Detecting(Understading) similar changes of concurrent related code*

[Extended Abstract] †

Feiyue Yu
Shanghai Jiaotong University
yufeiyue@sjtu.edu.cn

Hao Zhong
Shanghai Jiaotong University
zhonghao@sjtu.edu.cn

ABSTRACT

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for ACM SIG Proceedings. It is an *alternate* style which produces a *tighter-looking* paper and was designed in response to concerns expressed, by authors, over page-budgets. It complements the document *Author's (Alternate) Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX*. This source file has been written with the intention of being compiled under LaTeX2$_\epsilon$ and BibTeX.

The developers have tried to include every imaginable sort of "bells and whistles", such as a subtitle, footnotes on title, subtitle and authors, as well as in the text, and every optional component (e.g. Acknowledgments, Additional Authors, Appendices), not to mention examples of equations, theorems, tables and figures.

To make best use of this sample document, run it through LaTeX and BibTeX, and compare this source code with the printed output produced by the dvi file. A compiled PDF version is available on the web page to help you with the 'look and feel'.

CCS Concepts

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

Keywords

ACM proceedings; LaTeX; text tagging

1. INTRODUCTION

Concurrent programs are pervasive in nowadays software development activities. However, concurrent programs are known hard to write correctly for multiple threads accessing objects simultaneously or depending on each other usually need complex synchronization and hard to debug for the uncertainty of thread interleaving which makes it difficult to reproduce the bug. Developers struggle with various of synchronization methods and subtle concurrent bugs.

Benefit

However, this work has to face several challenges:

Challenge 1

Research shows that software modifications are usually similar but not identical.

Our main contributions are:

The rest of paper is organized as follows: Section 2 presents the methodology of our study. Section 3 presents our result and discussion. Section 4 presents related work. Section 5 presents future work and Section 6 concludes.

2. METHODOLOGY

This section presents the project sources of our study, research questions and methods of doing the study. We have developed a tool supporting the empirical study.

2.1 Project Sources

We investigate 8 Java open-source projects from Github including Hadoop, Tomcat, Cassandra, Lucene-solr, Netty, Flink, Guava and Mahout as shown in Table 1. They are all popular, large-scale, active, representative Java open-source projects and cover different areas like distributed computing, web server, database, information retrieval, I/O and machine learning. The Hadoop project develops open-source software for reliable, scalable, distributed computing and has become one of the most famous Java open-source software for many years. Tomcat is the most popular implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. Cassandra is a database system which can Manage massive amounts of data, fast, without losing sleep. Lucene-solr is two projects together in one respository in Github. Lucene is a search engine library and solr is a search engine server which uses lucene. Netty is an event-driven asynchronous network application framework. Flink is an open source stream processing framework with powerful stream- and batch-processing capabilities. Mahout is a machine learning project. Table 1 shows the lines of code in Java, the number of Java files and the number of commits of each project. All the projects are checked out for our study in November 2016.

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

†A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX* at `www.acm.org/eaddress.htm`

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

Table 1: Projects information (LOC and #Files are both of Java files)

Project	LOC	#Files	#Commits
Hadoop	1202764	7701	14930
Tomcat	301173	2192	17731
Cassandra	387980	2143	21982
Lucene-solr	918398	6310	26152
Netty	218131	2054	7759
Flink	414264	4068	9771
Guava	251205	1672	3850
Mahout	109584	1215	3703

2.2 Research questions

In order to understand the evolution of concurrent code better, we proposed 4 research questions:

RQ1. How frequent do concurrent related code modification appear in different kind of Java open-source projects?

Concurrent programming is very popular in today's Java development with the rapid developments of multi-core techniques which help exploit the power of concurrent programming. Java programming language provides convenient built-in concurrent libraries and users can also invoke third-party libraries. We want to know how frequent do concurrent related code modification in different kind of repositories.

RQ2. What is the trend of certain concurrent programming construct usage statistically?

