System Evolution at the Attribute Level: an Empirical Study of Three Java OSS and their Refactorings

S. Counsell and E. Nasseri
School of information Systems, Computing and Mathematics, Brunel University, Uxbridge, United Kingdom

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 (Nasseri et al. 2008), 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 (Dinh-Trong and Bieman 2004; Ferenc et al. 2004). Inheritance and attribute data was extracted from multiple versions of the 3 systems using the JHawk tool (JHawk 2009). 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 (Girba et al. 2005; Kemerer and Slaughter 1999; Lehman 1980). A study of evolution of an OSS by Capiluppi et al. (Capiluppi et al. 2004) used the number of folders, files and lines of code to quantify each version of the system. In a further study, Capiluppi and Ramil (Capiluppi and Ramil 2004) 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 (Cartwright and Shepperd 2000) 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) (Chidamber and Kemerer 1994) and the number of user reported problems were 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 Counsell et al. (Counsell et al. 2006).

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 tool to extract inheritance and size measures from each version of the three systems: 1) Depth in the Inheritance Tree of a class (DIT): measures the number of ancestors of a class including 'Object' (from which all classes inherit). The DIT metric is that proposed by Chidamber and Kemerer (C&K) (Chidamber and Kemerer 1994). 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 versions 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 Nasseri et al. (Nasseri et al. 2008) 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.

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.

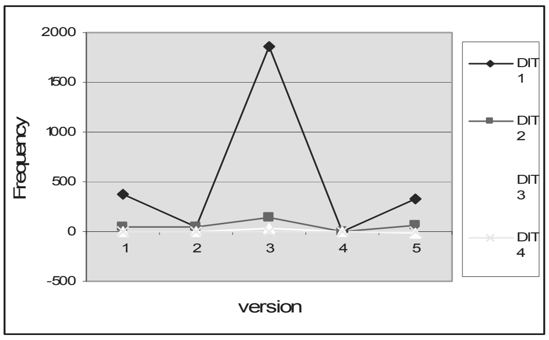

Figure 1. Net changes in NOA (HSQLDB).

Version	DIT 1	DIT 2	DIT 3	DIT 4	Total
1-2	50	21	3	0	74
2-3	10	6	1	0	17
3-4	133	34	8	1	176
4-5	−1	1	0	0	0
5-6	33	4	−1	−1	35

Table 1. Net class additions (HSQLDB).

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 (Fowler 1999). Developers should refactor 'mercilessly' 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. (Counsell et al. 2006) 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 (Counsell et al. 2006). (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' (Fowler 1999).
2) Pull Up Field. 'Two subclasses have the same field. Move the field to the superclass' (Fowler 1999).
3) Push Down Field: 'A field is used only by some subclasses. Move the field to those subclasses' (Fowler 1999).
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 (Fowler 1999).

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 Nasseri et al. (Nasseri et al. 2008), 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.

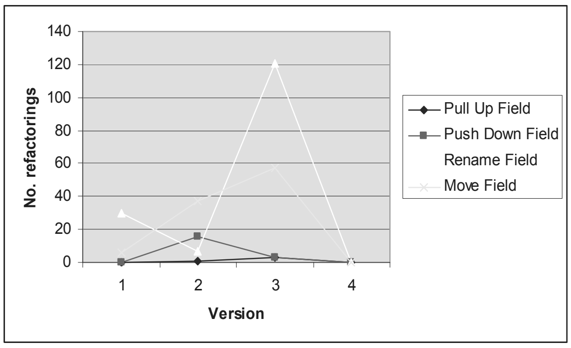

Figure 2. Refactorings for HSQLDB.

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.

Figure 4 shows the net changes of classes at all DIT levels in all versions of JasperReports and shows changes in every version. 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.

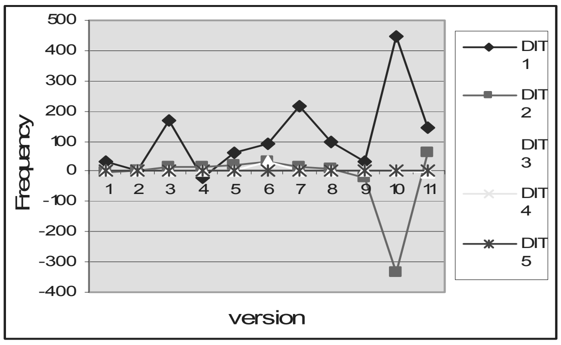

Figure 3. Net changes in NOA (JasperReports).

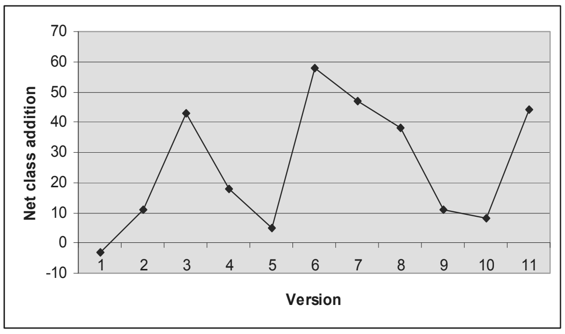

Figure 4. Net class additions (JasperReports).

