December 2007

Programming Standards for the Information Systems Curriculum

Julie Tomlin  
*Colorado State University*

John Hoxmeier  
*Colorado State University*

Susan Athey  
*Colorado State University*

Follow this and additional works at: [http://aisel.aisnet.org/amcis2007/](http://aisel.aisnet.org/amcis2007/)

**Recommended Citation**


This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 2007 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
Programming Standards for the Information Systems Curriculum

Julie Tomlin, John Hoxmeier, Susan Athey
Colorado State University
College of Business
Fort Collins, CO
jhox@lamar.colostate.edu

Abstract

Computer Information Systems Departments face a diverse set of platforms, languages, and architectures that make it difficult to establish a single, uniform guideline for coding standards and programming best practices. There are multiple “standards” available from vendors, associations, individuals, and user groups. Students and faculty alike benefit from practicing certain programming standards. Students become comfortable using an accepted, widely used style, and thus adapt more easily to a programming team upon entering the corporate world. There is considerable agreement that utilizing standards leads to better quality code that is easier to maintain. The goal of this project is to produce a set of recommendations toward a programming standards document for the CIS department. Research was done to identify the details behind multiple operating standards recommended by programming texts and supplier resources. Software professionals from several companies were interviewed to learn what standardized practices they employ. A set of base recommendations that seem to overlap the many offerings has been established by the CIS department based upon the results.

Keywords: Programming, Standards, Coding, Best Practice, IS Education
Introduction

The Computer Information Systems Department (CIS) at Colorado State University utilizes several platforms and languages within the CIS curriculum. While the department primarily utilizes the Microsoft Windows infrastructure, courses are also based on Java, C#, html/xml, IBM Linux and RUP, PHP, SQL, and incorporate database management and middleware from multiple vendors. Several years ago, the department utilized a programming standards document that described the coding techniques that students should follow when programming in COBOL. The professors used the standards to foster uniformity among CIS courses and to develop best practices among students. The now diverse curriculum makes it difficult to establish a single, uniform guideline for coding standards and programming best practices. There are multiple “standards” available from vendors, associations, individuals, and user groups. The department no longer has a document to address the current languages and environments used in the curriculum. Students and faculty alike would benefit from practicing certain standards. Students become comfortable using an accepted, widely used style, and thus adapt more easily to a programming team upon entering the corporate world. There is considerable agreement that utilizing standards leads to better quality code that is easier to maintain. For example, the Java Code Conventions state “Code conventions are important to programmers for a number of reasons:

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.”

Other arguments can be found in Fryman (1999) and Howles (2003).

Objective
The goal of this project was to begin to produce a set of recommendations toward a programming standards document for the CIS department. Research was done to identify the details behind multiple operating standards recommended by programming texts and supplier resources. Software professionals from several companies were interviewed to learn what standardized practices they employ. A set of base recommendations that seem to overlap the many offerings has been established by the CIS department based upon the results.

**History of Standards**

In 1974, the U.S. Navy began work on “one of the first standards treating the usage, control, and management of embedded computer resources” (Moore, 1998). In 1976, the National Bureau of Standards published *Guidelines for Documentation of Computer Programs and Automated Systems*. It focused on such requirements as functional, data, system, program, and database. In 1979, IEEE’s recently founded Software Engineering Standards Subcommittee (SESS) produced the Standard for a Quality Assurance Plan, which focused on documentation requirements. Subsequently, the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) created international software standards. They later created a joint committee focused on the area of information technology. The organizations continue to develop and publish standards in the IT field. Between all of them, they have published hundreds of standards on a variety of issues.

