From Indentation Shapes to Code Structures

Abram Hindle, Michael W. Godfrey, and Richard C. Holt
Software Architecture Group (SWAG)
School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
{ahindle,migod,holt}@cs.uwaterloo.ca

Abstract

In a previous study, we showed that indentation was regular across multiple languages and the variance in the level of indentation of a block of revised code is correlated with metrics such as McCabe Cyclomatic complexity. Building on that work the current paper investigates the relationship between the "shape" of the indentation of the revised code block (the "revision") and the corresponding syntactic structure of the code. We annotated revisions matching these three indentation shapes: "flat" (all lines are equally indented), "slash" (indentation becomes increasingly deep), or "bubble" (indentation increases and then decreases). We then classified the code structure as one of: function definition, loop, expression, comment, etc. We studied thousands of revisions, coming from over 200 software projects, written in a variety of languages. Our study indicates that indentation shape correlates positively with code structure; that is, certain shapes typically correspond to certain code structures. For example, flat shapes commonly correspond to comments while bubble shapes commonly correspond to conditionals and function definitions. These results can form the basis of a tool framework that can analyze code in a language independent way to support browsing targeted to viewing particular code structures such as conditionals or comments.

1. Introduction

Software developers often track the recent history of a project by tracing through the trail of revisions that have been committed since the last baseline. This task is time consuming and tedious, as there are often a large number of revisions to sift through, and only a small percentage of them are likely to be relevant to the task at hand. Additionally, revisions themselves can be difficult artifacts to work with: if one wants to fully understand the design change implied by the revision, the revision must be merged back into the source code, and the code then parsed and analyzed. This is not very convenient.

A strong motivation for our work is to help developers better understand the nature of revisions through the use of simple analysis tools whose effectiveness has been established by empirical study. In this paper, we provide a characterization of the underlying distributions of code structures represented by various indentation shapes and the variance of indentation depth. Using this knowledge a developer could write tools to sift through revisions quickly and pick out revisions that contained changes to comments or conditionals based on the shape of the indentation depicted in the revision.

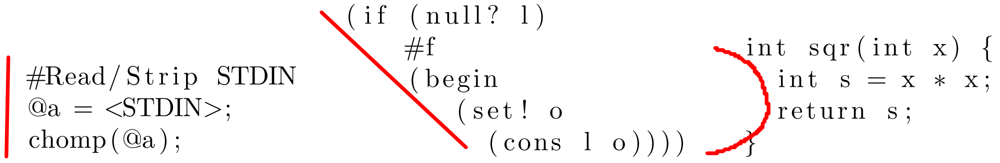
Our contributions consist of:

- the concept of indentation shapes;
- a characterization of the structure of the code that is represented by different shapes, lengths, and magnitudes of variance of indentation;
- a case study of revisions, annotated by their code structures.

We measure revisions (diff-chunks of the UNIX diffs of CVS revisions) by their revision length (lines of code per revision) and their variance of indentation (variance of the indentation of each line in a diff-chunk). We manually annotate each revision with a tag (annotation) that indicates the kind of code structure the revision is an instance of (loop, comment, etc).

Our indentation shapes are simple spatial patterns found in the indentation of revisions: in a *flat* revision the depth of indentation is constant for all lines; in a *slash* revision the indentation level progressively increases; and in a *bubble* revision the indentation increases and then decreases. We chose those shapes as they are simple to visualize and identify in code, and because our preliminary studies found that

978-0-7695-3353-7/08 $25.00 © 2008 IEEE
DOI 10.1109/SCAM.2008.31



```
#Read/Strip STDIN
@a = <STDIN>;
chomp(@a);
```

(a) Flat Indentation

```
(if (null? l)
    #f
    (begin
       (set! o
         (cons l o))))
```

(b) Slash Indentation

```
int sqr(int x) {
  int s = x * x;
  return s;
}
```

(c) Bubble Indentation

Figure 1. Examples of Flat, Slash and Bubble indentation shapes

they were common. An example of the 3 indentation shapes can be found in Figure 1.

In this study we sampled many revisions from many projects; we selected those revisions that matched our three indentation shapes and manually tagged these revisions with annotations that indicate the code structures within the revisions. We then analyzed these revisions and compared them against a control sample of revisions. We observed that the shape of a revision correlates with certain annotations like conditionals or comments.

1.1. Previous Work

Indentation is used in multiple languages as integral part of good programming style. Indentation is often promoted as helping program readability [13] and for emphasizing and defining structure in code [14]. Indentation measurements can be used to trace a program's control flow [10] and has been shown to aide program comprehension [11]. Indentation can be in software visualization, for instance, DrScheme [4] provides a birds eye view of code by showing whitespace and text characters with coloured pixels. Aspects of indentation have been measured: poor indentation as a code quality metric [6]; comparisons of indentation characters to the non-indentation characters [1]; horizontal spacing of source code [3]; and to indicate authorship in plagiarism detection [2]. Others have correlated complexity, nesting and modularity [7, 12], but did not specifically address indentation.

