Managing Computing Standards in a Distributed Environment Using the World Wide Web

Kay Keskinen
Systems Analyst
University of Idaho
Computer Services
Moscow, ID 83844-3155
Voice: (208)885-2076
FAX: (208)885-7539
kay@uidaho.edu
http://www.uidaho.edu/~kay

University of Idaho
Moscow, Idaho
Enrollment: 11,100
Public
Four Year Institution

The University of Idaho has implemented university-wide computing standards that apply to all administrative application programs that run in a production environment. The World Wide Web is used to distribute those standards, download forms used in change management, submit requests, and more. Going from a centralized computing staff to one where developers are dispersed throughout the campus required standards approved by all university vice presidents and implemented for all situations.

This paper describes the background of the standards, its several major portions (authority, change management, C programming standards and guidelines, library of common C programming functions, the peer review process, and naming conventions), and its application using the web. The presentation includes a demonstration of the web pages.
Managing Computing Standards in a Distributed Environment Using the World Wide Web

Introduction

The University of Idaho has recently migrated from its more than thirty home-grown on-line applications to a purchased set of administrative software packages. This migration process has resulted in program-developers now being distributed throughout the campus, in client offices as well as Computer Services, instead of being centralized in Computer Services. In order to assure that all program development for administrative applications meets the same quality requirements when going into a production environment, a Standards Committee was formed in the summer of 1995 to create the foundation of computing standards. I have served as a committee member since its inception and also as the committee chair since September of 1995. The university vice presidents approved the standards, which encompass the entire university’s production program development. This paper describes the standards and their implementation through the world wide web.

University Overview

The University of Idaho is a publicly-supported comprehensive land-grant institution with principal responsibility for performing research and granting graduate degrees in Idaho. Primary areas of statewide programs are agriculture, architecture, engineering, forest and wildlife, liberal arts and sciences, mining and metallurgy, foreign languages, and law. Additional University responsibilities include programs in business, economics and education as well as the regional medical and veterinary education programs in which the state participates.

The University of Idaho is located in Moscow, in the northern part of the state, about 150 miles south of the Canadian border. In the Fall of 1996 the University's enrollment was approximately 11,100. The students come from all states and over 80 foreign countries to participate in 142 fields of study. Programs are offered on-campus and at remote sites throughout the state. Most locations are directly tied to the University's communications network for timesharing and teleprocessing services. Additionally, the university offers technical assistance and access to the Internet via our data communications lines to grades K-12 throughout northern Idaho. We are working with the State of Idaho to create an integrated statewide network for voice, video, and data communications.

The administrative computer environment at the University of Idaho consists of three Sequent machines which use Dynix/ptx, a version of UNIX, with a Sequent S2000 model 750 (with eight Pentium 60 and 14 486/50 CPUs, 1.5 gigabytes of memory and 180 gigabytes of disk storage) used primarily for production applications. The five modules of the Banner administrative software (Alumni/Development, Finance, Financial Aid, Human Resources, and Student) were
purchased from Systems and Computer Technology Corporation (SCT) and run on an Oracle database as does the Facilities Management application licensed from Prism Corporation.

**Why Have Standards?**

When the university purchased the Banner software in the summer of 1993, most of the legacy administrative applications were written in COBOL, and Computer Services staff wrote all administrative production programs. At that time there existed some COBOL programming standards which were not formally enforced; many routines (modifying or writing a new program, for example) were done by common practice, but written guidelines were scarce. With client offices now responsible for their administrative module, functional experts began to develop and write their own production programs that would run against the production database. Customers used a mix of programs and procedures that were either purchased or developed in-house. In order to minimize training, we wanted both types of programs to be executed similarly; in order to be consistent in the way we developed new programs among the various modules, we wanted to have change management procedures in place; and in order to ease the maintenance of the in-house software and to provide the customer with a working product, we wanted the programs to be “well-written.” We saw programming standards as the way to achieve these things.