In the 1990s, some major companies produced standards for the language to which they own the rights. Sun produced standards for Java and last revised them in 1999 (Sun Microsystems, 1999). Microsoft published the “Visual Basic 6.0 Programmer’s Guide” in 1998. There are currently industry-accepted standard naming conventions for the Microsoft environments of J#, C# and VB.NET programs and a code analysis software tool (FX Cop) for the .NET framework (Brown, 2006; MSDN, 2007). Individual naming conventions at organizations may vary (Microsoft only suggests conventions for public and protected items). Oracle produces best practice and standards for languages such as Java, PHP and PL/SQL (Feuerstein, 2007).
**Literature Review**

We often hear in IT, “the nice thing about standards is that there are a lot to pick from.” In the programming standards arena alone, there are topics on which general agreement exists while other topics generate more disparate views. Regarding variable name length for example, the Sun Java website says that variable names should be short yet meaningful (McConnell, 2004), while Microsoft recommends that a name be as long as necessary to describe its purpose but warns to not make names longer than 32 characters because they can be hard to read on some screens. (Visual Basic Programming Conventions, 2004). The Hungarian convention for C also promotes shorter, more compact names. In “Code Complete, Second Edition,” Steve McConnell states that short names do not reveal enough meaning and longer names are hard to type and usually detract from the program (2004). Thus, there are contrasting views on name length.

Abbreviations are accepted in all of the standards. The authors use them frequently in code examples and acknowledge that programmers often utilize them. Some texts note that abbreviations can minimize the number of typos. For example McConnell reasons that longer names can be prone to errors when typing. For this reason, Microsoft and others recommend using abbreviations for frequently used or longer names. However, most texts warn to be consistent when using abbreviations because a programmer should not have to guess whether they abbreviated “current” as *curr* or *crnt*. Some sources, such as Sun’s website, say that abbreviations should be mnemonic, for example *stdnt* for “student.” McConnell agrees with this, yet claims that abbreviations are hardly ever needed with modern programming languages (2004). If abbreviations are used, he suggests: use standard abbreviations, remove vowels, remove insignificant words, do not remove just one letter from a word because it is not justified, be consistent, use names you can pronounce, and include a translation table in the comments of your code (2004).

The advice on using single character variables says they should only be used for temporary “throw-away” variables. The single character reveals that its function is limited in scope. Thus the programmer can assume “this variable is a run-of-the-mill loop counter or array index and doesn’t have
any significance outside these few lines of code” (McConnell, 2004). Sun notes that the characters $i, j, k, m$ and $n$ are often used for integers while $c, d,$ and $e$ are used for characters.

Both the Sun Java website and the Microsoft Visual Basic (version 6.0 and .Net) website recommend using mixed case and capital letters to denote new words in their naming conventions. The Microsoft permits starting the variable with a capital letter, while Sun says the first letter should be lowercase. Most other literature recommends mixed case, although Oracle recommends the use of all lowercase with underscores between words, yet uses mixed case as well in examples (PHP Coding Standard, 2003). Some texts utilize underscores between words in their program examples although they do not specifically advise a particular standard. The Sun website also advises not to start variable names with underscores or the dollar symbol.

Some sources recommend appending a data type or other meaningful character to a name, while others do not. There is no indication on the Sun website that variables or other data names should include characters denoting their data type. Oracle says that they are “sometimes helpful” and prefers suffixes rather than prefixes (PHP Coding Standard, 2003). However, Microsoft has extensive lists of object, menu, and variable abbreviations to use as prefixes. It also recommends appending the scope at the beginning of names in larger programs, using “g” for global, “m” for module, and no prefix for procedure-level, which is the default. An example is $mdblQuantity$, which would be a module-level variable of type double. Interestingly, Oracle’s standard for PHP is the only source that recommends all capitals and underscores for global variables. Other types of characters to append include qualifiers, (that further describe data by appending words such as $Min$, or $Last$) and base types, which are abstract data types such as $wn$ for window, $scr$ for screen region, and $pa$ for paragraph. The Hungarian convention recommends that base types be at the beginning of a name and qualifiers at the end (Simonyi, 1999). It is interesting to note that the recommendations for prefixes in the Microsoft Visual Basic 6.0 Programmer’s Guide differ from the website in that the website uses the Hungarian format for variable prefixes, which are usually only one to two characters long, while the book uses three characters.
There is general agreement in the literature about naming constants. Nearly all of the texts recommend naming constants with all capital letters and separating words with underscores. An example is `ROW_MAX`. One exception to this is the Microsoft Programmer’s Guide that recommends that constants be in mixed case and follow the same guidelines as variables. However, this is not what the Microsoft website recommends. Most of the texts also say that prefixes on constants are optional and can be helpful.