This work builds up on our previous work where we compared indentation measurements to McCabe Cyclometric Complexity (MCC) and Halstead complexity when applied to source code revisions [9]. This complexity comparison worked ties in with other work which discussed how lines of code (LOC) correlate with various metrics [8]. These revision metrics are essentially modification-aware change metrics as discussed by German et al.[5].

1.2. Motivation

Our motivation is to analyze revisions and patches using their indentation. We have previously shown that measuring indentation is almost as cheap and easy as measuring lines of code [9]. We also showed that indentation measures such as the sum indentation, the variance, and the standard deviation of indentation were correlated with complexity metrics.

We feel that indentation measurements are worthwhile to study and use because indentation gives structural hints about the code. It typically implies the context of the revision and how deeply nested the code truly is. It can be efficiently parsed and measured. It does not need tokenization, but it retains the nesting depth that tokenization often throws away.

One of our inspirations for investigating indentation shapes is a code browser in DrScheme [4], it has a small window where text is depicted as coloured pixels and whitespace as white pixels. Just from the indentation of text pixels in the tiny image one can see the structure of the file. It is obvious where functions and deep blocks were. Based on these observations we moved forward to investigate indentation shapes.

Indentation measurements require very little knowledge of the source code they are extracted from. In many cases you do not need to know the language or dialect (e.g., C++ and QT's moc, a preprocessor of C++ for the QT library) of the code itself.

This paper extends our previous work on indentation metrics and measurements [9] to include consideration of indentation shapes. Indentation shapes have the potential to support searches for certain kinds of code. We felt that complexity metrics were not enough, because sometimes one wants to look for comments, sometimes one wants to look for code that matched a certain description.

2. Methodology

We used the following methodology:

- A list of most active and most downloaded projects from SourceForge was retrieved.
- The CVS repositories of these projects were mirrored locally.
- Individual revisions were extracted from each file.
- The indentation of the revisions was analyzed.
- Histograms of frequency of each indentation depth were created and analyzed.

- Indentation shapes were created and defined.
- Revisions were sampled that matched our chosen indentation shapes.
- The sampled revisions where analyzed and annotated.
- The annotations were aggregated and the results analyzed.

We selected revisions of our three shapes (flat, slash, bubble) from 479 different source files (uniformly randomly sampled). The source files consisted of 84 C and .C files, 65 C++, 138 .h, 118 .Java, 51 PHP, 10 Perl and 13 python files. The number of revisions that matched our shapes was more than 5660. Our *control* set was not sampled directly from these files, a new subset of source files was chosen from which we randomly selected and annotated 1001 revisions. This set acts as a control to compare against these shape sets. The control set represents a sample, while the three indentation shape sets represent a population of a sample.

Per each revision, per each continuous indentation chunk, we measured the indentation metrics (length, sum of indentation, standard deviation and variance of indentation as described in section 2.3). Then for those revisions we sampled, we manually annotated them as described in section 2.2. We then analyzed our measurements and discussed the results in section 4.

In the following section we discuss the questions (section 2.1) we wished to answer in this case study and what annotations we tagged the revisions with (section 2.2). Figures 3 to 4 illustrate the proportions, frequencies, and distributions of the annotations of the indentation shapes and control set.

2.1. Questions

Questions we hope to answer in this paper:

- What kind of indentation correlates with function definition?
- What kinds of code correlate with zero variance indentation?
- What kinds of code correlate with non-zero variance indentation?

2.2. Annotations

We created a list of annotations to label the underlying code structures observed in the revisions we were investigating. We manually tagged each revision with our annotations: comments, type declarations, assignments, conditionals, function calls, data, function definitions, macros, loops, conditional macros such as `#ifdef`s, shape matching errors, assertion and exception handling, return values, concurrency and expressions. Each revision was tagged with an annotation that described the main body of the code, usually the top most scope (e.g., a functional call inside of an assignment is tagged as an assignment). The annotations that we used include:

Comments: often pose a difficulty to recognize; if the programming language uses special start/stop tokens to delineate comments (e.g., "`/*`" and "`*/`" in C), it can be difficult to automatically identify revisions to comments as such if the start/stop tokens do not also appear in the revision.

Type Declaration: refers to code that defines types such as structures, classes, or even just storage/name binding such as locals, attributes, slots or globals.

Assignment: refers to assignment statements, where an identifier is set to a new value, typically using the = or := operator.

Conditional: refers to code that is a conditional. Conditionals are structures such as if-blocks, switch statements, unless blocks, etc.

Function Call: refers to code that makes method, procedure or function calls.

Data: Data refers to code that represents constants, initializations, or magic numbers. These data revisions often occur when constants or initializations are listed or stored. Often these data revisions represent long lists of defines and values used to initialize constants and arrays.

Function Definition: refers to code that defines a function or a method. Often these revisions have a function definition in them whether it is the head of a function or just the function's prototype.

Macro: usually refers to preprocessor commands that create macro functions or even defines. Macros do not refer to macro-conditionals like `#ifdef`, although `#include` is considered a macro too. Macros would also refer to some of the extra syntax or keywords added by libraries such as QT.

