



3000 Landerholm Circle SE • Bellevue, WA 98007-6484 • www.bellevuecollege.edu

IT Security Standard:

Application Development

Introduction

This standard defines the steps needed to implement the Bellevue College IT Security Policy with regard to application development, with an emphasis on web applications. The standard will be reviewed on an annual basis or when changes are necessary.

Scope

In support of the Bellevue College IT Security policy, this standard defines specific procedural and configuration elements for developers to use in application development. This includes development and implementation of both web-based and non-web-based applications.

This standard addresses the security sensitive aspects of the application development process from concept through deployment and eventual retirement of the application. It applies to all applications developed by Bellevue College employees, contractors or purchased from a third party. It is recognized that Bellevue College may have less control over the techniques used in applications purchased from a third-party, but the requirements of this standard will be considered during purchasing decisions.

Exceptions

A variety of exceptions to this standard may be expected. These exceptions, when granted, will be documented in either a platform specific standard or in a memo documenting the exception. Exceptions may be granted by the Bellevue College IT Security Administrator, the Dean of Information Resources (IR), or any IR director authorized by the Dean. Copies of all documentation regarding exceptions will be kept on file with the Bellevue College IT Security Administrator. This documentation will include:

1. A detailed description of the exception.
2. A description of why the exception is necessary.
3. A risk assessment by the Bellevue College IT Security Administrator and/or the Dean of Information Resources, or designee.
4. A description of the compensating controls that are in place to mitigate risk created by the exception.

Business Impact, Risks, Threats, and Vulnerability Analysis

Application software has become critical to the daily business and operations of Bellevue College. While not true across the board, many of the college's business functions would be severely impacted by a disruption in the services provided by these applications. The main threats to IT security at Bellevue College are not directly related to the application development process, but to the products and services resulting from the development process.

The most significant threats related to application development at Bellevue College are:

1. Malicious and/or unauthorized disclosure, modification or deletion of information.
2. Malicious and/or unauthorized disclosure or modification of application data.
3. Denial of service.
4. Compromise of supporting servers.

The primary risks associated with college developed applications are unauthorized access, disclosure or modification of information, and creating vulnerabilities which can result in malicious attacks. This will

likely be used to commit some type of fraud or theft. All of these threats have associated risks: loss of revenue, dissatisfaction to Bellevue College customers and stakeholders, service interruptions, and loss of reputation.

Standard

A. Introduction

1. The security of an application system begins at the point of gathering business requirements and continues through design, development and implementation. Since applications can give a hostile or malicious user access to the Bellevue College network (or other applications), great care must be taken to assure best development practices are observed throughout the development life cycle.
2. This standard identifies best practices, expectations and issues in the development process that have an impact on the security of the data, application, and infrastructure. While impossible to specifically codify in a standard, the principle of keeping the design and code as simple and clear as possible must always be followed. This principle tends to strongly impact the reliability, supportability and security of applications.
3. College applications must comply with all provisions of the Bellevue College IT security standards addressing “Password Management” and “Web Information Accessibility”. Publicly-accessible applications must also comply with all aspects of the Bellevue College IT security standard addressing “Web Space Usage. Additionally, developed applications must be deployed and maintained using the Software Development Lifecycle guidelines as approved and published by the Washington state Center for Information Services (CIS).
4. For clarification, simple web sites are not considered by this standard to be “applications.” Web applications are differentiated from web sites in that an individual is adding or modifying server-side code in a web application, but not when developing or deploying a web site.

B. Information Resources Funded Application Development

1. Bellevue College supports centralized application development through developers assigned to the Information Resources (IR) division. Projects developed by IR generally impact multiple units across campus or have a significant impact on how services are delivered.
 - a. A Development Project Request process for enterprise application development has been implemented to facilitate communication and management of projects.
 - b. In addition to this standard, IR management may also establish additional guidelines, processes, procedures and practices necessary to professionally meet the requirements for enterprise development projects.

C. Self-Funded Application Development

1. Bellevue College also supports application development by campus divisions with the resources to develop their own applications, provided the applications impact delivery of services by the developing unit only.
 - a. A Development Project Request process for self-funded application development has been implemented to facilitate communication and management of projects.
 - b. In addition to this standard, IR management may also establish additional guidelines, processes, procedures and practices necessary to professionally meet the requirements for self-funded development projects.
2. In addition to the criteria articulated in this standard, supplementary procedures and processes are required to ensure the security of the college’s technology resources.
3. All funding for development and maintenance of a self-funded application comes from the individual unit. This includes not only the costs for development, but the subsequent costs for operating and maintaining the application.
2. Restricted Software Exception

- a. Software used by developers includes applications that, by Bellevue College and state policy, have traditionally been limited to installation on servers controlled and maintained by Information Resources.
- b. When a self-funded development project has been approved, an exception to the policy may be granted on the condition that the developer complies with all Bellevue College IT security policies and standards, and meets the following requirements:
 - i. Any installation of server software on a desktop computer is contingent on its use locally only. Permission to use the software will immediately be terminated if it is used to allow any remote user to access files stored on a desktop computer, whether that access is internal or external to Bellevue College.
 - ii. Under no circumstances should a desktop computer be used as a network server or application server providing access to any outside user. All testing must take place in the appropriate network environment.
 - iii. Deployment of any self-funded applications using Bellevue College network resources (servers, network, databases, etc.) must be coordinated through the Enterprise Applications Support (EAS) office, and approved by the appropriate division Vice-President.
 - iv. All requests for software installations on Bellevue College computers will be handled through Request Center.

D. Application Development Procedures

1. All applications must be documented and approved using the Development Project Request Form (Appendix C).
 - a. It is the intent of the form to:
 - i. Ensure compliance with Bellevue College IT Security policies, standards and established development processes;
 - ii. Provide project updates to the i-Bellevue College Steering Committee and IR to facilitate successful deployment and maintenance of the application; and
 - iii. Manage the use of centralized application support resources.
 - b. Formal signature approval by the appropriate division Vice President is required before an application development project is initiated.
 - c. Following VP approval, the form will be routed to the appropriate IR designee for review.
 - d. Enterprise Application Support (EAS) will notify and consult with the following campus units:
 - i. Technology Development & Support
 - ii. IT Security Administrator
 - e. Once all notifications and approvals have been completed, copies of the form will be filed with EAS and the IT Security Administrator. The originating division may retain copies, if desired.
2. All college employees and contractors developing applications must sign an Intellectual Property Agreement Form (Appendix I) and return it to Information Resources, for filing with the IT Security Administrator.
3. All developed applications must be in compliance with applicable Bellevue College Policies and Standards. Application developers must read and comply with all *required* Bellevue College Policies and IT Security Policies and Standards identified in the Security Requirements (Appendix H). In addition, it is the responsibility of the developer(s) to read and comply with any of the *optional* policies or standards identified in that appendix which apply to their project.
4. IR reserves the right to reject or delay deployment of any application which does not comply with all IT security standards and policies, and the right to remove or disable any application which compromises or disrupts campus computing services.

5. Appendix D outlines the Bellevue College Application Planning, Development and Support Process for both IR-funded and Self-funded application development, deployment and support, and is provided as a reference.