RQ3. Is there any change pattern in concurrent programming?

RQ4. How can these change patterns in history guide the maintaining in development?

2.3 Methods

All the projects of our study are under git which is one of the most popular version control systems in the world. Some projects of the study used svn or some other version control systems before because they have long histories, but they all support git now. We employ JGit, a lightweight, pure java library implementation of git, to retrieve all the commit logs in projects' histories. A typical commit log contains commit id which is a 20-character-long string uniquely identifying a commit, author, date and message. Once we get a commit id, we use "git show" command to show the log message and textual diff. The diff result contains one or more change files which contain one or more change hunks.

2.4 Collecting code snippets

2.4.1 Checking out versions

All the projects of our study are under git which is one of the most popular version control systems in the world. We first use "git diff-tree –no-commit-id –name-status -r <commitId>" command to get changed file list of each commit. For each Java source file in the list, we use "git checkout <commitId> – <filename>" and "git checkout <commitId> ^- <filename>" command to check out the old version before the commit and the new version after the commit. Then we extract matching methods of two files base on method name and list of its arguments.

Method-level unit is chosen as the unit of code snippets in our study because file-level unit is too large, whose intrinsic code correlation is not as compact as smaller code unit and which probably contains noises of multiple uncorrelated modifications. More fine-grained level of unit like a group of continuous statements has higher precision and correlation with the modification but is more difficult to identify and implement. Therefore, method-level unit is a practical choice.

2.4.2 Preprocessing

Pairs of methods extracted need to be preprocessed. Noticing that a Java class usually has more codes which are mostly class variable declarations outside method definitions, we rewrite the methods to make them more complete without losing the type information defined in the class. We copy a variable declaration into a method if it is referenced in this method. If a local variable in method and a class variable have the same identifiers, the class variable will be renamed to solve the conflict problem. Figure 1 shows an example from .

2.4.3 Selecting concurrent related snippets

After we have pairs of code snippets to be further analysed, we present how we select concurrent related snippets. Concurrent related code is a semantic concept. There are too many third-party libraries and user-defined classes that have concurrent related functionalities. So it is hard to select concurrent related code precisely. We use a naive but effective keyword matching method.

A concurrent keyword here is defined as one of the class names or interface names of Java concurrent package and some other concurrent related keywords in Java like 'synchronize' and 'Thread'. Java concurrent package is a common-used concurrent libraries which provide many useful features. It is introduced into Java standard library since Java 1.5. We define a concurrent related snippet as a snippet which contains a concurrent keyword. A pair of snippets is concurrent related if any snippet of the pair is concurrent related. We traverse all the pairs of snippets and reserve those which are concurrent related.

A string matching algorithm is used to check if a snippet contains a concurrent keyword. A keyword sometimes is not at the position which it should be. For example, a type name occurs in a string or a comment. But now that the keyword exists in the code for some reasons, the code tend to has some relationships with concurrency more or less. This method might omit some snippets that are concurrent related, but the selected ones are mostly concurrent related snippets indeed. So, string matching algorithm is acceptable and practical.

We first identify the Java code of a commit log.

2.5 Classification

Most of the commits selected by the last step are not our targets since they usually add or modify functionalities which are specific although they contain some concurrent keywords. The target commits to be analysed occupy a very small proportion of all the commits even we have already filtered them. So, we need some automatic methods to help finish the job.

We use the SVM method to classify commits as concurrent-related or not. SVM is a supervised classification algorithm which needs both positive and negative training data. We extracted 12 features from each commit and labeled 65 instances manually as a training data set. Testing on the

training set itself has an accuracy of 98.46%. The classifier selects 96 positive instances from 9891 instances.

2.6 Extracting changes

We employ a tree based differencing algorithm to extract code changes of matching methods. For a matching pair of methods before and after a modification, the differencing algorithm generates a sequence of edit operations. An edit operation is one of the following types:

Insert
Delete
Update
We give some definitions first.