**Creation of the Standards**

In July of 1995 the Computer Services associate director, who was also serving as the project manager for the implementation of the Banner system, created a Standards Committee to investigate and develop computing standards to meet the needs of our university. This committee consisted of six members: three from Computer Services (representing three different sections of Computer Services), two from different customer offices (Business and Accounting Services and Financial Aid), and one from the academic area of Computer Science. This group used its expertise and experience (together, the six had about 100 years of experience in computing) to suggest areas that the standards should cover.

Meeting weekly, the group developed the broad areas the standards would cover and by the end of that summer had the basic outline of the standards; we asked the associate director to review the committee’s direction with the university vice presidents. The vice presidents authorized the committee to continue and to develop standards for any program that would go into a production environment.

With the broad outline and approval from the vice presidents, the committee worked on the details of the various portions of the standards. The key problem of presenting available and up-to-date standards to the campus community was solved by the world wide web--the entire community could read the most recent version at any time.

**Distribution of the Standards**
After we created a web “home page” directory for each of the six committee members to access and update the various standards documents, we began building the web documents. Committee members who did not know how to create web pages in hypertext mark-up language (html) soon learned the basics; drafts of various portions were distributed among the committee members to work on outside of the actual meetings. While the committee completed its initial work, only those who knew the exact “URL” could access the web standards; this allowed the committee to develop the web pages without too much public scrutiny.

Once the standards were in a “working draft” status in February of 1996, Computer Services created a link to them from the Computer Services web home page under “Objectives.” We added a “last date revised” to the top of each section of the standards web pages so that readers could easily identify new or updated sections. We attempted to keep as much continuity as possible among the standards web pages, but because of the many people creating the pages, some differences do exist.
There are currently four chapters on computing standards: an overview, programming standards, implementation standards, and naming standards (see illustration above). The Overview has sections describing the scope of the standards, the various mandate levels, and the authority of the standards. Programming Standards includes information on C and our repository of C functions. The chapter on implementation discusses our various databases, our change management procedures, and the peer review process. Naming standards, the last chapter, describes the procedures we have for names as well as the UNIX directory structure. The main web page has a link to send e-mail to the standards committee members (all members receive copies of the mail); each link to minor sections has a link back to its major section. Figure 1 shows the current main standards web page, entitled “Computing Standards Home Page for the University of Idaho.”
Overview of Standards

The first web chapter presents an overview of the standards, and its first section describes the scope of the standards, that they apply to any locally-developed software process that affects university assets. Modifications to the purchased Banner software, or customizing Banner software to better meet our university’s needs, do not come under the auspices of the standards, since we are usually only modifying a small portion of the code; requiring the entire program to comply with our standards would burden any developer who makes the initial modification or who must apply our customizations to future releases.

The overview discusses the mandate levels of the standards, which have two expected levels of compliance—a standard, to which compliance is required, and a guideline, where compliance is recommended since past practices have proven that the action has merit or because compliance contributes positively to the goals of the standards.

The last section of the overview deals with the authority of the standards. In order to best use our limited personnel resources, the vice presidents of the university have endorsed the efforts within Computer Services to standardize access to university data and have directed that all user communities comply with these standards. The overview also contains a mission statement of the standards group: to construct a framework within which widely disparate groups on campus can take maximum advantage of an integrated database with module re-use an explicit benefit. The four sub-goals of the standards committee are to publish the standards in an easily-accessible manner, to establish standards and guidelines for more prudent and comprehensive user testing, to provide a procedural mechanism for reviewing and storing code, and to teach this methodology.

Programming Standards

The second chapter of the Standards is Programming Standards, which has two areas: the actual C programming requirements and the central repository of functions available to C programs. Other languages besides C may also have programming standards in the future.

C Standards

We adopted the initial C programming standards from those SCT used for their C programs. The Standards Committee reviewed the list from SCT; we modified some items, deleted others, and added a few that committee members from their years of C programming experience believed would be beneficial. There are two categories of C regulations, standards and guidelines. Standards are rules that must be adhered to, while items listed in guidelines are prohibited in a program without good reason (the developer has to defend the use of these techniques in the guidelines).
This is a sample of the C Programming Standards:

Standards: Programs which meet UI C Standards will...