E. Bellevue College Supported Technologies

1. Application development must take place on:
 - a. Bellevue College-owned equipment combined within Bellevue College's development environment, or
 - b. Personal equipment
 - i. After initial development, the application must be moved to and tested on Bellevue College's test environment.
2. Bellevue College supports a three-tiered application development and deployment server environment:
 - a. A development server to support initial development (described below).
 - i. Information Resources will maintain a "development server" to which both IR and self-funded developers will be granted rights sufficient to manipulate and edit any database they are using in development of an application.
 - ii. Information Resources' EAS office will be the contact for rights to tables and databases, and will grant appropriate rights to the server.
 - iii. The EAS office will also coordinate updating and maintenance of applications with the division responsible for development.
 - iv. Rights to the development server can be granted to IR developers, Self-funded developers and Contractors/Vendors, provided all Bellevue College IT Security policy and standard requirements have been met.
 - b. A test server to mimic the use of the application in a realistic environment.
 - i. When an application is ready for testing, the developer(s) will notify the EAS office, responsible for coordinating installation of and access to the application and its supporting environment.
 - c. A production server to provide operational access to the finished application.
 - i. When an application has been tested and is ready for deployment, the developer will notify the EAS office, responsible for coordinating movement of the application and supporting environment from the test server into production.
3. NOTE: Bellevue College's main web cluster consists of primary and secondary servers that together provide increased site performance and fault tolerance. These benefits are offset by three challenges:
 - a. Session objects are supported for ASP.NET applications; however, they are not supported for classic Active Server Pages (ASP) session variables because ASP client state is not replicated between cluster members.
 - b. All session objects must be designated "Serializable" in order to conform to the Bellevue College cluster replication mechanism.
 - c. Writable data stores located on the web server are not supported (e.g., Microsoft Access databases, Microsoft Excel spreadsheets, and text/XML files). File/directory writing is also not supported. These methods are not supported on the cluster because files from the primary server overwrite the files on the secondary server during replication.
4. **Supported Technologies:**
 - a. *Web Server Platform*
 - i. Windows 2003/IIS6
 - ii. NET Framework 2.0
 - b. *Application Server*
 - i. ASP.NET
 - ii. Classic ASP

- c. *Data Access Technologies*
 - i. ADO.NET
 - ii. MDAC 2.8
 - 1. ADO
 - 2. OLE DB (provider = SQLOLEDB)
 - 3. No ODBC (provider = MSDASQL)—Microsoft has deprecated this technology in favor of OLE DB
 - iii. MS XML Parser
- d. *Database Server*
 - i. SQL Server 2000
 - ii. Access, Excel, or text file databases are not supported
- e. *Security*
 - i. Windows Integrated Authentication
 - 1. Website access
 - a. This is the preferred method of securing a limited-access website.
 - b. A list of users (Bellevue College faculty or staff), level of access, etc. must be submitted to Enterprise Applications Support so the appropriate groups can be created.
 - 2. Database access
 - a. No SQL Server-based logins—Windows integrated authentication is the preferred method for database access.
 - ii. Anonymous Authentication
 - 1. This is the default (allows public access) but can be turned off.
 - iii. Basic Authentication
 - 1. Caution! Passwords and other data are sent in clear text unless the application is using SSL/HTTPS (see the next bullet).
 - iv. SSL/HTTPS for public web sites is only available on the main web site host (www.bcc.ctc.edu)
 - 1. Additional public certificates can be purchased by the division sponsoring the application.
 - 2. Additional private/internal certificates can be generated upon request.

F. Development Guidelines

1. The System Development Lifecycle guidelines approved by the CIS Board will be followed with regard to all IR-funded applications development, deployment and maintenance activities. In addition, the following guidelines apply to all Bellevue College development:

G. Web Interface Requirements

1. All applications developed at Bellevue College using a web interface (front-end) must follow the requirements and guidelines found at: <http://www.bellevuecollege.edu/webpublishing>. In addition, all HTML will be coded in accordance with the DIS policy “State Standards for Internet Markup Language”, and meet all requirements of the Bellevue College IT security standard addressing “Web Space Usage.”

H. Application Design Practices

1. The project lead will work with project sponsors to collect and document requirements including:
 - a. Identification of sensitive data, and its handling concerns,
 - b. Issues regarding separation of duties that may impact design,
 - c. Special audit and control needs or concerns, and/or
 - d. Other business process concerns that impact security.
2. Developers will peer-review code to minimize security issues.

I. Development Practices

1. Application design and implementation will respect the principle of "least privilege," that is, use the absolute minimum access privileges necessary to perform the task.

2. Developers should use and reuse trusted components.
3. Passwords used to access subsystems and resources will be stored securely. The following criteria will be used:
 - a. Never embed a password directly into the "program" (e.g., scripts, jobs, executables, and configuration files).
 - b. Passwords stored in clear text will be stored as securely as possible via other security layers (e.g., file ACLs - Access Control Lists).
 - c. Passwords for more sensitive resources will be guarded strongly with password obfuscation, encryption, or an appropriate security subsystem (e.g., LSASS - Local Security Authority Sub System).
4. Bellevue College's sensitive production data is frequently used for testing and/or training.
 - a. Printouts with sensitive information will be shredded.
 - b. Any data shared with people outside Bellevue College will be sanitized to protect its confidentiality.
 - c. If data sanitation is not an alternative, the IT Security Administrator and/or Dean of Information Resources will approve the release of data to an entity outside Bellevue College.
5. Source code will be secured in a manner that prevents unauthorized access.
6. The deployment process will assure that only applications that have been tested and approved are allowed to move into production.

J. Application Documentation

1. Attention to IT security issues and adherence to standard security principles will be included in all planning, design, development, and deployment stages of developed applications. This should be reflected in all application documentation. To standardize application documentation, the template attached to this standard as Appendix E will be used. In addition, the standards for Source Code Documentation (Appendix F) and the .NET Naming Guidelines (Appendix G) will be met in all application documentation.
 - a. System documentation will provide a summary of the security-related requirements, how the application was designed and coded to accommodate those requirements, and any significant security features or considerations that might be pertinent to understanding the application and its use.
 - b. User documentation will contain instructions and training materials for proper use of the application, including security features within the system, as well as the manual procedures and best practices necessary to use the application in a production environment.

K. Source Code Management

1. Source code management is an important part of the development process for any project. A source code control application:
 - a. provides regular backups in case of local system crashes or hardware failure.
 - b. allows developers the freedom to make changes without fear of breaking the code base.
 - c. facilitates multiple developers working on the same project.
 - d. provides a structure for troubleshooting and maintaining products.
2. Automated source control tools will be used to ensure the reliability of developed applications and data used in the application. This software must track all versions of every file imported, including change comments, as necessary, and must be compatible with the other development tools/software being used.

L. Input Validation

1. All inputs that are outside the control of the application will be validated.
2. Input validation will test for correct data, instead of testing for the absence of bad data.

3. In addition to input values, numeric ranges and input lengths will also be validated.
4. Data validation will be performed when crossing trust boundaries (“chokepoints”).
5. Any externally reachable function, variable, or method will be validated.
6. Data will only be inserted into the database via approved applications or interfaces to assure that all appropriate edits have been performed.

M. Error Handling

1. In the event of a fatal program error, the application will terminate gracefully and fail to a secure state. The practice of compartmentalizing users, processes, data, and privileges will be used to limit the scope of damage if something fails. As appropriate, the program will assure that:
 - a. Sensitive data is not left in memory;
 - b. Data is not left in an inconsistent state;
 - c. All open files are closed;
 - d. Temporary files are purged;
 - e. Databases and remote computers are disconnected; and/or
 - f. A log is created that contains as much useful post-mortem debugging information as possible.

N. Application Architecture

1. When possible, users will be authenticated and granted access by individual account so that accountability is maintained.
2. Access to specific functions within a program by application user will be configured and controlled by the person assigned responsibility for application security.
 - a. Responsibility for application security rests with the Bellevue College network administrator responsible for the server upon which the application resides. If the application exists on a stand-alone system, Bellevue College IT support personnel may create appropriate access rights at the direction of the network administrator.
3. Where appropriate, the application user's activity will be logged for later review.

O. Naming Conventions

1. In addition to the information included in this section, please refer to the .NET Naming Guidelines (Appendix G).
2. Database Names
 - a. Use title case with no spaces (e.g., WeatherData)
 - b. Don't hardcode the name of the database throughout the application because the name may be changed to better reflect program/department/owner/function
3. Stored Procedure Names
 - a. “usp_” followed by a name that clearly describes the purpose of the stored procedure or what it does. (e.g. usp_GetStudentRecord)
4. Files, tables, stored procedures, etc.
 - a. There are no set rules—just make sure the names are consistent and meaningful.
5. URLs
 - a. The Bellevue College Web Services department assigns appropriate college URLs for websites and applications based on established naming conventions. See the Bellevue College IT security standard addressing “Web Space Usage” for further information.
 - b. Since the URL may not be known when development begins, use relative links within applications.