A few of the standards guides provide guidelines for the overall structure and appearance of code. Sun advises that a programmer avoid wrapping lines (expressions that do not fit on one line of code) by breaking after a comma or before an operator. Sun’s website, as well as the company Programming Research, advises that one line of code should not exceed 80 characters. Also, Sun recommends that a programmer indent four to eight spaces, while Microsoft believes one should indent two to four spaces because more than that is unnecessary and causes unnecessary statement hiding through truncation. Fewer than two is not effective in reflecting logic nesting (Visual Basic Programming Conventions, 2004). Oracle recommends using four spaces at all times, otherwise the code is hard to read. None of the other texts had any specific recommendations for how much to indent. The Georgia Institute of Technology (GIT) published its internal standards, which require that programmers use white space to distinguish between important blocks of code (Standards & Guidelines, 2003).

Another programming standards topic, discussed in only a few of the texts, is commenting. The Georgia Institute of Technology addresses comments that should be in the header. It requires name of module, author, purpose, restrictions, and revision history (which should include date, name, and any comments). The modification comments inside the code should include the author and date. It also states that “comments should be used at all levels” (Standards and Guidelines, 2003). Consistent with GIT, the other literature states that comments above a block of code should describe the function of the code (meaning what it does), not the implementation (how it does it). In addition, anything that is not clear should also be described in comments. The Microsoft Visual Basic 6.0 Programmer’s Guide advises that every procedure header should include purpose, assumptions, effects, inputs, and returns. It also states
that “every important variable declaration should include an inline comment describing the use of the variable being declared” (Visual Basic 6.0 Programmer’s Guide, 2004, 878). However, it also says that names of variables, procedures, etc. should be clear enough so that comments are not necessary for the most part. CodingStandard.com also mentions that if a variable declaration is not clear, then you should add a comment. Oracle’s standard for PHP says “at every point where you had a choice of what to do place a comment describing which choice you made and why” (PHP Coding Standard, 2003).

There are a variety of suggestions for programming standards in literature today. Many of them vary according to the language used, although there are some similarities across languages. Some of the resources are quite strict and detailed in their standards; however, most of them maintain that they are only suggestions and are not required.

Research Questions

This research project, addresses several questions of interest:

- Do programmers use the same programming standards as those recommended in popular literature?
- Do companies enforce standards or do employees generally have freedom when coding?
- Are smaller companies more disciplined than larger ones, or vice versa?
- Are companies with standards in place more disciplined than those without?

Methodology

Interview Questions

A survey and interview draft were written and revised to obtain a full array of questions. The survey was tested in a practice interview with a programmer working in the IT department of a local company. Further modifications were made to the interview questions after doing the practice interview.

The survey consists of 34 questions, divided into four sections: System Architecture, Coding Structure and Documentation, Naming Conventions, and Software Development. The System
Architecture section provides information on what environment and languages the company might work with. The Coding Structure and Documentation section includes questions regarding use of standards, white space, documenting revisions, and frequency of comments. The Naming Conventions section inquires about how the programmer names variables, constants, classes, tables, etc. It attempts to compare the interviewee’s naming conventions with those in the literature on standards. The Software Development section asks how the company develops software, such as in-house or off-the-shelf customization, and whether the company requires its outsourcers to follow standards.

All of the interview questions were qualitative and most were open-ended. Some questions asked the interviewee to give specific examples, such as how they name constants or what procedures they require for revisions.