Loop: refers to revisions to while loops, goto loops, obvious recursive loops, for loops, do loops, foreach, iterators, etc.

Conditional Macro: refers to preprocessor macros such as `#ifdef` as well as the block it encloses.

Anomaly: refers to code that was mistakenly selected as a shape (flat , slash, or bubble). This might occur due to hanging indentation on a new line below that revision.

Exception: refers to code that handles exceptions in Java or C++, or performs error handling in other languages.

Return: refers to code which simply returns an expression in a function.

Concurrency: refers to code which deals primarily with concurrency primitives, locking or critical sections.

Expression: refers to code that produces a value. Its importance varies depending on the language.

2.3. Extraction and Measurement

In this study, we used the same data-set as in our previous study [9]. For each change recorded in CVS to the files in our sample, we analyzed the new and revised code of the diff. These files were C, C++, Java, Perl, PHP, and Python files. If one revision was not continuous, we would analyze the revised code blocks (these sub-parts of a diff are called diff-chunks). In this paper we consider these diff-chunks to be revisions. Our initial dataset consisted of about 13 million revisions; we evaluated only the resulting revised code (the new code). Of those, we selected the revisions of 479 files, and we annotated those (described in section 2.2). We did not measure the initial commits because they would skew the results as these are often full files that are imported, because there was no previous file to revise.

In this study we used raw indentation. We consider raw indentation to be the actual preceding white space on each line of the new code in a revision. For example, if a line consisted of "`␣␣def sqr`", where ␣ was a leading space, we would say it has 2 units of raw indentation. We showed in our indentation and complexity study [9] that raw indentation was consistent across multiple languages.

We had 51GB of CVS repositories and it took about 3 days of run-time to measure the indentation and complexity of each revision of every repository on an Intel Pentium IV; this resulted in 13 million revisions (diff-chunks, not CVS revisions). We subselected 479 files from these repositories and analyzed the revisions of those files. Of these revisions we manually annotated over 6660 of the revisions (revisions described in section 2).

The distribution of the length of revisions is similar to an exponential distribution with most revisions being comprised of only a single line of code. The distribution of measurements such as standard deviation of indentation or variance of indentation is similar; they follow an exponential/power law like distribution. Revision length is somewhat linearly correlated with variance, standard deviation and summation of indentation. This means that variant indentation shapes like slashes and bubbles are not very common.

3. Indentation Shapes

Indentation shapes describe certain patterns of indentation extracted from revision's code. Indentation shapes are especially relevant to revisions because they contain less lines than the actual source files. Indentation shapes are supposed to relate to shapes observed in the curves formed by the indentation of source code.

We hypothesize that indentation shapes give some indication of the structure of the underlying code. This is important if you want to search through revisions that match a certain criterion, you might want to first find candidate revisions based on a lightweight criteria such as an indentation shape and save the heavier criteria for later.

In this study we defined three indentation shapes, each shape matched revisions of at least two lines long. The shapes illustrated in Figure 1 were our initial idea of shapes to look for, they are not necessarily representative of the majority of shapes found with in a repository:

- Flat revisions are revisions where there is no change in indentation.
- Slash revisions are revisions where the indentation progressively increases (or stays the same) without ever decreasing. Examples include expressions which are long and are split up over multiple lines or small if blocks.
- Bubble revisions are revisions where the indentation increases and then decreases like the curve of a bubble. This would include simple if blocks, for loops, while loops and short function definitions.

We also studied a general sample of revisions which we refer to and use as a *control* set. We use the control set to describe how the variance of indentation relates to different kinds of underlying code.

3.1. Flat

Flat revisions are revisions where there is no change in indentation. This could be visualized as a line of elements with the same indentation. We expected flat revisions to be common and that they would correspond to comments, assignment statements and function calls. An example flat revision is depicted in figure 1(a).

More formally, a revision of N lines ($N \geq 2$) is said to be flat if $\forall i : 1..N \bullet I_i = k$ for some constant $k \geq 0$ where I_i is the indentation of line i.

Flat revisions were the most common shape, we annotated 3319 flat revisions. The most common revisions were (in descending order) comments, followed by assignments, type definitions, data and function calls. Flat shapes were least likely to be conditionals, loops, exceptions, function definitions, or plain expressions.

3.2. Slash

Slash revisions are revisions where the indentation progressively increases (or stays the same) across the revision. Example of slashes would include if blocks with only statement inside the block, or long function calls which are wrapped. An example slash revision is depicted in figure 1(b).

Slash revisions can be described as revisions of N lines (where $N >= 2$), where I_i represents the indentation of line i, $\forall i : 2..N \bullet I_i \geq I_{i-1}$, and $I_1 < I_N$.

1552 slash revisions were found among our initial sampling. Conditionals, type declaration, function implementation, function calls, assignment and comments were the most common revisions, although conditionals were the most common. Slash revisions were more likely to be type declarations than function definitions while bubble revisions were more likely to be function definitions and assignment statements. Slash revisions were unlikely to be data, macro, or conditional macro revisions.