P. Security

1. Cross-site scripting errors

- a. If the website blindly accepts input then re-displays it to users, there is a cross-site scripting security hole.
 - b. Never trust user input! Always validate input using regular expressions!
 - c. Use HTML encoding (Server.HtmlEncode) when displaying user input.
2. SQL injection errors
 - a. SQL injection is a technique used to exploit applications that build dynamic SQL queries from user input
 - b. Again: Never trust user input! Always validate input using regular expressions!
 - c. Use stored procedures and/or parameterized queries instead of dynamic queries to reduce the chances of SQL injection
3. Trust no input
 - a. Always validate input using regular expressions!
 - b. The point is not to strip out invalid characters, but to only accept valid input.
4. Authenticating a user once at the “front” of the application will not protect the rest of the application—a user can always return to the “protected” pages without authenticating.
5. Obscurity is not security. Simply hiding connection strings, algorithms, pages, etc. does not guarantee they won’t be found by a determined hacker. Use encryption, server-side logic, per-page authentication, etc. to minimize security threats.

Q. Tool Preferences

1. Use ASP.NET rather than classic ASP.
 - a. Choice of languages: C#.NET, VB.NET, J#.NET.
 - b. Session state is supported for ASP.NET on the Bellevue College server cluster.
 - c. The built-in, role-based security makes it easy to specify who has access to the application.
2. Use SQL stored procedures (or at least parameterized queries)
 - a. Stored procedures are reusable.
 - b. Stored procedures help separate the data and business logic tiers.
 - c. Parameterized queries are more secure than dynamic queries because parameters are type checked.
 - d. Parameterized queries are faster than dynamic queries because they are compiled.
3. Use Windows-based security for intranet websites.
 - a. ASP.NET’s built-in, role-based security is most effective when using Windows-based groups. This allows role-based authorization to be managed by Active Directory rather than the application.

R. Active Content and Content Generation

1. Client side scripting (such as JavaScript, Java applets, VBScript) will be used only to enhance the Web user’s experience (such as help, movement between fields, and other aids). In order to be compliant with the Bellevue College IT security standard addressing “Web Information Accessibility”, an application should never rely solely on Javascript.
2. All data manipulation (such as input validation, computation, security tests) will be done on the server.
3. Active Server Page (ASP) and similar server-side page generation software must never store sensitive data (such as database logons and the like).

S. General

1. Remove comments from within the Hypertext Markup Language (HTML), Java script, Java programs, and any other client-side code as they can provide information to a potential attacker. Program version numbers are an exception. This can be accomplished by passing all the code

through a filter that strips comments at the point of deployment. This applies only to Bellevue College-developed applications, not third party applications.

2. The use of code obfuscation for HTML and JavaScript should be used when feasible.
3. Passwords, user IDs (i.e., database login information), and other sensitive information must not be hard-coded into pages sent to the client browser.
4. "Easter Eggs," debugging code, and other "hidden functionality" must not be included in production code.
5. Hidden fields in forms must not be used to store data that must be protected from disclosure. If hidden fields are used to store process control data across a transaction, they must be protected from tampering by the user (for example, stored hashes of the fields).
6. Error messages reported back to the user must not release any information about the operating environment, login process, paths, process control information, or other sensitive data.
7. The Post method must always be used when transferring sensitive user input data. This especially includes: a) login account information (user ID, Personal Identification Number (PIN), password), b) personal data for students and employees (name, address, phone, IDs, and other data belonging to a person). The Get method may be used when absolutely necessary, provided the data is encrypted. The Get method may leave sensitive information in the browser's history file, the web server logs, and proxy server logs, as well as the Hypertext Transfer Protocol (HTTP) Referrer field at the next site.
8. Hypertext Transfer Protocol Secure (HTTPS) and HTTP frames must not be used in the same page.
9. Operations performed by the Web application must not require escalation of privileges beyond those minimal privileges already granted to the Web server.
10. Temporary files created by the Web application during processing must be maintained in a protected sub-directory, and access to those files will be restricted to the Web server process. Uploads performed by the Web application will be written to a directory designated solely for that purpose with only read/write access (not executable or script access).

T. Filtering User Input Data

1. Filtering all user input data is paramount in preventing an attacker from hijacking the application code through techniques known as "SQL Injection", "Perl Injection", "Cross Site Scripting", format string attacks, and buffer overflows, as well as others.
 - a. Input validation must be performed on the server on all data received from the browser (data fields, visible or hidden, cookies, and HTTP headers) prior to using it in an application. Client-side code can never be trusted to assure the data is safe and properly validated.
 - b. Input validations must include length, data type, and special character tests.
 - c. Business rule edits must also be applied to all appropriate fields retrieved from the form and the HTTP header.
 - d. The developer must assume that all data fields could contain modified, invalid, and dangerous data.

U. Sign-on and Sign-off

1. If a sign-on function is required for a site, it must be presented at the earliest reasonable point; authenticated access is preferred to anonymous access.
2. Sensitive data must not be cached through use of the "Expire" and "Cache-Control" HTTP tags and the "no-cache" HTML pragma.
3. When user authentication is required for highly sensitive applications, client certificates must be used. For lower level (normal) user authentication, form-based or basic authentication over SSL (or client certificate) authentication must be used.
 - a. Note: RCW 19.34 controls the issuance of digital signature certificates for Washington state agencies.

- b. If multiple authentication methods are used (certificates and form-based), then the stronger authentication will take precedence over the weaker for granting rights or access.
4. Consideration must be given to how many times a user may concurrently login to an application.
5. To prevent brute-force password attacks, a method to lock out invalid login attempts will be included in a developed application. To prevent the corresponding denial of service attack, lockouts must be reset after a reasonable period of time has lapsed.
6. User privacy and acceptable use statements must be displayed or available to the user before entering the application. For a "cluster" of small applications, this may be displayed on a common menu page.
7. To help prevent denial of service attacks, resources must not be allocated for a user prior to their successful login.
8. A logout process must also be implemented to destroy all stored user credentials and release any resources. The "Back" button must not allow the user to regain entry to the site and must not release login information.
9. Session inactivity must trigger a logout. The length will vary depending upon the transaction, but must always be fairly short for sensitive transactions.
 - a. It is preferable to "dictate" a specific value for this timeout, but it must be dependant on multiple application factors, including how long it takes a typical user to complete the transaction. For sensitive transactions, this must be set as short as practically possible.
 - b. When a timed response is required, the user must be alerted via a prompt and given sufficient opportunity to indicate whether additional time is required.

V. Session Tracking

1. Session tracking, if implemented, must be done with cookies as opposed to URL rewriting.
2. Session IDs must be generated in a cryptographically sound fashion to assure a reasonable level of randomness, and must not be related in any way to user information.
3. The session ID must be at least 16 bytes long.
4. To prevent session hijacking, the user's domain (second level domain) must be tracked. If the domain changes during a session, the session must be closed, a security event written to the logs, and the transaction ended.

W. Cookies

1. The Secure flag must be set to assure that the cookie is only transmitted over encrypted links.
2. The Expires field must not be set to prevent caching the cookie on disk.
3. The Domain field must be set to the Web site's host name.
4. Since a user may manipulate the cookie, both input data validation and tamper detection techniques must be incorporated.
5. If the cookie must persist on the user's disk, it must be encrypted using a strong, industry standard encryption algorithm.

