shows the net changes of attributes in Tyrant. Since the number of classes at DIT 4 and 5 falls to zero in version 5, we excluded attributes at DIT 4 and 5 from the figure. We see that the net changes in number of attributes at DIT level 1 are predominantly positive. In version 4 of Tyrant, the number of attributes at DIT 2 and 3 falls with a corresponding increase in number of attributes at DIT 1. The most notable feature for this system is the fact that after version 4, where maximum DIT drops from 5 to 3, the system stabilizes and thereafter no change in number of classes, methods or attributes is made to the system for the duration of a number of versions.

![Figure 6. Net change in attributes (Tyrant)](image)

In Tyrant, the total number of removed attributes at DIT level 1, 2 and 3 are -14, -169 and -197, respectively throughout the entire 45 versions of the system. This implies that the number of added attributes at DIT level 1 were significantly higher than the number of removed attributes at this level. In contrast, the number of added attributes at DIT level 2 and 3 tend to be significantly lower than the number of removed attributes in these levels. This latter result again implies that while new attributes are added at DIT level 1, some attributes may have been moved from DIT levels 2 and 3 to level 1 as a result of refactoring (possibly using Pull Up Field and Pull
Up Method). Figure 7 shows the net changes of classes in Tyrant. The system stabilizes after version 4 where significant change is made to the system. We believe the system underwent re-engineering activity and, as a result, system ‘stability’ was improved.

Furthermore, the maximum change in number of classes in Tyrant (+29) occurred between versions 4-5. In terms of changes of NOA the maximum (-848) occurred between the same versions (4-5). Again, evolution at lower granularity shows an opposite trend in systems’ evolution.

4.3.1. Tyrant refactoring

Figure 8 shows the same four refactorings for Tyrant (as was presented for HSQLDB and JasperReports). In keeping with the other two systems, few refactorings were undertaken for this system across the versions studied. (When the refactoring tool was run, version 9 was the latest available version of Tyrant.) We do see some evidence of renaming of attributes at later versions of the system, but this is related to moving of existing class features than addition of new ones.

4.5. Deeper levels of inheritance

Figure 9 shows the net changes of attributes in SwingWT (max DIT 7, 22 versions and 522 classes). A peak in version 9 of the system was also observed in changes of methods in [13] and classes in [14]. In version 9 of SwingWT, the number of attributes increases by 645, accompanied by 1929 methods and 160 classes.

It is interesting that the change in number of classes in SwingWT is always positive, suggesting growth in the system in every consecutive version. The transition between version 9 and 10 is the point where the maximum classes (160) were added to the system. Analyzing a system at a lower level of granularity (method and attribute) can often provide a more detailed insight into the evolution of the system than that of analyzing the same system at a higher level of granularity. In version 20 of SwingWT, we observed that the total NOA at DIT 2 was 476 and the total number of classes in the same level was 87. In version 21,
the total NOA at DIT 2 dropped to 276 and the total number of classes at the same level stayed constant at 87. This was also found between versions 14 and 15 of SwingWT where the number of classes at DIT 1 increased from 296 to 329 with a corresponding drop of 116 attributes at the same level. Between versions 15 to 16 of SwingWT the total number of classes at DIT 3 dropped from 25 to 15 with a corresponding rise in number of attributes of 65.

4. Conclusions

In this paper we described the evolution of three open-source systems at the attribute level. Results suggest that looking at just ‘class’ evolution is a short-sighted policy. We need to consider features inside the classes (i.e., attributes). The results may thus be of interest to developers because trends in system evolution at lower levels of granularity bring to light more detailed evolutionary patterns not possible with analysis at a class level; it can also help developers predict changes in future versions. In particular, if we look at just coarse-grained additions of classes, then this ignores a large amount of evolutionary behaviour at the attribute level. Finally, the study shows where remedial effort can be applied by developers, since we can identify behaviour inside classes that may need to be re-engineered.

5. References