To select the candidates for the interviews, a list of programmers and IT managers representing a wide variety of business sectors was obtained from the CIS Department Advisory Council. The council is comprised of company employees who work closely with CSU. The organizations represented both small and large firms, internal and external developers, and several different industries.

Results

The results are based on ten qualitative interviews, one for each company, and the surveys completed by the participants. The participants work for national, international, and local companies. While the sample population was small and limited to the local area, the results were consistent with anecdotal evidence from working students in the classroom and previous industry experience from several faculty. Similar to the published literature, there were several areas where the respondents differed in their opinions and a few where they were in general agreement.

Six companies that participated in the study follow an industry standard, although some of them do not follow it strictly or have modified it to fit their own style. The standards followed include Sun’s standard for Java, Microsoft’s standards for Visual Basic and Access, Oracle, RUP, and CMM. Four
companies said they do not follow any industry standard (Figure 1). Half of the companies also maintain a standards document in-house for new or current employees to use.

The sampled companies use a variety of languages and all of them use multiple languages. They include Java, Visual Basic, PLSQL, Oracle, Perl, C#, C++, and Cold Fusion, as well as others. In general, employees have freedom to choose both an internal and external documentation style; however, they are often expected to stay within certain guidelines or templates. Nearly all of the employees said that they either usually or always conform to one certain programming style. One manager of a consulting firm said the employees seldom conform, yet this is because they adapt their code to fit the client’s style. Two interviewees said there are never any circumstances under which they do not conform, while most admitted they do not conform when they believe no others will see the code, when it is easier or better another way, or when they are under time constraints.

When asked which was more important – performance or readability - four chose performance, three chose readability, two said both are equally important, and one respondent had no answer. (Figure 2). All of the employees encourage structure through indentation. Most companies have a standard for indenting, some of which are stricter than others. The standards range from two to five spaces, mostly favoring the shorter side. The average of the ratings on use of white space was “moderate.” Few erred on the minimal side and some preferred high to increase readability. Two of the three employees preferring high use of white space also said their company values readability over performance.

Nearly all of the interviewees reported that they use comments either often or very frequently. All of the respondents use comments above a block of code as well as inside. Most added that these comments address the function (or the what) versus the implementation (the how), unless the implementation is not clear or is hard to understand. Also, nine respondents noted that their company requires certain documentation in the header of the code. Most of them require roughly the same information, which includes author, date, program name, and a modification log. Other items include author description, request ID, version number, systems affected, inputs, and review initials.
Similarly, nine companies have a policy for documenting revisions to code. The policies range from updating the block comments and header to adding detailed information such as author, description, systems affected, and inputs. Most of the interviewees utilize configuration management and/or version control libraries in their company.

When asked about naming conventions, five interviewees do not use underscores, while the other half does use them, although one of them only uses them in UNIX (Figure 3). The majority does not use any other symbols when naming variables or other data; however, one person uses dashes in COBOL since COBOL does not allow the use of underscores. Eight interviewees use mixed case when naming (Figure 4).

Six of the interviewees say they add characters denoting data type, although two do not use them consistently. One person said he generally does not do this because the variable might be reassigned and the programmer should not have to change the name. This same person works for a company that uses the Java standard (which suggests specifying data type), but has adapted the standard for their own needs. The majority of the respondents name constants the same as other variables. Interestingly, the eight companies that do not follow any industry standard name constants the same as other variables. The two companies that distinguish constants from other data name them with all capital letters and underscores between each word, just as the literature recommends. One employee added he does not use prefixes for constants.