3.3. Bubble

Bubble revisions represent code which has a bubble-like shape, that is their indentation grows then shrinks after it reaches a peak. A bubble is not a backwards slash. The last line of a bubble revision must be indented at least as much as the first line of the revision. An example bubble revision is depicted in figure 1(c).

Formally, a revision of N lines, where $N \geq 3$ and where I_i is the indentation of line i, is said to be a bubble revision if there exists a peak k where $\forall i : 2..k\ I_{i-1} \leq I_i \leq I_k$ and $\forall i : (k+1)..N \bullet I_k > I_{i-1} \geq I_i$ and $I_1 \leq I_n$. Thus indentation depth increases up till line k, then after, it decreases. The last line has the same or greater indentation than the first line.

805 different revisions matched this shape and were categorized. Most of the bubble revisions were conditionals, function implementations and assignments. Less popular annotations were type definitions, conditional macros, function calls and comments. Bubble revisions usually were not data or macro revisions.

3.4. Control

The control revisions were a random sampling of revisions meant to represent a typical sample of revisions, that is the revisions that the indentation shapes might not represent.

We annotated 1001 control revisions. These revisions were sampled by first choosing random files to sample, then sampling revisions from these files (effectively sampling from 13 million revisions, not the 479 subselected files). The distribution of revision length was nearly exponential, so most revisions were only 1 line long.

Most of these revisions were comments, type declarations, assignments and function definitions. It seemed like static typing and type declaration took up most of the revisions. Control revisions were not usually exception, return, concurrency or conditional revisions.

4. Discussion

We will break down the analysis of the annotated revisions by properties such as revision length (number of lines in a revision) (section 4.1), total shapes (section 4.2), and quartiles of variance of indentation of revisions (section 4.3). Then we shall address the questions we posed in section 2.1 and address tool integration (section 4.5).

4.1. Revision Length

Figure 3 depicts the distribution of annotations broken down by lengths among the 3 shapes, while figures 2(a) and 2(b) show the same information but for the control sample. The lengths are broken down into bins from 0 lines (e.g., code removal) to 10 or more lines (the last bucket is for revisions 10 lines or greater in length).

Most revisions were short, as 1 line was the most common revision size. For our random sampling (control) of revisions there were many revisions over 10 lines in length (see Figures 2(a) and 2(b)). The most common code structures annotated were comments, type declarations, function definitions and macros.

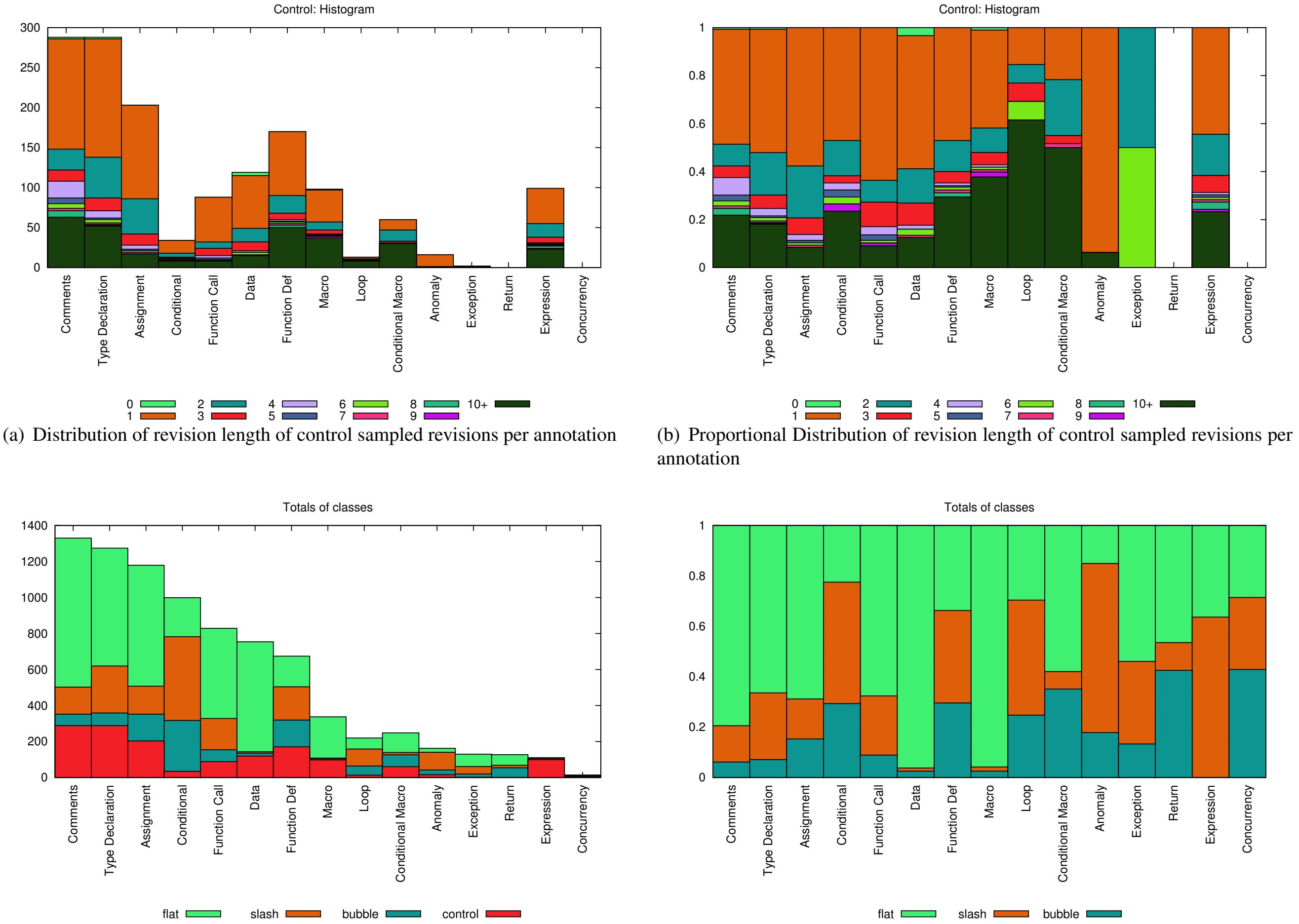
Flat revisions were mostly small, usually only 2 lines in length (see Figures 3(a) and 3(b)). Flat revisions were the most common shape found of the 3 indentation shapes. Comments were the most common flat revisions. Long flat revisions were often data definitions, macros and conditional macros.