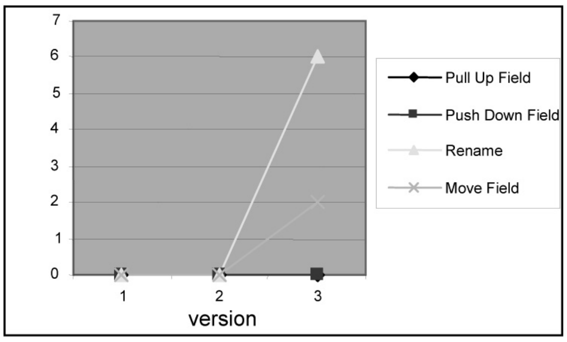

Figure 5. Refactorings for JasperReports.

Version	DIT 1	DIT 2	DIT 3	DIT 4	DIT 5	Total
1-2	−2	−1	0	0	0	−3
2-3	3	6	2	0	0	11
3-4	36	5	2	0	0	43
4-5	9	9	0	0	0	18
5-6	11	−6	0	0	0	5
6-7	35	18	5	0	0	58
7-8	43	4	0	0	0	47
8-9	33	3	1	0	0	37
9-10	5	6	1	0	−1	11
10-11	−1	9	0	0	0	8
11-12	33	9	2	0	0	44

Table 2. Class additions (JasperReports).

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.

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.

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

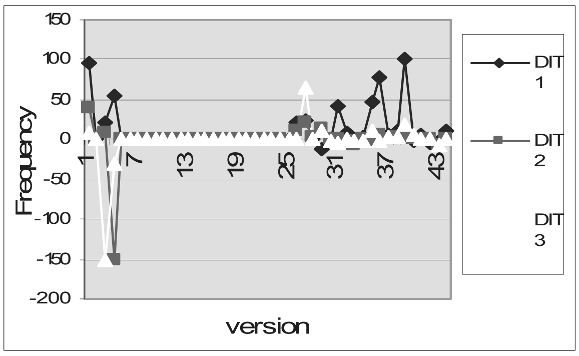

Figure 6. Net change in attributes (Tyrant).

of the system. This implies that the number of added attributes at DIT level 1 was 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.

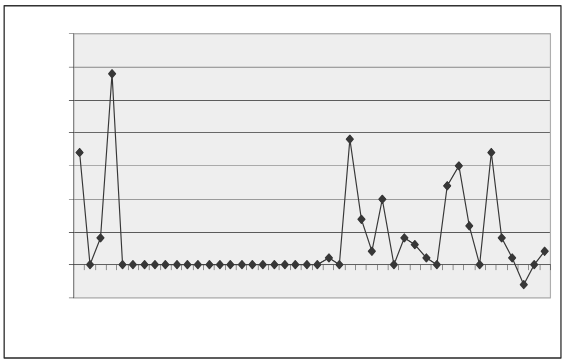

Figure 7. Net change in classes (Tyrant).

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 movement of existing class features rather than addition of new ones.

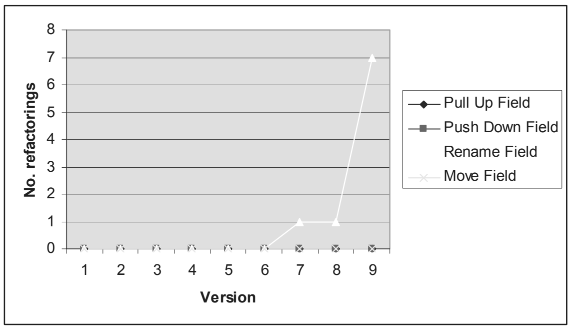

Figure 8. Refactorings for Tyrant.

4.4. 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 Nasseri et al. (Nasseri et al. 2009) and classes in Nasseri et al. (Nasseri et al. 2008). In version 9 of SwingWT, the number of attributes increases by 645, accompanied by 1929 methods and 160 classes.

Figure 10 shows the net change in classes across the versions of SwingWT.

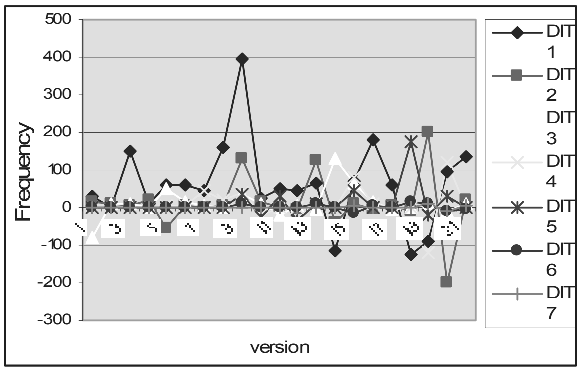

Figure 9. Net change in NOA (SwingWT).