There are a variety of ways that employees name tables. They include using all capitals, underscores, and prefixes. The programmers are not consistent in these names either. Columns in a database table are often named the same as the data name and follow the same rules. For example, if the software (such as Microsoft Access) allows the programmer to change the viewable name, or “caption,” of the data when it is shown in the table column, the programmers usually do not do this. Two employees use underscores and one sometimes uses prefixes for column names. All of the companies try to distinguish between tables, queries, and views by adding characters denoting type. However, there are a variety of ways that they add characters. Two of the companies place “vw_” at the beginning of views
and one places a “v” at the beginning. Another employee adds “.v” for views or “.SQL” for queries at the end of the name. Another method used is placing “tbl,” “qry,” or “lkp” (for lookup table) at the beginning of the name, as some of the literature recommends. For those companies which do object-oriented programming, two employees said they name objects the same way they do other variables and they start classes with a capital letter (e.g., MyClass). One respondent said they name both objects and classes in the same way as other variables, and another interviewee said the naming conventions changes depending on the developer.

Nearly all of the employees use abbreviations in their code. One said his team rarely uses abbreviations unless it is an industry common term and another responded that they never use abbreviations (although this data came from a manager versus a programmer, thus the accuracy may be questionable). There are no company guidelines for abbreviations that the employees are aware of. One interviewee said the only guideline they have for abbreviations is that the abbreviations make sense. When asked if the employees prefer longer, more meaningful names or shorter and more concise names, the majority of them preferred longer, more meaningful names. Five out of the nine respondents to this question prefer longer names, while two of them concluded that a balance is the best. Surprisingly, only two people preferred shorter, more concise names (Figure 5).

There was a slight variety in how the employees abbreviate the word “index.” Most that do abbreviate it shorten it to \textit{idx}, while two sometimes use \textit{indx}, and one sometimes abbreviates it \textit{i}. Three of the companies said they do not or prefer not to abbreviate the word “index.” Most of the interviewees abbreviate the word “number” as \textit{num}, however a couple of them abbreviate it \textit{no}. For temporary variables, nearly all of the employees use a single character at least sometimes and for most of them, this is the rule or norm (Figure 6).

**Discussion**

There were some overall trends and implications of the data. Even with a myriad of vendor-recommended and published programming standards, conformance was highly variable. The company
that was the least strict on header and revision documentation and that used shorter names was one of the smallest companies with only three developers. In their situation, each developer had freedom to choose his or her own style of documentation, although they were expected to follow loose standards.

Also, the next two smallest companies (of size 0-2 and 4) do not follow any industry standard. This seems to signify that smaller companies are less disciplined than larger ones. Furthermore, the largest company, of approximately 600 employees worldwide, was the strictest in requiring that employees conform to standards and was the only one that responded “no” when asked if individuals have freedom to choose a documentation style. They added that “it depends,” and this is only to allow programmers to accurately address each program in the best way.

Surprisingly, most of the employees prefer longer names. This shows that even under time constraints, employees value the readability and maintainability of code and are willing to take the time to use more meaningful, descriptive names.

The companies that follow industry standards do adhere to them for the most part. In general, they follow the rules for indentation, comments, using capital letters for new words, appending the data type, distinguishing between tables and queries, and using single characters in loops. However, they are not very consistent or precise in the manner in which they distinguish between tables and queries, etc. The only area where a majority of them do not follow the industry standards is naming constants.

**Recommendation for those teaching IS classes**

After reading the literature and analyzing the results of the interviews, some general recommendations can be made that cross multiple platforms. First, students should follow the most accepted industry standard for the language they are using. Readability is critical and seems to lead to better quality of code. However, students should also learn how to improve the performance of a program. To improve readability and encourage structure, students should use a moderate amount of white space. Too little makes it hard to read the program and too much wastes space and is unnecessary.
“Moderate” can be defined as using white space between procedures or blocks of code and adding in white space where necessary for clarity.

Both the standards and the interviewees concur that it is good practice to indent. There is a good amount of variation as to tab settings, but the important point is to use nesting and to make the program clear and readable. Students should avoid wrapping lines, as advised in the standards.