X. Data and Database Management

1. All developers must guard the storage and transmission of sensitive data. Developers must be familiar with all provisions of the Bellevue College IT security standard addressing "Database Management", which defines specific procedural and configuration elements for data systems.
2. All developed applications manipulating data must comply with that standard, particularly the standards established for storage and destruction of data. In addition, developed applications must not violate the standards set for creating and managing a Microsoft SQL server.
3. Additional requirements for application development identified include:

- a. Access to existing SQL databases or space to create a new SQL database will be granted following the SQL procedures described in the Bellevue College IT security standard addressing Network Data Storage.
- b. Microsoft SQL server access within an application must only be through Windows integrated authentication.
- c. Messages from SQL Server must be sanitized prior to passing them back to the end-user; such things as paths, IP addresses, column names, table names, SQL commands, and login information must be removed.
- d. Bellevue College created stored procedures must not be entered into the SQL Server master or msdb databases.
- e. Data manipulation by the application programs must be performed through stored procedure wrappers instead of direct calls to the SQL verbs select, insert, update, and delete. User queries must be performed via view instead of the base tables. This centralizes business logic within the database and allows for better control of the base tables.
- f. All data passed from the application layer must be validated prior to passing it to the database layer. These validations must include: appropriate character classes (alphabetic, numeric, punctuation, white space, control) and minimum/maximum length edits. Other business rule edits must be done at the application layer or within the database. Unsupported or undocumented (e.g., hidden or internal) APIs must not be used to manage and store data within the SQL Server database. Developers must use only standard SQL APIs and not vendor proprietary extensions.
- g. Consideration must be given to additional security for confidential data, such as credit card numbers and a Personal Identification Number (PIN); this might include encryption of specific columns.
- h. Non-DBA users must not be granted access to the master tables, or views of those tables.
- i. If an application has security vulnerabilities, they must be addressed in the application using vendor-supplied algorithms and protocols. Using homegrown security or encryption algorithms can expose data due to design and/or coding errors.

Y. Copyright

1. Washington state statutes require that all materials, processes, and inventions produced by a Bellevue College employee or contractor within the scope of their employment or contract and intended for use by Bellevue College and/or Bellevue College clients, are the property of the college and shall be copyrighted or patented in the college's name. All developed applications must include a textual copyright notice assigning ownership of the application and all copyrights to Bellevue College and identifying the year development of the release version of the application was completed.

Z. Sensitive Information

1. All applications developed at Bellevue College, whether IR-funded or self-funded, must adhere strictly to all policies, standards and laws regarding protection of private and sensitive state and student information. No application may be developed whose use violates:
 - a. The goals of the "Acceptable Use of the Bellevue College Network and Bellevue College Data Management Systems Policy," which are to:
 - i. Help assure the integrity and reliability of Bellevue College internal networks, hosts on those networks, Bellevue College databases, legacy systems, web-accessible resources, and any computing resource connected to them.
 - ii. Ensure the security and privacy of Bellevue College computer systems, networks and data.
 - iii. Ensure the protection and retention of sensitive college data.
 - iv. Establish appropriate guidelines for the use of the Bellevue College network and Bellevue College-owned data, whether accessed from on or off-campus.

- b. The purpose of the “DIS IT Security Policy,” which is to create an environment within State of Washington agencies that maintains system security, data integrity and privacy by preventing unauthorized access to data and by preventing misuse of, damage to, or loss of data.
 - c. The intent of the “Bellevue College Acceptable Use of State Resources” policy, which states that Education records of students attending the college are confidential and can only be released in accordance with the Family Education Rights and Privacy Act of 1974 (FERPA) and the administrative rules of the college.
2. All developers should be familiar with and adhere to, all of the above policies, as well as Bellevue College Policies numbered:
- a. 1500, Access to Public Records
 - b. 2550, Federal Privacy Act: Disclosure of Social Security Numbers, and
 - c. 2600, Family Education Rights and Privacy Act: Disclosure of Student Information

AA. Application Design Packet Submittal

1. In some circumstances, Bellevue College may be required to submit information regarding internally developed Internet-based applications to the DIS Senior Technology Management Consultant for review. Generally, because both Bellevue College and our de facto ISP, the CIS, are outside the state’s Digital Government Framework, this submission of applications for review is not required. However, if such a review is requested, the Dean of Information Resources will approve any such submission, and it will include, at a minimum, the following IT security-related information:
- a. A general description of the application, including its purpose and the nature of the information processed.
 - b. A description of the nature of the services to be provided to the user of the application (such as static data, interactive queries, data entry, electronic payments, etc.).
 - c. Clarification whether the application requires a high, medium or low level of confidence for user authentication and a summary of the analysis completed to determine this level.
 - d. A description of any known application integration issues if the proposed authentication mechanism involves the use of digital certificates
 - e. If the application involves providing access to an existing application, a description of the nature of the application’s access controls (user ID, password, etc.).
 - f. If it is intended that a user have to re-authenticate at the application level after having previously been authenticated by a centralized mechanism and process, the justification for not accepting the initial authentication will be described.
 - g. Any specific requirements for encryption during data transmission and/or storage will be articulated.
 - h. If known, the proposed development tools to be used in the creation or modification of the application for use on the Internet will be described.
 - i. Information regarding the hardware, operating system, and services provided by the Web server platform supporting the application must be included, if known.

Sanctions

Violations of the provisions of this, or any Bellevue College IT security standard or policy, will be dealt with immediately in the same manner as any violations of Bellevue College policies, and may result in disciplinary review. In such a review, the full range of disciplinary sanctions is available, including:

- 1. Permanent loss of computer use privileges;
- 2. Denial of future access to Bellevue College IT resources;
- 3. Disciplinary action – any disciplinary action will be taken in accordance with appropriate procedures as established by the Vice President of Human Resources (for employees) or the Dean of Student Services (for students);
- 4. Dismissal from the college; and/or
- 5. Legal action.

Those users who misuse or abuse any computing or network resource may have their login accounts closed and access to the systems immediately terminated. Some violations of this standard may also constitute a state, local, or federal criminal offense.

Appendix A – References

1. Howard, Michael and LeBlanc, David. *Writing Secure Code*, 2nd Edition. MS Press 2003.
2. McCluskey, *Development Services Security Best Practices*, CIS, 2002.
3. Tracy, Jansen, McLamon, Guidelines on Securing Public Web Servers, NIST, Feb. 2002.
4. *CIS Web Applications Standard*, CIS, 2002.
5. Fosenen, *Securing Internet Information Server 5.0*, SANS, May 2001.
6. Rhoades, *Auditing Web-Based Applications*, SANS, 2002.
7. Peteanu, *Best Practices for Secure Web Development*, http://www.blazonry.com/devnotes/secure_webdev-3.0.pdf, Revision 3.0, September 2000
8. DIS, *Public Records Privacy Protection Policy*, <http://www.isb.wa.gov/policies/portfolio/804P.doc>, 2006
9. DIS, IT Security Policy, <http://isb.wa.gov/policies/portfolio/400P.doc>, 2006
10. CIS, *Model College Privacy Notice*, <http://www.cis.ctc.edu/wctc/policies/CollegePrivacyNotice.htm>, 2002
11. Washington State Digital Signature Law – Chapter 19.34 RCW
12. Bellevue College Policy #2550: Federal Privacy Act: Disclosure of Social Security Numbers
13. Bellevue College Policy #5250: Information Security (IT) Security
14. Bellevue College Policy #5150: Acceptable Use of Bellevue College Networks and Bellevue College Data Management Systems
15. Bellevue College Policy #5000: Acceptable Use of Bellevue College Computers
16. Bellevue College Policy #4400: Acceptable Use of State Resources
17. Bellevue College Policy #5100: Software Licensing Compliance
18. Bellevue College Policy #6900: Records Storage and Disposal
19. Bellevue College Policy #2600: Family Education Rights and Privacy Act (FERPA): Disclosure of Student Information
20. Bellevue College Policy #3600: Copyright and the Right of Fair Use
21. Bellevue College Policy #1500: Access to Public Records
22. Bellevue College IT Security Standard: Data and Information Security
23. Bellevue College IT Security Standard: Database Management
24. Bellevue College IT Security Standard: Password Management
25. Bellevue College IT Security Standard: E-Commerce
26. Bellevue College IT Security Standard: Encryption
27. Bellevue College IT Security Standard: External Data Transfer
28. Bellevue College IT Security Standard: Incident Response
29. Bellevue College IT Security Standard: Internet Software Security
30. Bellevue College IT Security Standard: Network Data Storage
31. Bellevue College IT Security Standard: Web Information Accessibility
32. Bellevue College IT Security Standard: Web Servers
33. Bellevue College IT Security Standard: Web Space Usage
34. CIS System Development Lifecycle
<http://www.cis.ctc.edu/pub/groups/pres/longtermPDF/Appendix%20E.2.pdf>, 2006
35. Federal Access Board Web-based Intranet and Internet Information and Applications Standard --
<http://www.access-board.gov/sec508/guide/1194.22.htm>, 2006