... have a beginning comment header documenting the program.
... not have hardcoded filenames.
... produce no errors or warnings at compile time.
... not modify any string literals.
... use UI functions or macros where functionally appropriate.
... close all cursors in a Pro*C program.
... etc.

Below is a sample of some of the C Programming Guidelines:

Guidelines: For a program to meet C standards, the author will be called upon to defend:

... the use of any goto statement.
... the use of any platform-specific functions.
... every global variable.
... use of temporary tables.
... all SQL joins, views, unions, and subselects in a Pro*C program.
... etc.

Each standard and guideline then has a link with details explaining and rationale defending that standard or guideline. A few standards include an example to help the developer utilize the standard. Appendix A, titled “C Standards for the University of Idaho,” shows the initial C standards web page.

Central Repository of Functions

The second area under Programming Standards presents the useful C routines that are available for UI developers to use in their own programs, thus keeping errors down and making changes simpler. The web pages explain where the repository is, how to use the functions, and how to add new routines to the repository. The library of functions is designed to make programming in C easier and more efficient. All library functions that use SQL have been checked to ensure that they use Oracle indexes when possible, in order to guarantee maximum program efficiency.

Each library function has a link to additional information that describes the function, tells where to find its source, explains the syntax of the function, shows parameters for the function, presents what the function returns (shows under what circumstances it returns a 1 or a 0), illustrates use of the function with an example, and has explanatory notes about the function.

Examples of common functions are to convert a date, add dashes to a social security number, retrieve titles for accounting codes, get rollup levels of accounting codes, get one of the many keys a person can have in the database (SSN, student ID, Alumni ID, etc.), get an address for a person (including pulling addresses from a hierarchical list of address types), and validate access to data in a particular accounting code level.
These functions are very valuable when they are used for routines common to many programs. Customers who request a report program, for example, do not always understand the technical issues behind retrieving data from the database. They know only that they want the current “mailing” address for some selected people, and, if that address does not exist, they want the “billing” address. They do not know how the database utilizes address “from” and “to” dates nor that there is a status flag for each address which marks the address as active or inactive. The developer may not know how the address logic works either, but the address function does know; using it correctly will retrieve the address the customer needs. Additionally, the function uses a table index, so the address is retrieved efficiently as well as correctly.

The web section devoted to the library of functions also includes a section on SQL optimization techniques, for use by developers when they must create their own code to retrieve data from the database outside a library function. The SQL section discusses when to optimize, how to “know” the data, and what the various types of Oracle “hints” are that can be used in any form of PL/SQL (use of indexes, range scan, and full scan), including an example of how to use a “hint.”

**Implementation Standards**

The third chapter, Implementation Standards, has three sections: database instances and permitted usage, change management, and peer reviews. Now that departments all across campus develop their own software, creating standards for implementation that would work for all departments and developers took much time, effort, and cooperation. We wanted common enough implementation procedures to guarantee that software worked and was tested before being implemented but we also wanted to permit enough flexibility for departments to place their own requirements on their software development procedures.

**Database Instances**

The first area in Implementation Standards, database instances and permitted usage, describes the proper use of each of our four databases, who “owns” it, and what types of update and access are allowed for each. For example, our training and development databases are separate from the production database.

**Change Management**

Change Management, one of the larger sections of the standards, describes how one initiates a software “change,” such as a new report or on-line form or a modification to existing software. We use change management to ensure program efficiencies, to allow for backup coverage during developer absence or assignment change, and to meet the requirements of our external auditor. The standards address the four basic steps we use to create or modify software: project initiation, development, testing, and implementation.
Initiation. A project begins at the university of Idaho with a customer drafting a requirements document, the form for which is available on the web. On the form, the customer describes what is needed, the benefits, timing, priority, etc., then prints it there or downloads a text version of the form and completes it using a word processing package. The requirements statement then goes to the manager of the resources requested to complete the work; depending on what technical resources are being requested, the approval of the requirements document can come from one of the project leaders (we have one for each Banner module) or from the Policy Group of directors representing departments throughout campus. In our case, if resources come from within a department, the project leaders approve the request; but if resources come from Computer Services or cross project boundaries, the larger Policy Group approves the project.