Slash revisions were mostly conditionals and type declarations, these revisions were often 2 to 3 lines in length (see Figures 3(c) and 3(d)). There were not a lot of large slash revisions, although conditionals were the most numerous large slash revisions.

Bubble shape revisions were mostly conditionals and function definitions (as shown in figures 3(e) and 3(f)). Conditional revisions were usually 3 to 6 lines in length, function definition revisions where 3 to 7 lines in length. Returns and assignments were the shortest and most common of bubble revisions.

4.2. Total Classes

Figure 2(c) depicts the cumulative totals of the 3 indentation shape populations of indentation shapes and the sampled control set. Figure 2(d) shows the proportions of the 3 indentation shape populations. The control set was not included because it sampled from a different set of files so the



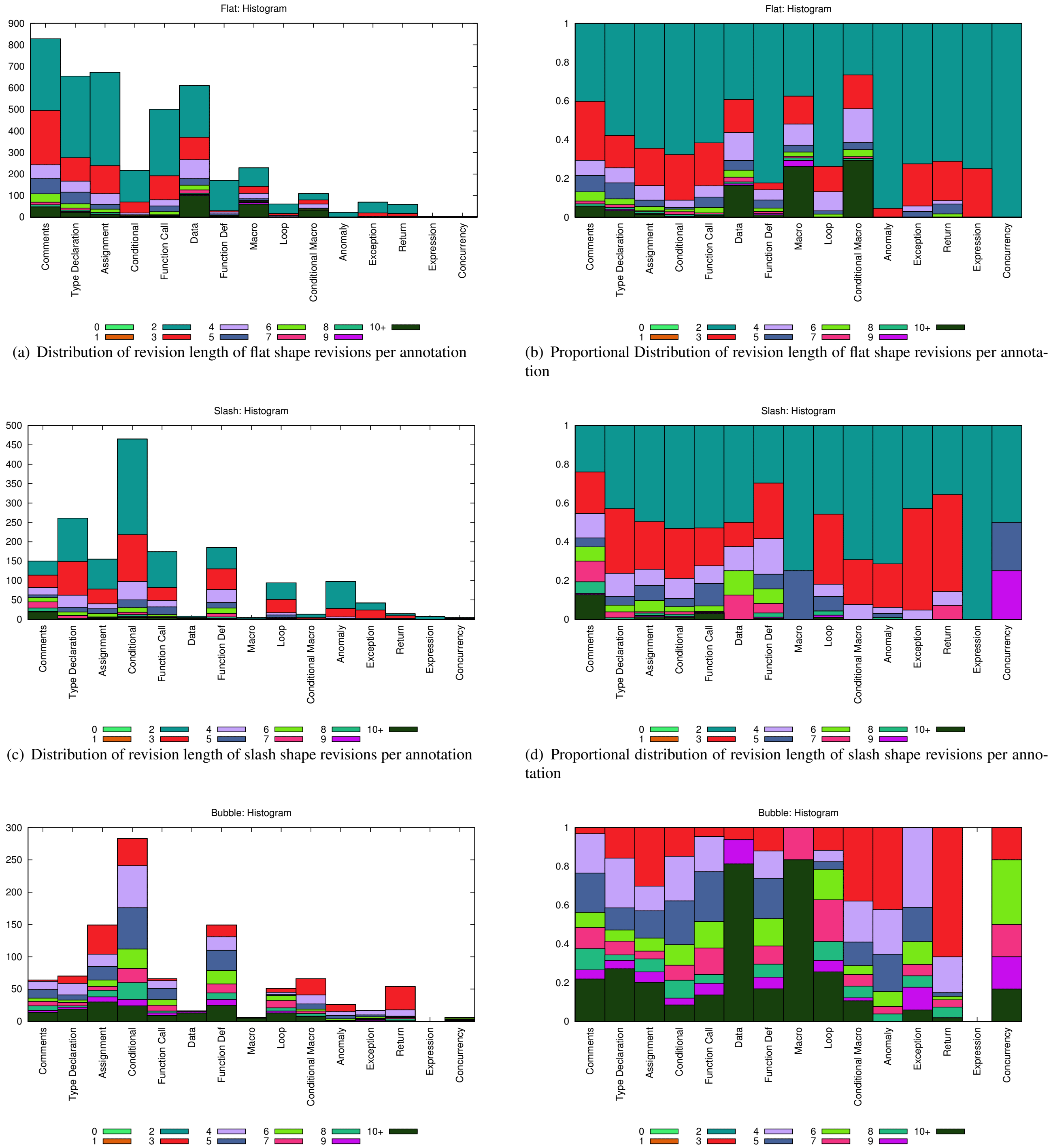
(a) Distribution of revision length of control sampled revisions per annotation