Another universal way to improve comprehension of code is through commenting, cutting across all languages. Use of comments is valued by companies and it is important for students to know and follow the best practices. Comments should be used often throughout the code, although “very frequently” is not necessary. Comments should be placed above every block or procedure of code and should describe its function. More comments may be used inside the block to describe function. If the code may be difficult to understand, implementation-specific comments should accompany the code. The comments in the header should include author, date, program/assignment name, description, and any other information each teacher requires. If any assignment requires students to modify a prior program, comments should be updated if necessary above the block of code and a modification log should be maintained in the header, including the date and changes that were made.

It is recommended that students name variables, data, methods, etc. using mixed case since most of the companies do this and the industry standards recommend it. If the environment has an industry standard that favors underscores over mixed case, then students should follow this format. Otherwise, no other symbols or spaces should be used. Because industry standards recommend appending the data type in the name and most companies also practice this, this should be part of best practice. For constants, although most of the companies name them the same as other variables, students should follow the industry standards, which is all capitals and underscores between each word. It is good practice to distinguish constants from other variables. Similar to the companies and standards, prefixes are optional.

Column or attribute names in a database should be named the same as data. Also, students should add characters distinguishing between object types, i.e., tables, queries, reports, etc. The student should follow the industry standard for the particular environment they are working in. For example, in
Microsoft Access students should use the prefixes of “mnu,” “frm,” and “rpt.” For object-oriented programming, all classes should be capitalized. Objects need not be capitalized.

Abbreviations are generally acceptable; however, students should follow the recommendations by authors such as McConnell on guidelines and pitfalls in abbreviating (2004). In particular, abbreviations should be mnemonic and clear. The word “index” should not be abbreviated because idx is not mnemonic and indx leaves out only one letter, which is not recommended. It should only be abbreviated to i in the case of a loop index. Whenever abbreviating the word “number,” it should be shortened to num. Use single characters for temporary variables or loop indexes that are used within a few lines. This makes it obvious that they are temporary and will not be used outside of the local scope.

Although most companies prefer longer, more meaningful names, these can be tedious to type and can lead to errors if the name is used frequently. Medium length descriptive names are preferred. Shorter names can be unclear, so students should be sure to name data intuitively if they tend to make shorter names.

**Further Work**

This paper represents a basic starting point for studying coding conventions and was intended to be used as a tool for developing coding conventions for students to follow. Interesting further work could include conducting a large survey of a wide variety of companies to determine which languages they use and which coding conventions they follow within each language. It would also be interesting to determine the types of organizations who have published coding standards for the entire company versus standards which are used within small groups of programmers within the company.

**Conclusion**

Clearly, it is important for programmers to practice and appreciate best coding practices. They have been developed and revised for several years and make code more clear, readable, and uniform and may lead to better quality systems. However, the standards tend to be specific to a language, vendor, or
platform. Therefore, a document that communicates best practices as it incorporates the predominate vendor standard would be effective in CIS departments. Both faculty and students would benefit from improved readability and adherence to standards.

Most companies seem to follow an industry standard regarding software development; however, many allow their employees some degree of freedom. The standards recommended in this document should help students conform to accepted standards and adjust more easily upon entering the corporate world.
References


Brown, P., *NET Programming Standards and Naming Conventions*


http://freshmeat.net/articles/view/139/


Figure 1. Interview Question #3: Does your company follow an industry or vendor standard for software development?

Figure 2. Interview Question #9: When considering program functionality/performance versus readability/maintainability, which is more important within your organization?

Figure 3. Interview Question #18a: As far as variable/constant/data naming conventions, do you use underscores?

Figure 4. Interview Question #18c: As far as variable/constant/data naming conventions, do you use capital letters to denote new words?
**Figure 5.** Interview Question #25: Do you prefer longer, more meaningful names or shorter and more concise names?

![Pie chart showing 80% for longer names, 20% for shorter names.]

**Figure 6.** Interview Question #28: For temporary variables, do you ever use a single character?

![Pie chart showing 56% for longer, 22% for shorter, and 22% for balance.]

![Pie chart showing 89% for yes, 11% for no.]

**Figure 6.** Interview Question #28: For temporary variables, do you ever use a single character?