Definition 1. Two nodes are exact matching if their

We traverse the two ASTs of a pair of snippets using a pre-order search and compare the corresponding nodes.

2.7 Calculating similarity

The code similarity has two aspects: context similarity and change similarity. Detecting similar code snippets can use code clone detection techniques. Code clone detection approaches can be categorised into four main categories [6], namely, text or string based approach, lexical or token based approach, tree based approach and semantic approach.

2.7.1 Grouping

Comparing each code snippet with each other cost too much time which is in an order of $O(n^2)$ where n is the number of snippets. We narrow the search space of clone detection by grouping snippets fast by keyword. The groups are not orthometric. It is not classification. A pair snippet may belong to multiple groups. For example, a group consists of all pairs which contain keyword k.

2.7.2 Extracting change-related context

We use data dependency analysis to keep only nearly code which has relationship with the changes.

2.7.3 Context similarity

A context is several neighbour statements of change locations. We take the original code that will be changed into consideration.

We employ a tree based code clone detecting approach to detect similar code snippets.

We apply machine learning techniques to classify code snippets.

2.7.4 Change similarity

Prepare snippet pairs Java concurrent package is a common-used concurrent libraries which provide many practical features. We use class name and interface name of the package to selected more than 100000+ concurrent related program snippets modification pairs from 8 popular projects from github. If a program snippet contains any class or interface from java concurrent package, it is considered as a concurrent related program snippet.

Clustering We believe that similar program pieces tend to have some similar changes in commit histories. Program pieces are clustered into several group based on concurrent keywords which are class name and interface name from the java concurrent package and 'synchronize'. We use a tfidf model to build a numeric vector for each snippet then employ weka clustering tools including kmeans and em algorithms to cluster the extracted snippets.

Collecting code snippets Prepare a database of concurrent related pair of code snippets. A pair means the original code and modified code.

Extracting changes Extract changes of pair of code snippets. Changes consist of a sequence of edit operations like insert, delete and update.

Calculating similarity

3. RESULTS

3.1 RQ3.

Most changes to the concurrent code are due to logic change, new feature or bug fix, such as we should synchronize this method because it probably be accessed concurrently. They are usually specific and hard to apply them to other programs. It needs much effort to understand these changes because they heavily depend on the business logic. A small amount of these changes can be regarded as common transformation patterns which often appear in software evolution history and can be apply to other programs.

We identify and categorise these transformation patterns into the following classes:

(1).Use thread-safe class instead of handling synchronization problem manually.

(2).Use thread synchronization instead of checking and sleep cyclically.

(3).Replace a class with another class which has similar functionalities but offers some advantages in the scenario.

We also have some other findings:

(1).Developers are refactoring their code by using the later version of the JDK library.

(2).Developers are using some code-checking tools like findbugs to help them inspect their code.

(3).Some transformations have opposite directions.

(4).

use readlock instead of synchronized block use new class in jdk7 ThreadLocalRandom guava cache or jdk concurrenthashmap atomicinteger, atomiclong, synchronized concurrentskiplistmap for performance atomicreference Use ThreadLocalRandom and remove FBUtilities.threadLocalRandom Use AtomicIntegerFieldUpdater in RefCountedMemory to save memory in row cache replace volatile regionCount w/ AtomicInteger replace one-shot lock w/ synchronized block use CallerRunsPolicy instead of rejecting runnables on multithreaded executors w/ blocking queues Replace synchronization in Gossiper with concurrent data structures and volatile fields switch singleton implementation from double-checked-locking to synchronized (code is not performance-sensitive) use ConcurrentHashMap to simplify ugly loop (normal HashMap you cannot delete from during iteration) volatile should ensure double locking to work properly Changed Counters to use ConcurrentSkipListMap for performance Use ThreadLocalRandom where possible 820 1205 3641 Code was used although I can't see why a simple AtomicInteger wasn't sufficient.