(b) Proportional Distribution of revision length of control sampled revisions per annotation

(c) Distribution of revisions by annotation, broken down by indentation shape

(d) Proportional Distribution of revisions by annotation, broken down by indentation shape (control is not included because it was sampled and does not represent a population)

Figure 2. Totals of Each Class per Population and Control Sample



(a) Distribution of revision length of flat shape revisions per annotation

(b) Proportional Distribution of revision length of flat shape revisions per annotation

(c) Distribution of revision length of slash shape revisions per annotation

(d) Proportional distribution of revision length of slash shape revisions per annotation

(e) Distribution of revision length of bubble shape revisions per annotation

(f) Proportional Distribution of revision length of bubble shape revisions per annotation

Figure 3. Distribution of annotations broken down by shape and by revision lengths of 0 to 10+, the length 10 bucket represents revisions of 10 or more lines.

proportion of control revisions to these 3 shape revisions is not really relevant.

Comments, type declarations, assignments and conditional were the most common revisions. Flat revisions were the most numerous revisions, they consisted of predominantly comments, assignments and data. Slash revisions and bubble revisions comprised the majority of the conditional revisions. Slash revisions represented more loop revisions than any other shapes.

4.3. Indentation Variance Perspective

The variance of indentation is calculated by creating of vector of indentation per line of a revision and then calculating the variance of that vector. We then break down the distribution of variance of indentation by bucketting them into quartiles. Figure 4 depicts the quartiles of the variance of indentation for various subsets of revisions. The quartiles are calculated by first bucketing each revision by their annotation. Then per each bucket, we order the revisions by their variance of indentation. This ordered list of revisions is split up into four contiguous subsets of equal size. These quartiles give us an idea of the proportion of revisions that are associated with a particular annotation, and what proportion of those revisions, in a particular bucket, are associated with certain values. Note that if one value is predominant (particularly 0) that the buckets that share the same range (0 to 0) are interchangeable (this is notable in Figure 4(a) and figure 4(b)).

Most slash revisions were conditionals or type declarations (see Figures 4(c) and 4(d)). Lower values of variance correlated more with conditionals than type declarations. Higher values of variance correlated with type declarations and function calls. Loops were more probable in the first three quartiles than the last quartile.

Most bubble revisions were conditionals, assignments and function definitions. The variance of these revisions was non-zero and was split uniformly across the quartiles (see Figures 4(e) and 4(f)). Assignments, returns and comments had the largest percentage of lower variance revisions. Data, conditional macros, and macro calls were not common bubble revisions.

Flat revisions had variances of 0, thus it was not useful to analyze flat revisions with variance as the revisions were evenly distributed across the quartiles.

More than half of *Control* revisions had an indentation variance of 0. Comments, type declarations and function definitions were biased to those revisions with variances larger than 0 (see Figures 4(a) and 4(b)). Control revisions seemed to be split into two groups, zero variance and non-zero variance. Small variances were associated with data revisions, assignments, comments and type declarations. Loops predominantly had larger variances.

We compared all the annotated revisions with the *control* set of revisions and found that the majority of revisions had 0 variance. The larger variance revisions were conditionals, function definitions and type declarations. Less frequent but still associated with large variances were loops and conditional macros.

4.4. Answers to Research Questions

Now we address the questions posed in section 2.1 about the distribution of the code shapes and the kind of change they represent.

What kind of indentation correlates with function definitions? Function modification seems much more prevalent in revisions with a higher variance of indentation like bubble, slash and the upper quartiles of revision length and variance of indentation of revisions.

What kinds of code correlate with zero variance indentation? Comments, type declarations, assignments and data were commonly found among revisions with flat or zero variance indentation.

What kinds of code correlate with non-zero variance indentation? Common code with non-zero variance indentation included conditionals, type declarations, function definitions, comments and assignments. The main difference between zero and non-zero variance indentation revisions that certain annotations such as data, comments and assignments were far less common. Also, macros and data revisions were far more common with zero indentation than non-zero indentation.