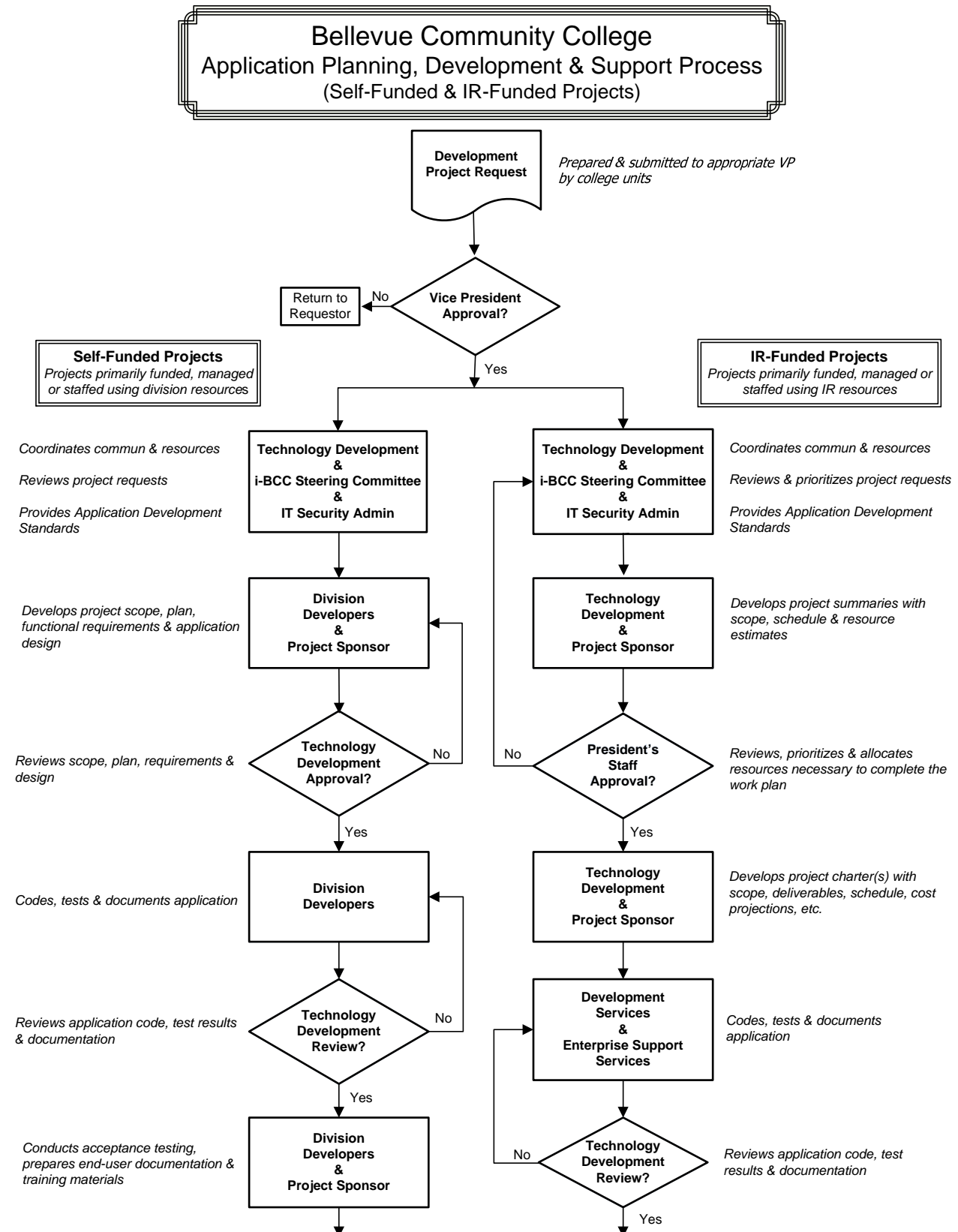
Effective Date: June 2005
Date Last Modified: July 10, 2009

Appendix B – Appendices Information

To clarify the appendices to this standard, they are listed below with a brief description:

Appendix A	IT Security Standard References Included with all IT Security Standards
Appendix B	Appendices Information This chart clarifying the appendices.
Appendix C	Development Project Request Form Form that is required to be completed prior to undertaking any development project. Its purpose is to document vital information regarding the project.
Appendix D	Application Planning, Development and Support Process Illustrated charts identifying the process and flow of the application planning, development and support processes for both self-funded and IR-funded development projects.
Appendix E	Application Documentation Template Template used by all Bellevue College developers to standardize overall documentation of a development project. The completed document can be used as a reference during development, deployment and maintenance phases of the application lifecycle.
Appendix F	Source Code Documentation Software development standards for building reference documentation for source code within an application.
Appendix G	.NET Naming Guidelines Software development standards for naming and style conventions within developed applications.
Appendix H	IT Security Requirements List of required and optional readings for developers from the Bellevue College IT Security program.
Appendix I	Intellectual Property Agreement Form This is the form that all college employees and contractors developing applications must sign and return to Information Resources, for filing with the IT Security Administrator.