Double checked locking is a synchronization pattern in software engineering. It first check the condition without lock to reduce the time overhead when the condition is not satisfied. A typical usage of it is singleton pattern which uses lazy initialization to provide an unique instance dur-

ing the process execution time in multi-threaded scenario. But sometimes programmers make some mistakes using this pattern like forgetting 'volatile' modifier of the member of object in Java.

Concurrent data structures provide convenience for concurrent programming without caring about complicated thread-safe problems. So many commits turn to use these well-written libraries instead of synchronize on their own.

AtomicIntegerFieldUpdater

ThreadLocalRandom is combination of ThreadLocal and Random.

3.2 Implications

This research provides some implications from different kinds of perspectives.

(1) Developers are facing more and more concurrent programming requirements now. But concurrent programming is notoriously error-prone because of the complexity of data synchronization and thread interleaving. Our study gives developers some guidelines of writing concurrent programs. First, use handy concurrent libraries to finish the job instead of rewrite them by yourself unless all the available concurrent libraries cannot satisfy your requirement and you are absolutely confident of your comcurrent programming skills. Using existing libraries allows you to write less code to finish the same work and enjoy the high quality of implementation which is always reliable, strong and fast.

(2) Automatic tools are needed to help developers inspect and revise concurrent programs with the help of history information. There has already been some tools, but they usually look for concurrent bugs such as race detection, deadlock detection and atomicity violation without considering software evolution history. Both project specific and project independent transformation patterns exist in real-world software projects. So we need some concurrent code refactoring tools to give advice of what code need change and perform the transformations automatically. Developers will benefit a lot if such kind of automatic tools can actually help them automate their development and maintaining activities.

(3)

4. RELATED WORK

The related work includes studies on concurrent program, transformation pattern and refactoring.

Studies on concurrent program

Transformation pattern

Refactoring

5. FUTURE WORK

We have learned that similar code changes of concurrent related code are common in history of software evolution. This indicates some automatic tools may help developers maintain their code.

Identify and extract transformation pattern in concurrent programs automatically.

Apply the patterns to new suitable context automatically.

6. CONCLUSION

We conduct a study on similar code changes of concurrent related code.

7. INTRODUCTION

The *proceedings* are the records of a conference. ACM seeks to give these conference by-products a uniform, high-quality appearance. To do this, ACM has some rigid requirements for the format of the proceedings documents: there is a specified format (balanced double columns), a specified set of fonts (Arial or Helvetica and Times Roman) in certain specified sizes (for instance, 9 point for body copy), a specified live area (18 × 23.5 cm [7" × 9.25"]) centered on the page, specified size of margins (1.9 cm [0.75"]) top, (2.54 cm [1"]) bottom and (1.9 cm [.75"]) left and right; specified column width (8.45 cm [3.33"]) and gutter size (.83 cm [.33"]).

The good news is, with only a handful of manual settings[1], the LaTeX document class file handles all of this for you.

The remainder of this document is concerned with showing, in the context of an "actual" document, the LaTeX commands specifically available for denoting the structure of a proceedings paper, rather than with giving rigorous descriptions or explanations of such commands.

8. THE *BODY* OF THE PAPER

Typically, the body of a paper is organized into a hierarchical structure, with numbered or unnumbered headings for sections, subsections, sub-subsections, and even smaller sections. The command `\section` that precedes this paragraph is part of such a hierarchy.[2] LaTeX handles the numbering and placement of these headings for you, when you use the appropriate heading commands around the titles of the headings. If you want a sub-subsection or smaller part to be unnumbered in your output, simply append an asterisk to the command name. Examples of both numbered and unnumbered headings will appear throughout the balance of this sample document.

Because the entire article is contained in the **document** environment, you can indicate the start of a new paragraph with a blank line in your input file; that is why this sentence forms a separate paragraph.