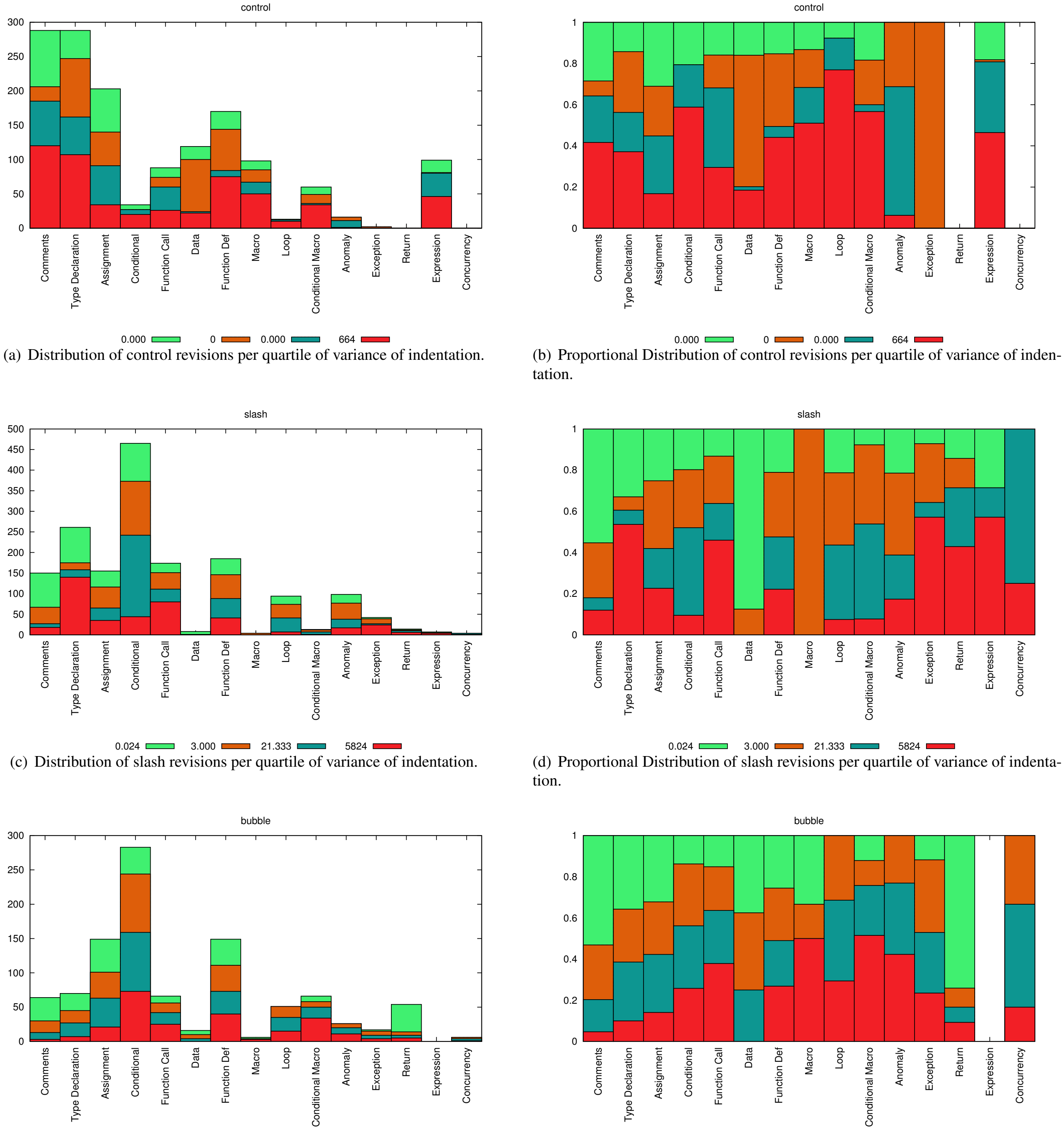
Among the bubble and slash indentation shapes, conditionals, and function definitions were far more common than with both flat revisions and revisions with 0 variance of indentation.

4.5. Integration with End-User Tools

We hope that with this data-set and these characterizations of the distributions of our annotations (effectively the underlying kinds of code), one could integrate this knowledge into a tool which browses revisions and emphasizes the kind of code to look for while browsing a repository. The browser could parse the indentation of the revisions, then allow the user to create a distribution from the annotation tags. Then by combining techniques like Bayesian inference, the indentation shapes and the measurements of indentation, the browser could suggest revisions for the user to inspect.

5. Validity Threats

Our work faces some potential threats to validity. The main vector of these threats include: sampling issues, generalizability, language and development tools issues, data cleaning, and bias issues.



(a) Distribution of control revisions per quartile of variance of indentation.

(b) Proportional Distribution of control revisions per quartile of variance of indentation.

(c) Distribution of slash revisions per quartile of variance of indentation.

(d) Proportional Distribution of slash revisions per quartile of variance of indentation.

(e) Distribution of bubble revisions per quartile of variance of indentation.