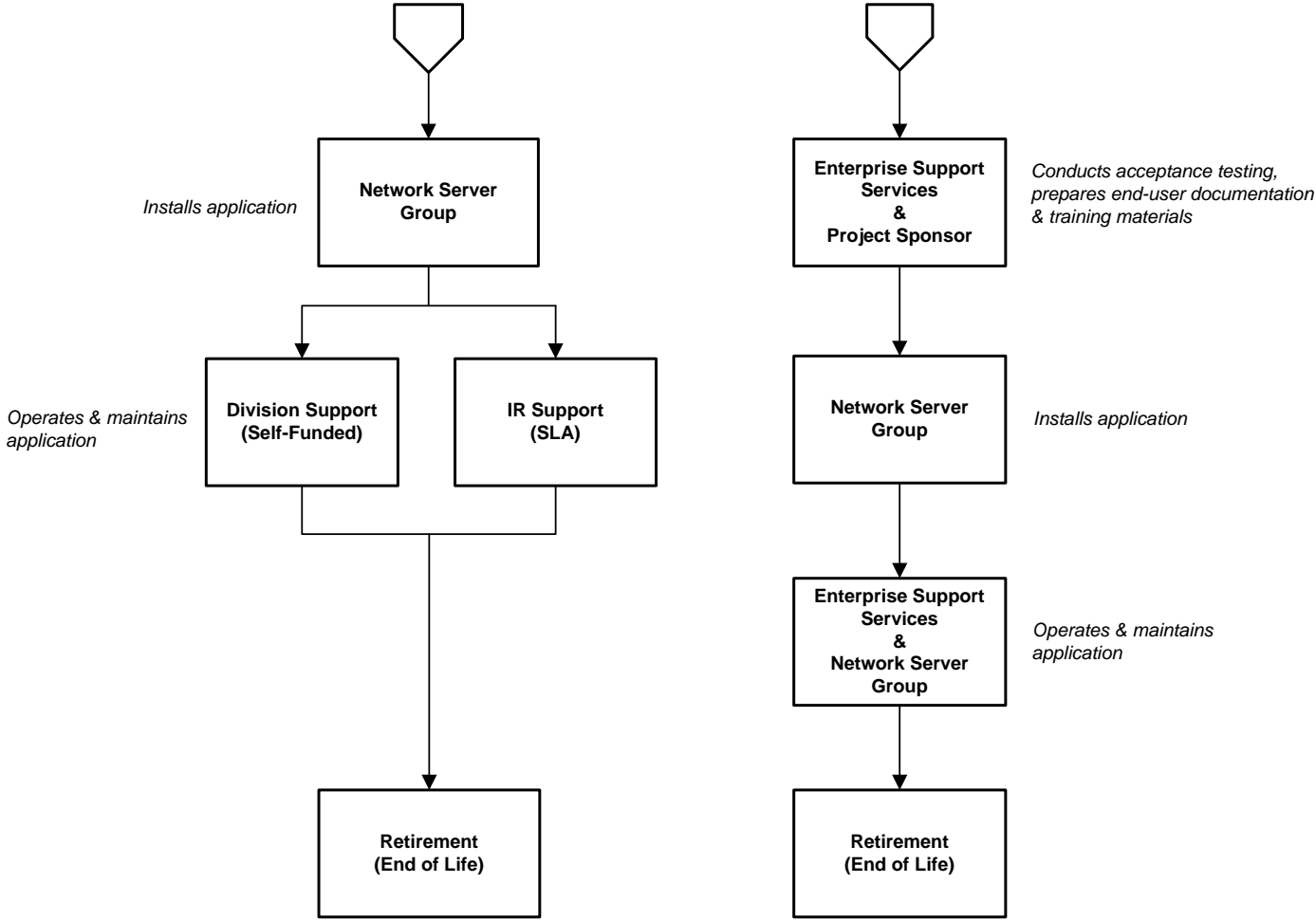
Appendix D – Planning Development and Support Process



Bellevue Community College Application Planning, Development & Support Process (Self-Funded & IR-Funded Projects)

Self-Funded Projects
Projects primarily funded, managed or staffed using division resources

IR-Funded Projects
Projects primarily funded, managed or staffed using IR resources



Projects that interface with college applications located at CIS must be reviewed by i-BCC and approved by President's Staff

Federal, state or local mandates may supersede this process

Appendix E –Application Documentation Template



[Project Title] **Detailed Web Application Design Structure**

Version [Project Version]
Last Updated: Month ##, ####

All sections of this template must be addressed by the application developer(s).

1. Summary:

Summary of entire project/site – its purpose, primary users, etc. (no more than a few sentences are necessary).

2. General Information:

Live URL:	
Authentication (windows/forms/mixed):	
Vault Path:	
Primary Customer Contact:	
Primary Dev. Contact:	
Language(s) Used:	
Special DB Access:	<i>(list any non-standard users with rights to this DB)</i>
Special Web Server Access:	<i>(list any non-standard users with rights to this app's folder)</i>

3. Assumptions and Dependencies:

Does this project depend on the Employees table, the HPSA, secure certificate, end user characteristics, preferred browser/OS specifications, etc?

4. Database/Data Information:

Live Location (Server/DB Name):	
Live Site Auth (SQL vs. Integrated):	
Live Site Web Server Location:	
Dev Location (Server/DB Name):	
Dev Site Auth (SQL Account Info):	
Dev Site Web Server Location:	

Data Received From Other Sources:

Source:	<i>Server.DB.Table</i>
Destination:	<i>Server.DB.Table</i>
Frequency:	<i>Daily, hourly, etc.</i>
Comments:	<i>Ex: Joins on ADAccounts to make a more useful Employees table.</i>

Data Pushed To Other Sources:

Source:	<i>Server.DB.Table</i>
Destination:	<i>Server.DB.Table</i>
Frequency:	<i>Daily, hourly, etc.</i>
Comments:	<i>This only happens if ...</i>

SQL Jobs:

Define any regularly scheduled data transformations or nightly emails that are generated from this site's data.

External Data:

Define any data outside of a database that is either pushed or pulled by this application. Ex.: Request Center files are stored on a share on Rainier; CPS pushes Learning Outcomes to a share on the Intranet.

5. IIS Settings:

Define the necessary settings within IIS such as Basic Authentication/Integrated/Anonymous or a combination thereof, folder listing permissions, security certificates, etc.

6. File Descriptions:

List all controls with a brief description (a sentence should be enough) of each as well as every instance the control is used within the site:

Control location/name.ascx	Description & Instances
----------------------------	-------------------------

List all classes with a brief description of each:

Class location/name	Description
---------------------	-------------

7. Registry Settings:

Define all paths, keys and expected values for registry settings for this application.

8. Third Party Integration Notes:

Describe any add-ons to this project such as a rich text control or Microsoft Application Block.

9. Upload Cautions / Directions:

State if there are any necessary warnings associated with uploading updates. Ex.: Do not upload folder XYZ because the latest addition is always on the web server.

10. Known Issues / Next Version Requests:

Report any known issues such as any browser/platform combinations which don't work. What is planned for the next version?

11. Related Documents:

List the names and locations of any documents relating to this project. Note: Data Flow Diagrams and Database Relationship diagrams should be added to the end of this document.

12. General Comments:

Please state anything that has not been covered thus far (is there anything unique about this project, any alterations to standard files, etc.?)

Appendix F – Source Code Documentation

Reference Documentation

Bellevue College primarily develops software using the .NET Framework and Microsoft Visual Studio.NET. These tools provide built-in mechanisms for building reference documentation from special comments added directly to the source code. In the industry this is known as *XML documentation* or */doc* (pronounced “slash-doc”).

Note: C# comes with built-in syntax for generating XML documentation. Microsoft also provides a plug-in for VB.NET – called VBCommenter – which provides the same functionality.

At a minimum, all classes and public methods must include a <summary> description:

```
///
```

Keep the <summary> brief and to the point. This text is also used by Visual Studio.NET’s Intellisense to provide pop-up help for class methods.

It is recommended that the <remarks>, <param>, <returns>, <see>, <seealso> and <note> tags be used where appropriate to supply comprehensive reference documentation regarding the code in question.

See MSDN or the NDoc manual for a complete list of XML documentation options:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vcorixmldocumentation.asp>
<http://ndoc.sourceforge.net/usersguide.html>

Once documentation has been added to the source code file(s), a utility can be used to extract that documentation into a standard format for easy reference. Bellevue College uses the open source application NDoc to produce an MSDN-style compiled help (.chm) file.

XML documentation is required for all libraries and classes that are expected to be [re]used by other code. It is strongly recommended for all other code.

Code Comments

Code must be written with proper naming conventions and in a style that makes it clear what is happening and why. On occasion, however, it is important to provide additional comments regarding pitfalls to watch out for, additional explanations, clarification, etc.

When deciding whether or not to add comments to a particular portion of code, consider the following:

1. If you hadn’t seen the code for six months would you be able to understand what it was doing right away?
2. If a stranger had to modify your code, would they understand it quickly enough to be productive in their alterations, or would they need to spend time figuring out what the code is doing?
3. Is there another, more common, solution that you have already determined doesn’t work due to special circumstances of the environment, etc.?

Commenting dos and don’ts:

1. Keep comments short, clear and concise.
2. Address comments to the code in the immediate vicinity. Information regarding the entire method or class should go in XML Documentation.
3. Don’t add comments that provide unnecessary duplication of information.
e.g.:

```
private string student; //the student
```

Appendix G – .NET Naming Guidelines

One of the most important elements of a managed class library is the use of a consistent naming pattern. Many of the common user questions do not even need to come up once these conventions are understood and widely used.

There are three elements of naming guidelines:

1. **Case-** Use the correct capitalization style.
2. **Mechanics-**Use nouns for classes, verbs for methods, etc.
3. **Word Choice-**Use terms consistently across libraries.

