The Style Guide as an Instruction Tool for Structured Programming

Scott D. Baldwin
Oklahoma State University

Abstract

Structured programming skills are a must for anyone writing computer software. This paper will present to the reader the concept of the style guide as a tool for insuring structured programming skills are developed by both novice and experienced programmers.

A style guide is normally a written document, either hard copy or on-line, that provides rules for formatting, documenting, variable naming and other important skills that are required for insuring written code is readable, reusable and maintainable. One style guide is not suitable for all programming languages and therefore each language will normally have it's own style guide covering a standard set of topics.

Topics of the paper are a detailed description of a style guide, how to develop a sound style guide, introduction of the style guide concepts through visual examples and methods of enforcement in the class room environment.

I. Introduction

In this age where software is embedded in practically every consumer product sold, topics such as software reusability and maintainability are on the minds of all software managers. As engineers, our graduates can almost be guaranteed that they will be expected to develop code at some time in their career. It is almost an even greater certainty that the code they develop will be used again or modified at a later date. It has been shown that 40 to 60% of all code is reusable from one application to another, 60% of the design and code in all business applications is reusable, 75% of the program functionality is common to more than one program, and only 15% of the code is unique to a specific application[1].

Reuse and maintenance of code typically means code modification. The modification may or may not be minor in scope. But with today's dynamic employment market, we can expect that the individual modifying the code did not originally write the code and will therefore have to rediscover the architecture of the system[2]. With an expected shortfall of 1 million software engineers in the year 2000[3], we can expect more and more of our graduates to be tasked with the job or developing original code and modifying existing code. It is therefore important that we provide them with the guidance from industry as to what is considered to be acceptable design practices for software. A common tool used in industry, and sometimes in academia is the style guide.

II. What is a Style Guide?

Style guides are documents that provide guidance to the software developer in terms of how to make the code they develop easier to read and therefore easier to debug and modify. Topics such as what constitutes an acceptable variable name, appropriate code indentation and guidelines on required levels of commenting are usually defined. Obviously, one style guide does not fit all. There are inherent differences between assembly language code, C++, and PASCAL that force a different style guide to be a written specifically for each language. There will be obvious overlap between style guides designed for different languages.

Style guides are exactly that, an effort to create a specific style in the written code among members of the same development team. It should provide alternatives whenever possible, not rules[4]. Therefore, a style guide written by company X is probably not acceptable for company Y. The factors that influence style guide content can be functional, internal and external customer requirements and personal preference. These issues should be pointed out to the student. They should be prepared to encounter totally different versions of style guides as they progress in their careers and move from company to company. They may find that no such document exists at their current employer. If not, hopefully the educational institute from which they graduated exposed them to the concept and prepared them adequately.

III. Developing A Sound Style Guide

A style guide, at a minimum, should provide the code developer with the following guidance as applicable:

- What are valid variable names
- What are valid function names
- What are valid file names
- Proper use of comments
- How to properly indent sections of code to show dependencies
- Proper use of white space
- Program organization

An example section of a style guide is provided below and is taken from a style guide for a C++ course taught at Oklahoma State.

Start of Example:

Identifiers

Standard
Identifiers must be meaningful names. The names of variables, functions and classes should be descriptive and easy to read. A variable name must reflect what the variable represents.

For example, the name you use for a variable to represent resistance should be resistance, resistorValue or ohms, not r or o. You can abbreviate identifier names to a degree. For example, you could abbreviate resistorValue to resVal.

Standard

Identifiers for things must be nouns. Things includes variables representing items or functions that return non-boolean values.

For example, a variable representing the name of a student might be studentName, but not nameStudent (as this implies an action).

Standard

Identifiers for functions must be verbs or verb-noun combinations since they indicate action

For example,

- *A function that displays a list of test scores might be DisplayTestScores, but not TestScores.*
- *A member function that increments a counter in a timer object might be IncrementTimer or Increment, but not Time.*
- *A member function that stores the name of a player in a class instance might be SetName.*

End of Example

Obviously, the best way to develop a style guide is to look at existing style guides, take what you like and change what you don't. Some links to on-line style guide resources are at the following web addresses:

Language	Web Address
HTML	http://guinan.gsfc.nasa.gov/Style.html
C++	http://techweb.ceat.okstate.edu/eet/Syllabi/eet2303/index.htm
Java	http://www.infospheres.caltech.edu/resources/index.html
MATLAB	http://www.met.nps.navy.mil/~jordan/mr2020/styleguide.html

IV. Introducing Style Guide Concepts Into the Classroom

The best way to introduce style guide concepts is through code examples. Everyone knows that a moving target is hard to hit. Many texts preach style guide concepts but the code examples within are in direct conflict. It is important that the student see what is considered properly created code that meets the intent of the published style guide. This can be done by providing correctly formatted code within the style guide as examples. Also use example code that comply with the style guide during lecture. Try to avoid using code examples with poor design techniques, even if the code is taken from a textbook.

The style guide can also be presented to the student in several ways. Hard copies of the style guide can be provided to the student. On-line versions are also popular. A piecemeal approach may be less overwhelming with pertinent sections of the style guide provided to the student as lab assignments are worked throughout the semester. By the end of the semester, the whole style guide will have been provided.

V. Enforcement of Style Guide Concepts In the Classroom

Program functionality has little to do with how the source code of a program is documented and variables are named. Grading programs based on functionality alone is not sufficient when trying to develop structured programming skills. The style guide is the written document with the structure guidelines to follow.

Style guides usually present programming structure concepts in two forms: **Standards** and **guidelines**. Standards are concepts that should be enforced absolutely. Guidelines are intended to provide some personal choice. Deviation from guidelines should not influence grades very much as long as the intent of the guide is addressed. Deviation from standards should have greater impact on the programming grade.

People like lists[3]. A single page checklist can provide the student with an easy way of establishing if the code they are developing meets all of the requirements if through no other method than prompting. The style guide can be used to explain why the requirement is there, the checklist can be used to remind the student that there is an associated requirement. As an instructor or grader, a similar checklist with room for comments can be used to provide students with feedback about how well they are complying with the style guide. This process is highly manual in nature, but does introduce the student to the concept of code reviews.

An extension of this process would be code reviews by student groups. Assignment of fellow class mates to review each other's work can help foster work environment dynamics in the classroom. These peer reviews will also expose each student to different styles of code development.

VI. Conclusions

Quality of code is above and beyond functionality. Although functionality and other qualities are closely related, functionality often takes not only the front seat in the development scheme but the only seat. Systems are frequently redesigned not because they are functionally deficient but because they are difficult to maintain. A style guide can be used to help develop sound coding skills so that code is easier to debug and maintain.

Bibliography

1. Blum, B. Software Engineering, A Holistic View, Oxford University Press, 1992.
2. Bass, L., Clements, P., and Kazman, R. Software Architecture in Practice, Addison-Wesley, 1998.
3. Moriguchi, S. Software Excellence, A Total Quality Management Guide, Productivity Press, 1997.
4. Kovitz, B Practical Software Requirements, A Manual of Content and Style, Manning, 1999.

SCOTT BALDWIN

Scott Baldwin is an Assistant Professor in the Electrical Engineering Technology department at Oklahoma State University where he has taught since January, 1999. He has worked in industry as a test engineer for several companies. He received both his B.S. degree in Electrical Engineering Technology in 1988 and M.S. in Electrical Engineering in 1998 from Oklahoma State University.