8.1 Type Changes and *Special* Characters

We have already seen several typeface changes in this sample. You can indicate italicized words or phrases in your text with the command `\textit`; emboldening with the command `\textbf` and typewriter-style (for instance, for computer code) with `\texttt`. But remember, you do not have to indicate typestyle changes when such changes are part of the *structural* elements of your article; for instance, the heading of this subsection will be in a sans serif[3] typeface, but that is handled by the document class file. Take care with the use of[4] the curly braces in typeface changes; they mark the beginning and end of the text that is to be in the different typeface.

[1]Two of these, the `\numberofauthors` and `\alignauthor` commands, you have already used; another, `\balancecolumns`, will be used in your very last run of LaTeX to ensure balanced column heights on the last page.

[2]This is the second footnote. It starts a series of three footnotes that add nothing informational, but just give an idea of how footnotes work and look. It is a wordy one, just so you see how a longish one plays out.

[3]A third footnote, here. Let's make this a rather short one to see how it looks.

[4]A fourth, and last, footnote.

You can use whatever symbols, accented characters, or non-English characters you need anywhere in your document; you can find a complete list of what is available in the *LaTeX User's Guide*[5].

8.2 Math Equations

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of the three are discussed in the next sections.

8.2.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin. . .\end` construction or with the short form `$. . .$`. You can use any of the symbols and structures, from α to ω, available in LaTeX[5]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\to\infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

8.2.2 Display Equations

A numbered display equation – one set off by vertical space from the text and centered horizontally – is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n\to\infty} x = 0 \tag{1}$$

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_0^{\pi+2} f \tag{2}$$

just to demonstrate LaTeX's able handling of numbering.

8.3 Citations

Citations to articles [1, 3, 2, 4], conference proceedings [3] or books [7, 5] listed in the Bibliography section of your article will occur throughout the text of your article. You should use BibTeX to automatically produce this bibliography; you simply need to insert one of several citation commands with a key of the item cited in the proper location in the `.tex` file [5]. The key is a short reference you invent to uniquely identify each work; in this sample document, the key is the first author's surname and a word from the title. This identifying key is included with each item in the `.bib` file for your article.

The details of the construction of the `.bib` file are beyond the scope of this sample document, but more information can be found in the *Author's Guide*, and exhaustive details in the *LaTeX User's Guide*[5].

Table 2: Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

Figure 1: A sample black and white graphic.

This article shows only the plainest form of the citation command, using `\cite`. This is what is stipulated in the SIGS style specifications. No other citation format is endorsed or supported.

8.4 Tables

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material is found in the *LaTeX User's Guide*.

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed dvi output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed dvi output of this document.

8.5 Figures

Like tables, figures cannot be split across pages; the best placement for them is typically the top or the bottom of the page nearest their initial cite. To ensure this proper "floating" placement of figures, use the environment **figure** to enclose the figure and its caption.

This sample document contains examples of **.eps** files to be displayable with LaTeX. If you work with pdfLaTeX, use files in the **.pdf** format. Note that most modern TeX system will convert **.eps** to **.pdf** for you on the fly. More details on each of these is found in the *Author's Guide*.

As was the case with tables, you may want a figure that spans two columns. To do this, and still to ensure proper "floating" placement of tables, use the environment **figure*** to enclose the figure and its caption. and don't forget to end the environment with figure*, not figure!

8.6 Theorem-like Constructs

Other common constructs that may occur in your article

Table 3: Some Typical Commands

Command	A Number	Comments
`\alignauthor`	100	Author alignment
`\numberofauthors`	200	Author enumeration
`\table`	300	For tables
`\table*`	400	For wider tables

Figure 2: A sample black and white graphic that has been resized with the `includegraphics` command.

are the forms for logical constructs like theorems, axioms, corollaries and proofs. There are two forms, one produced by the command `\newtheorem` and the other by the command `\newdef`; perhaps the clearest and easiest way to distinguish them is to compare the two in the output of this sample document:

This uses the **theorem** environment, created by the `\newtheorem` command:

THEOREM 1. *Let f be continuous on $[a, b]$. If G is an antiderivative for f on $[a, b]$, then*

$$\int_a^b f(t)dt = G(b) - G(a).$$

The other uses the **definition** environment, created by the `\newdef` command:

Definition 1. If z is irrational, then by e^z we mean the unique number which has logarithm z:

$$\log e^z = z$$

Two lists of constructs that use one of these forms is given in the *Author's Guidelines*.