The following section describes rules for case and mechanics, and some philosophy for word choice.

Capitalization Styles

The following section describes different ways of capitalizing identifiers. These terms will be referred to throughout the rest of this document.

Pascal Casing

This convention capitalizes the first character of each word. For example:

```
BackColor
```

Camel Casing

This convention capitalizes the first character of each word except the first word. For example:

```
backColor
```

Upper case

Only use upper case, with an underscore (`_`) separating words, for constants, and for identifiers which contain an abbreviation that is two characters long or less. (Identifiers of three or more characters should be Pascal Cased.) For example:

```
public const string LOG_FILENAME = "errors.log";  
private const int MAX_RETRIES = 3;
```

```
System.IO  
System.Web.UI  
System.CodeDom
```

Capitalization summary

The following table describes the capitalization rules for different types of identifiers.

Type	Case	Notes
Class	PascalCase	
Enum values	PascalCase	
Enum type	PascalCase	
Events	PascalCase	
Exception class	PascalCase	Ends with Exception
Final Static field	PascalCase	
Interface	PascalCase	Begins with I
Method	PascalCase	
Namespace	PascalCase	
Property	PascalCase	
Public Instance Field	PascalCase	Never use, prefer properties
Protected Instances Fields	PascalCase	Never use, prefer properties
Private Instance Fields	PascalCase	Prefix with underscore. ex: private string <code>_Name</code> ;
Parameter	camelCase	
Local Variable	camelCase	

Word Choice

1. Avoid using class names duplicated in heavily used namespaces. Do not use the following for a class name:

System Collections Forms UI

2. Avoid using identifiers that conflict with common keywords. Do not use the following:

AddHandler *AddressOf* *Alias* *And* *Ansi*
As *Assembly* *Auto* *BitAnd* *BitNot*
BitOr *BitXor* *Boolean* *ByRef* *Byte*
ByVal *Call* *Case* *Catch* *CBool*

<i>CByte</i>	<i>Cchar</i>	<i>CDate</i>	<i>CDec</i>	<i>Cdbl</i>
<i>Char</i>	<i>Cint</i>	<i>Class</i>	<i>CLng</i>	<i>CObj</i>
<i>Const</i>	<i>Cshort</i>	<i>CSng</i>	<i>CStr</i>	<i>CType</i>
<i>Date</i>	<i>Decimal</i>	<i>Declare</i>	<i>Default</i>	<i>Delegate</i>
<i>Dim</i>	<i>Do</i>	<i>Double</i>	<i>Each</i>	<i>Else</i>
<i>Elseif</i>	<i>End</i>	<i>Enum</i>	<i>Erase</i>	<i>Error</i>
<i>Event</i>	<i>Exit</i>	<i>ExternalSource</i>	<i>False</i>	<i>Finally</i>
<i>For</i>	<i>Friend</i>	<i>Function</i>	<i>Get</i>	<i>GetType</i>
<i>Goto</i>	<i>Handles</i>	<i>If</i>	<i>Implements</i>	<i>Imports</i>
<i>In</i>	<i>Inherits</i>	<i>Integer</i>	<i>Interface</i>	<i>Is</i>
<i>Let</i>	<i>Lib</i>	<i>Like</i>	<i>Long</i>	<i>Loop</i>
<i>Me</i>	<i>Mod</i>	<i>Module</i>	<i>MustInherit</i>	<i>MustOverride</i>
<i>MyBase</i>	<i>MyClass</i>	<i>Namespace</i>	<i>New</i>	<i>Next</i>
<i>Not</i>	<i>Nothing</i>	<i>NotInheritable</i>	<i>NotOverridable</i>	<i>Object</i>
<i>On</i>	<i>Option</i>	<i>Optional</i>	<i>Or</i>	<i>Overloads</i>
<i>Overridable</i>	<i>Overrides</i>	<i>ParamArray</i>	<i>Preserve</i>	<i>Private</i>
<i>Property</i>	<i>Protected</i>	<i>Public</i>	<i>RaiseEvent</i>	<i>ReadOnly</i>
<i>ReDim</i>	<i>Region</i>	<i>REM</i>	<i>RemoveHandler</i>	<i>Resume</i>
<i>Return</i>	<i>Select</i>	<i>Set</i>	<i>Shadows</i>	<i>Shared</i>
<i>Short</i>	<i>Single</i>	<i>Static</i>	<i>Step</i>	<i>Stop</i>
<i>String</i>	<i>Structure</i>	<i>Sub</i>	<i>SyncLock</i>	<i>Then</i>
<i>Throw</i>	<i>To</i>	<i>True</i>	<i>Try</i>	<i>TypeOf</i>
<i>Unicode</i>	<i>Until</i>	<i>Variant</i>	<i>When</i>	<i>While</i>
<i>With</i>	<i>WithEvents</i>	<i>WriteOnly</i>	<i>Xor</i>	<i>eval</i>
<i>extends</i>	<i>Instanceof</i>	<i>package</i>	<i>var</i>	

3. Do not use abbreviations in identifiers (including parameter names).
4. If you must use abbreviations, use camelCasing for any abbreviation over 2 characters, even if this is not the standard abbreviation.

Namespaces

1. The following is the general rule for naming namespaces:

```
CompanyName.TechnologyName
```

Thus, we expect to see namespaces like the following:

```
Microsoft.Office
```

```
PowerSoft.PowerBuilder
```

Note: *This is merely a guideline. Third party companies can choose other names.*

2. Avoid the possibility of two published namespaces having the same name by prefixing namespace names with a company name or other well-established brand. (For example, Microsoft.Office for the Office Automation Classes provided by Microsoft.)
3. Use PascalCasing, and separate logical components with periods (For example, Microsoft.Office.PowerPoint). If your brand employs non-traditional casing, follow the casing defined by your brand, even if it deviates from normal namespace casing (for example, NeXT.WebObjects, and ee.cummings.)
4. Use plural namespace names where appropriate. For example, use System.Collections not System.Collection. Exceptions to this rule are brand names and abbreviations. For example, use System.IO not System.IOs.
5. Do not specify the same name for namespaces and classes. For example, do not use *Debug* for a namespace name and have a class named *Debug*.

Class and Class members

Class Naming Guidelines

1. Name classes with nouns or noun phrases.
2. Use PascalCasing.
3. Use abbreviations in class names sparingly.
4. Do not use any class prefix (such as **C**).
5. Do not use the underscore character.

The following are examples of correctly named classes.

```
public class FileStream {  
    }  
  
public class Button {  
    }  
  
public class String {  
    }
```

Interface Naming Guidelines

1. Name interfaces with nouns or noun phrases, or adjectives describing behavior. For example, **IComponent** (descriptive noun), **ICustomAttributeProvider** (noun phrase), and **IPersistable** (adjective).
2. Use PascalCasing.
3. Use abbreviations in interface names sparingly.
4. Do not use the underscore character.
5. Prefix interface names with the letter **I**, to indicate that the type is an interface.
6. Do not prefix class names with the letter **C**. Occasionally, it is necessary to have a class name that begins with **I** that is not an interface. This is acceptable as long as the character that follows **I** is lower case (For example, **IdentityStore**.)
7. Use similar names when defining a class/interface pair where the class is a standard implementation of the interface. The names should differ only by the letter **I** prefix on the interface name.

The following example illustrates these guidelines for the interface **IComponent** and its standard implementation, the class **Component**:

```
public interface IComponent {  
}  
  
public class Component : IComponent {  
}  
  
public interface IServiceProvider{  
}  
  
public interface IFormatable {  
}
```

Attribute Naming Guidelines

1. Add the **Attribute** suffix to custom attribute classes as in the following example:

```
public class ObsoleteAttribute{ }
```

Enum Naming Guidelines

1. Use PascalCasing for enums.
2. Use PascalCasing for enum value names.
3. Use abbreviations in enum names sparingly
4. Do not use a prefix on enum names (For example, adXXX for ADO enums, rtfXXX for rich text enums, etc.).