Development. Once the project is approved, time and resource needs are estimated, more detailed analysis is done, and one or more developers are assigned to the project. Every program that is developed initially at UI or is modified from an original Banner program has a program folder with various materials in it. If the project is a modification or customization of existing software, the program folder is retrieved in order to get past documentation; if this is a new project, a new program folder is begun. We have links on the web pages to describe the contents of a program folder as well as other aspects of project development. If this is new software, the developer requests a name using a web form that sends e-mail automatically to someone who approves or denies use of the name. (More about this process and naming conventions will be discussed later under the Naming Standards portion of the standards.) The developer codes and tests the program; if it involves modifying Banner software, the affected lines of source code are marked such that any changes are indicated with a special identifier, usually “UI Mod,” along with an explanatory comment, helping future developers apply the same modifications to future releases of the Banner software.

Testing. Testing begins with the software placed in a test environment so that customers can try it out. Once the program appears to meet the user requirements, we conduct a peer review of the code to ensure that the program meets the standards. In the peer reviews a facilitator, recorder, two to three peers, the customer, and the developer review the initial requirements, program source, test cases, and test results. The purposes of the peer review process are to increase the likelihood that the software will work correctly the first time and to provide on the job training and rapid dissemination of information among developers. The focus of the review is correctness of the solution; efficiency is taken into consideration only if some doubt arises about meeting schedules. At the conclusion of the review, the customers and peers vote on whether the program is ready for implementation as presented, needs minor modifications (no additional peer review), or needs serious enough modifications to require another review before it is ready for production. The facilitator votes only to break a tie. The recorder then distributes electronically the notes of the review along with the peer review team’s recommendation. If the program does not need another peer review, the developer prints a copy of the review notes for the program folder to indicate that the program is ready to be implemented.
Implementation. Once the software has passed the peer review, it goes into the fourth phase of project development, actual project implementation. The developer completes the program folder, which now includes the current source listing, test cases, sample output, peer review report, requirements document, any specifications, customer signoff, developer signoff, and implementation details. Both the completed folder and a transfer sheet which tells where the software is, where it is to be transferred, and why it is being transferred go to the Computer Services manager of Applications Development, even if the software was written by personnel outside of Computer Services. The AD manager then reviews the folder for completion, signs it if it is complete, gives the transfer sheet to the database administrator, and updates the web page that lists completed projects.

The final area in Implementation Standards is a description of the peer review process, which was described above under Change Management. Change Management has a link to the peer review web pages.

Naming Standards

The fourth, and last, chapter of the Standards is Naming Standards, which has two divisions: file names and UNIX directory names. We apply standards to file names for several reasons: so that developers, particularly in different groups, do not assign the same name to different files; so the name is mapped to a description of the function being performed; and so we can avoid unnecessary restrictions on developers. Banner software is identified by a unique first letter (student applications begin with an “S,” for example); and common file extensions and their applications are listed in the standards.

Banner-related software goes into a name registry, with requirements to use a “W,” “Y,” or “Z” as the second letter of the name in order to indicate that this is UI software in Banner. The current name registry appears on the web; a developer completes an electronic form on the web to request a name for the registry. If the name meets the standards, the chair of the Standards Committee sends e-mail to the developer approving the name; if it does not meet the standards, the chair sends e-mail with the reason for denial included. The person who receives the electronic request for the use of a name then updates the web name registry with the name, description of the software, and the date. As we become more proficient with web forms and automatic updating, the approval process, too, may become more electronic.