(f) Proportional Distribution of bubble revisions per quartile of variance of indentation.

Figure 4. The Quartiles of the Variance of the different indentation shapes and the control sample. Flat is not displayed because it has a variance of 0. Note how almost 3/4s of the control samples have a variance of 0.

Our selection of Source-Forge projects was not a random sample, it consisted of the most active and downloaded, that had repositories to examine. This software might not be representative of many classes of software. Sampling was done at the file level, not the diff chunk level, so this means that certain files might represent a larger proportion of the revisions depending on the age of the file. Due to the issues sampling and the languages studied our results may not be very generalizable. Another issue would be that code which changes a lot might exhibit certain properties, we might be measuring these properties rather than the properties of revisions themselves. We could suffer from a multiple counting problem because of multiple changes to the same hotspots.

Our annotations potentially suffer from personal bias. Although we tried to be as objective as possible, the annotation of a revision was made using our own best judgement. As researchers we could have let personal bias slip in even though we tried to be objective.

We studied code revisions to C, C++, Java, Perl, PHP and Python files, we did not analyze the more common XML files, Makefiles and shell scripts, we did not analyze HTML or Javascript. As well, these 6 languages are related to each other, they cover similar domains and share similar syntax. Also we did not study full source files, although much of that had been addressed by nesting metrics [7, 12].

Data cleaning posed a problem: sometimes a revision slipped through that did not match our rules well (like a slash shape of pure whitespace and no code). We had to mark these revisions as anomalies.

6. Conclusions

In conclusion, we have shown that indentation shapes such as slash and bubble have important properties. Namely, they commonly correspond to potentially complex code, such as loops or conditionals. Bubbles and slashes relate to branches in code. By comparison, more common shapes like flat shapes or revisions with a low variance of indentation are often comments, assignments, data and type declarations.

We have shown that revisions that have a non-zero variance of indentation are less likely to be comments and more likely to have syntactically interesting structure, that is they consist of conditionals such as if blocks, preprocessor conditionals, loops and function definitions. Revisions with non-zero variance of indentation often indicate code that is more interesting, more complex than revision with less variant indentation.

With this knowledge and these distributions it should be easy to create tools that browse revisions, measure their indentation and allow users to query for revisions which potentially match the annotation they are looking for. This is useful since parsing diffs requires contextual information often not included with the diff, this often happens if the diffs are from a patch received on a mailing list.

6.1. Future Work

Future work should further investigate indentation and revisions, specifically collections of revisions such as those found in a diff or a commit. We should also try other alternative metrics like characters and code characters per line.

Other questions remain, such as what the coupling of diff-chunks indicates. We also wish to identify the multiple purposes of a revision, commit or patch, and determine if we can associate the diff-chunks with each purpose; more specifically, can we identify which revisions are related to which purpose?

Acknowledgements: This work was partially funded by an NSERC PGS D Scholarship. We would like to thank SHARCNET for use of their infrastructure.

References

[1] R. E. Berry and B. A. Meekings. A style analysis of C programs. *Commun. ACM*, 28(1):80–88, 1985.

[2] X. Chen, B. Francia, M. Li, B. McKinnon, and A. Seker. Shared information and program plagiarism detection, 2004.

[3] D. Coleman, D. Ash, B. Lowther, and P. W. Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994.

[4] R. B. Findler. PLT DrScheme: Programming environment manual. Technical Report PLT-TR2007-3-v371, PLT Scheme Inc., 2007.

[5] D. M. German and A. Hindle. Measuring fine-grained change in software: towards modification-aware change metrics. In *Proceedings of 11th International Software Metrics Symposium*, 2005.

[6] N. Gorla, A. C. Benander, and B. A. Benander. Debugging effort estimation using software metrics. *IEEE Trans. Softw. Eng.*, 16(2):223–231, 1990.

[7] W. A. Harrison and K. I. Magel. A complexity measure based on nesting level. *SIGPLAN Not.*, 16(3):63–74, 1981.

[8] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Towards a theoretical model for software growth. In *MSR 2007: Proceedings*, page 21, Washington, DC, USA, 2007. IEEE Computer Society.

[9] A. Hindle, M. Godfrey, and R. Holt. Reading beside the lines: Indentation as a proxy for complexity metrics. In *Proceedings of ICPC 2008*, June 2008.

[10] R. F. Mathis. Flow trace of a structured program. *SIGPLAN Not.*, 10(4):33–37, 1975.

[11] R. J. Miara, J. A. Musselman, J. A. Navarro, and B. Shneiderman. Program indentation and comprehensibility. *Commun. ACM*, 26(11):861–867, 1983.

[12] J. Munson and T. Khoshgoftaar. The dimensionality of program complexity. *Software Engineering, 1989. 11th International Conference on*, pages 245–253, May 1989.

[13] P. W. Oman and C. R. Cook. Typographic style is more than cosmetic. *Commun. ACM*, 33(5):506–520, 1990.

[14] R. Power, D. Scott, and N. Bouayad-Agha. Document structure. *Comput. Linguist.*, 29(2):211–260, 2003.