There is one other similar construct environment, which is already set up for you; i.e. you must *not* use a `\newdef` command to create it: the **proof** environment. Here is a example of its use:

PROOF. Suppose on the contrary there exists a real number L such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = L.$$

Then

$$l = \lim_{x\to c} f(x) = \lim_{x\to c} \left[gx \cdot \frac{f(x)}{g(x)} \right] = \lim_{x\to c} g(x) \cdot \lim_{x\to c} \frac{f(x)}{g(x)} = 0 \cdot L = 0,$$

which contradicts our assumption that $l \neq 0$. □

Complete rules about using these environments and using the two different creation commands are in the *Author's Guide*; please consult it for more detailed instructions. If you need to use another construct, not listed therein, which you want to have the same formatting as the Theorem or the Definition[7] shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A *Caveat* for the TEX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use TEX's `\def` to create a new command: *Please refrain from doing this!* Remember that your LATEX source code is primarily intended to create camera-ready copy, but may be converted to other forms – e.g. HTML. If you inadvertently omit some or all of the `\def`s recompilation will be, to say the least, problematic.

9. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LATEX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

10. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

11. ADDITIONAL AUTHORS

Additional authors: John Smith (The Thørväld Group, email: `jsmith@affiliation.org`) and Julius P. Kumquat (The Kumquat Consortium, email: `jpkumquat@consortium.net`).

12. REFERENCES

[1] M. Bowman, S. K. Debray, and L. L. Peterson. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.*, 15(5):795–825, November 1993.

[2] J. Braams. Babel, a multilingual style-option system for use with latex's standard document styles. *TUGboat*, 12(2):291–301, June 1991.

[3] M. Clark. Post congress tristesse. In *TeX90 Conference Proceedings*, pages 84–89. TeX Users Group, March 1991.

[4] M. Herlihy. A methodology for implementing highly concurrent data objects. *ACM Trans. Program. Lang. Syst.*, 15(5):745–770, November 1993.

[5] L. Lamport. *LaTeX User's Guide and Document Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[6] O. Onuoha. Detecting code clones: A review. *CoRR*, abs/1605.02661, 2016.

[7] S. Salas and E. Hille. *Calculus: One and Several Variable*. John Wiley and Sons, New York, 1978.

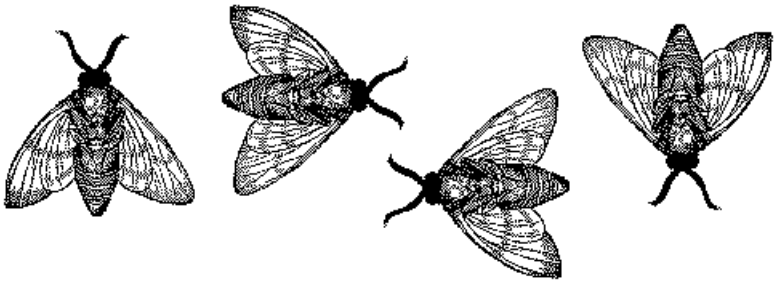

Figure 3: A sample black and white graphic that needs to span two columns of text.

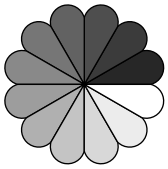

Figure 4: A sample black and white graphic that has been resized with the `includegraphics` command.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations.

Display Equations.

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LaTeX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The sig-alternate.cls file itself is chock-full of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LaTeX, you may find reading it useful but please remember not to change it.