The second division of Naming Standards is the UNIX directory names, a description of how the Banner software and Banner-related software are organized by directories on our UNIX system. We have modified these file systems slightly because they grow when we place more Banner and UI Banner software in them.
Conclusion

These computing standards for administrative applications will continue to evolve as new issues arise in university computing, but we’ve already seen numerous benefits from their use at our campus. Their implementation has resulted in software that is easy to read, maintain, and use. The peer review process in particular has been beneficial, both to developers and customers. Because of the peer review process, developers have learned valuable C programming techniques, and customers have felt they were more a part of the process. The ability to use the UI library of functions has meant that programs have reliable processes to perform common program functions. Developers can use a function instead of writing their own code, which makes programming faster and programmers less prone to errors. SQL optimization has benefited the customers who have to wait for the program or process to run to get a report or data updated. We have modified some Banner software to use Oracle tables more efficiently; one specific program that formerly took more than a day to run now runs in a few hours. Other more complicated table processes have been reviewed and modified for optimization; this, too, has resulted in shorter run times. The final and most significant benefit has been that we meet our external auditor’s recommendations, and when the external auditors are happy, so are the administrators.
Appendix A

C Standards for the University of Idaho

Date last revised: October 25, 1996

(Discussion of scope of C standards within the University...)

C Standards are divided into two major categories: Standards and Guidelines. Standards must be followed. No exceptions are allowed. Guidelines are individual items which should not be allowed in a program except for good reason. All programs for which these standards are in effect must pass through a peer review process. During this process, programs which contain code which violates guidelines will have to be defended. Look to the Peer Review Process for more in-depth discussion of this topic.

Standards: Programs which meet UI C standards will...

- ...have a beginning comment header documenting the program.
- ...meet ANSI 2 standards for function prototyping, with descriptive names for parameters.
- ...not have hardcoded filenames.
- ...have a header file containing all prototypes, typedefs, structure definitions and defines.
- ...produce no errors or warnings at compile time.
- ...allow the -t switch if it is a PRO*C program.
- ...not modify any string literals.
- ...use UI functions or macros where functionally appropriate.
- ...use macro definitions for a specific process, where appropriate.
- ...have main as the first function, if the program has a main.
- ...initialize all variables right before they are used, not when they are defined or declared.
- ...have comments identifying all blocks of codes and functions.
- ...identify all macros and type definitions by uppercase.
- ...not use closing semi-colons (;) in macro definitions.
- ...not use a cursor for a known single return select in a PRO*C program.
- ...close all opened cursors in a PRO*C program.
- ...delete all rows for their job from GENERAL-G/BPRUN if the job was submitted through Banner.
Appendix A (continued)

* ...write all parameters passed to a program through job submission in Banner to a known destination, preferably standard error.
* ...follow every EXEC SQL DECLARE, SELECT, UPDATE, INSERT, DELETE, CREATE, FETCH and PREPARE statement in a PRO*C program with a POSTORA statement.
* ...be consistent in style throughout the entire program.
* ...ignore any purely stylistic guidelines in the program wherever it interferes with the readability of the code.
* ...pass all stylistic standards and guidelines if it conforms to the db standard.
* ...indent any members of complex data structures or typedef statement.
* ...not ignore returned status of key functions.
* ... print a unique error message explaining an error situation, including relevant information.
* ... release any resource explicitly obtained.
* ... compile with a standard makefile.

Guidelines: For a program to meet UI C standards, the author will be called upon to defend...

* ...the use of any goto statements.
* ...the use of any platform-specific functions.
* ...the use of the -x switch.
* ...every global variable.
* ...the readability of the code with respect to commenting.
* ...the descriptivity of variables.
* ...use of temporary tables.
* ...not putting the program in job submission in Banner.
* ...all SQL joins, views, unions and subselects in a PRO*C program.
* ...all debug code left in a program at transfer time.
* ...the use of any true variable with constant values that isn't using C's # define to set up the variable with its value is never changed.
* ...not using standardized reporting procedures.

Link to UI Standards

Direct electronic mail to standards@uidaho.edu