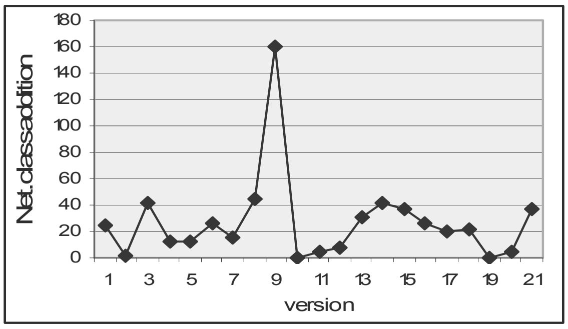

Figure 10. Net change in classes (SwingWT).

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 versions 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.

5. Conclusion

This paper reported on the design and testing of two throwaway mobile chill applications. By their very nature these applications must be quick to design and build and as a result the designers were given the same time to design each application. The interface produced by B was quicker and easier to learn and use, and produced less errors. Users were able to complete each given task in an average of 18 seconds. This time is suitably low for a mobile device as researchers [1] have found that once a task takes more than 30 seconds on a mobile device, users start to become frustrated. We can conclude therefore that the time factor is not likely to discourage people from using either of the applications.

We have therefore gone some way to proving our hypothesis set out in our introduction i.e. that this application worked better, did not fail as often and was more user friendly because one of the designers was given HCI guidelines to follow and implement.

There are of course the usual caveats to attach to our findings e.g. the small evaluation conducted, the inherent talent of a designer and the type of mobile application itself. However, we still believe that, were the experiment to be repeated, we would anticipate seeing very similar results. The reason for this is that we focused on the tasks to be completed and assessed this rather than the aesthetics of the design.

References

[1] A. Capiluppi, M. Morisio, J. Ramil, Structural evolution of an open source system: A case study. Presented in *Proceedings of the International Workshop on Program Comprehension*, Bari, Italy, 2004.

[2] A. Capiluppi, J. Ramil, Studying the evolution of open source systems at different levels of granularity: Two case studies. Presented in *Proceedings of the Workshop on Principles of Software Evolution*, Kyoto, Japan, 2004.

[3] M. Cartwright, M. Shepperd, An empirical investigation of an OO system. *IEEE Transactions on Software Engineering 26(8)*, pp. 786–796, 2000.

[4] S. Counsell, Y. Hassoun, G. Loizou, R. Najjar, Common refactorings, a dependency graph and some code Smells: An empirical study of Java OSS. Presented in *Proceedings of the International Symposium on Empirical Software Engineering*, Rio de Janeiro, Brazil, 2006.

[5] S. R. Chidamber, C. F. Kemerer, A metrics suite for object-oriented design, *IEEE Transactions on Software Engineering 20(6)*, pp. 467–493, 1994.

[6] T. Dinh-Trong, J. Bieman Oss, Development: A case study of FreeBSD. Presented in *Proceedings of the IEEE International Symposium on Software Metrics*, Chicago, USA, 2004.

[7] F. Ferenc, I. Siket, T. Gyimothy, Extracting facts from open source software. Presented in *Proceedings of the International Conference on Software Maintenance*, Chicago, USA, 2004.

[8] M. Fowler, Refactoring: Improving the design of existing code. Addison-Wesley, NJ, USA, 1999.

[9] T. Girba, M. Lanza, S. Ducasse, Characterizing the evolution of class hierarchies, Presented in *Proceedings of the European Conference on Software Maintenance and Reengineering*, Manchester, UK, 2005.

[10] Jhawk, Software Metrics Collection Tool. `www.virtualmachinery.com/jhawkprod.html`, [01/01/09].

[11] C. F. Kemerer, S. Slaughter An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering*, 25(4), pp. 493–509, 1999.

[12] M. Lehman, On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software 1*, pp. 213–221, 1980.

[13] E. NASSERI, S. COUNSELL, An empirical study of Java system evolution at the method level. Presented in *Proceedings of the IEEE International Conference on Software Engineering, Research, Management and Applications*, Hainan Island, China, 2009.

[14] E. NASSERI, S. COUNSELL, M. SHEPPERD, An empirical study of evolution of inheritance in Java OSS. Presented in *Proceedings of the Australian Software Engineering Conference*, Perth, Australia, 2008.

Received: June, 2009
Accepted: April, 2010

Contact addresses:
Steve Counsell
School of information Systems,
Computing and Mathematics
Brunel University, Uxbridge
Middlesex, UB8 3PH
e-mail: steve.counsell@brunel.ac.uk

Emal Nasseri
School of information Systems, Computing and Mathematics
Brunel University, Uxbridge
Middlesex, UB8 3PH, UK
e-mail: emal.nasseri@brunel.ac.uk

STEVE COUNSELL is currently a Senior Lecturer in the Department of Computing and Information Systems at Brunel University. His research interests are in software engineering and, in particular, the empirical study of software engineering facets such has refactoring and re-engineering. He also has a strong interest in software metrics and in data quality. He is also investigator on a two currently running funded software engineering research projects, both of which have strong industrial ties.

EMAL NASSERI is currently a Research Associate at the University of Wolverhampton and interfaces with industry on a Knowledge Transfer Partnership. Emal received his PhD in 2009 from Brunel University exploring facets of Java inheritance hierarchies. His other research interests focus on the use of Java open-source software and the collection of data from these systems.