5. Do not use an **Enum** suffix on enum types.
6. Use a singular name for enums.
7. Use a plural name for bit fields.
8. Define enumerated values using an enum if they are used in a parameter or property. This allows the tool to know the possible values for a property or parameter.

```
public enum FileMode{
    Create,
    CreateNew,
    Open,
    OpenOrCreate,
    Truncate
}
```

9. Use the Flags custom attribute if the numeric values are meant to be bitwise or'ed together

```
[Flags]
public enum Bindings {
    CreateInstance,
    DefaultBinding,
    ExcatBinding,
    GetField,
    GetProperty,
    IgnoreCase,
    InvokeMethod,
    NonPublic,
    OABinding,
    SetField
    SetProperty,
    Static
}
```

An exception to this rule is when encapsulating a Win32 API. It is common to have internal definitions that come from a Win32 header. You can leave these with the Win32 casing, which is usually all –capital letters.

10. Use Int32 as the underlying type of an enum.

An exception to this rule is if the enum represents flags and there are many flags (>32) or the enum may grow to many flags in the future or the type needs to be different than type int for backwards compatibility.

11. Use enums only if the value can be completely expressed as a set of bitflags. Do not use enums for open sets (such as Operating system version etc).

ReadOnly and Const Field Names

1. Name static fields with nouns, noun phrases, or abbreviations for nouns.
2. Name static fields with ALL_CAPITALS.
3. Do not prefix static field names Hungarian type notation.

Variable Names

1. Use descriptive variable names. Variable names should be descriptive enough that in most scenarios the name of the variable and its type can be used to determine its meaning.
2. Name variable with camelCasing.
3. Use names based on a variable's meaning rather than names based on the variable's type. We expect development tools to provide the information about type in a handy manner, so the variable name can be put to better use describing semantics rather than type. Occasional use of type-based variable names is entirely appropriate.
4. Identify variables that have a class or global scope. The recommended way to identify a class or global scope variable is by beginning the variable name with one of the following:
 - `_` (e.g. `_retryCount`)
 - `m_` (e.g. `m_retryCount`)
 - `g_` (e.g. `g_retryCount`)
 - `m` (e.g. `mRetryCount`)
 - `g` (e.g. `gRetryCount`)
5. Do not use reserved words for variable names.
6. Do not prefix variable names with Hungarian type notation.

Parameter Names

1. Use descriptive parameter names. Parameter names should be descriptive enough that in most scenarios the name of the parameter and its type can be used to determine its meaning.
2. Name parameters with camelCasing.
3. Use names based on a parameter's meaning rather than names based on the parameter's type. We expect development tools to provide the information about type in a handy manner, so the parameter name can be put to better use describing semantics rather than type. Occasional use of type-based parameter names is entirely appropriate.
4. Do not use **reserved** parameters. If more data is need in the next version, a new overload can be added.
5. Do not prefix field names with Hungarian type notation.

```
Type GetType (string typeName)  
string Format (string format, object [] args)
```

Method Naming Guidelines

1. Name methods with verbs or verb phrases.
2. Name methods with PascalCasing as in the following examples.

```
RemoveAll()  
GetCharArray()  
Invoke()
```

Property Naming Guidelines

1. Name properties using a noun or noun phrase.
2. Name properties with PascalCasing.

3. Consider having a property with the same as a type.
4. When declaring a property with the same name as a type, also make the type of the property be that type. Although it sounds odd, it is correct. The following example uses correct property naming guidelines:

```
public enum Color {...}
public class Control {
    public Color Color { get {...} set {...} }
}
```

The following is incorrect.

```
public enum Color {...}
public class Control {
    public int Color { get {...} set {...} }
}
```

In the incorrect example, it will not be possible to refer to the members of the Color enum because Color.Xxx will be interpreted as being a member access that first gets the value of the Color property (of type int) and then accesses a member of that value (which would have to be an instance member of System.Int32).

Event Naming Guidelines

1. Name Event handlers with the **EventHandler** suffix as in the following example.

```
public delegate void MouseEventHandler(object sender, MouseEventArgs e);
```

2. Use two parameters named **sender** and **e**.

The sender parameter represents the object that raised the event. The sender parameter is always of type object, even if it is possible to employ a more specific type.

The state associated with the event is encapsulated in an instance of an event class named e. Use an appropriate and specific event class for its type.

```
public delegate void MouseEventHandler(object sender, MouseEventArgs e);
```

3. Name event argument classes with the **EventArgs** suffix as in the following example:

```
public class MouseEventArgs : EventArgs {
    int x;
    int y;
    public MouseEventArgs(int x, int y)
        { this.x = x; this.y = y; }
    public int X { get { return x; } }
    public int Y { get { return y; } }
}
```

4. Name event names that have a concept of pre and post using the present and past tense (do not use the BeforeXxx\AfterXxx pattern). For example, a close event that can be canceled would have a Closing and Closed event.

```
public event EventHandler ControlAdded {
    //..
}
```

5. Consider naming events with a verb.

Case Sensitivity

1. Do not use names that require case sensitivity. Components must be fully usable from both case-sensitive and case-insensitive languages. Since case-insensitive languages cannot distinguish between two names within the same context that differ only by case, components must avoid this situation.

2. Do not have two namespaces whose names differ only by case.

```
namespace ee.cummings;  
namespace Ee.Cummings;
```

3. Do not have a function with two parameters whose names differ only by case.

```
void foo(string a, string A)
```

4. Do not have a namespace with two types whose names differ only by case.

```
System.WinForms.Point p;  
System.WinForms.POINT pp;
```

5. Do not have a type with two properties whose names differ only by case.

```
int Foo {get, set};  
int FOO {get, set}
```

6. Do not have a type with two methods whose names differ only by case.

```
void foo();  
void Foo();
```

Avoiding Type Name Confusion

Different languages use different terms to identify the fundamental managed types. Designers must avoid using language-specific terminology. Follow the rules described in this section to avoid type name confusion.

1. Use semantically interesting names rather than type names.
2. In the rare case that a parameter has no semantic meaning beyond its type, use a generic name. For example, a class that supports writing a variety of data types into a stream might have the following methods:

```
void Write(double value);  
void Write(float value);  
void Write(long value);  
void Write(int value);  
void Write(short value);
```

the above example is preferred to the following language-specific alternative:

```
void Write(double doubleValue);  
void Write(float floatValue);  
void Write(long longValue);  
void Write(int intValue);  
void Write(short shortValue);
```

3. In the extremely rare case that it is necessary to have a uniquely-named method for each fundamental data type, use the following **universal type** names:

C# type name	Visual Basic type name	JScript type name	Visual C++ type name	IL representation	Universal type name
Sbyte	SByte	SByte	char	I1	SByte
Byte	Byte	byte	unsigned char	U1	Byte
Short	Short	short	short	I2	Int16
Ushort	UInt16	UInt16	unsigned short	U2	UInt16
Int	Integer	int	int	I4	Int32
UInt	NA	NA	unsigned int	U4	UInt32
Long	Long	long	__int64	I8	Int64
Ulong	UInt64	UInt64	Unsigned __int64	U8	UInt64
Float	Single	float	float	R4	Single
Double	Double	double	double	R8	Double
Bool	Boolean	boolean	bool	I4	Boolean
Char	Char	char	wchar_t	U2	Char
String	String	String	String	System.String	String
Object	Object	Object	Object	System.Object	Object

A class that supports reading a variety of data types from a stream might have the following methods:

```
double ReadDouble();
float ReadSingle();
long ReadInt64();
int ReadInt32();
short ReadInt16();
```

The above example is preferred to the following language-specific alternative:

```
double ReadDouble();
float ReadFloat();
long ReadLong();
int ReadInt();
short ReadShort();
```

ASP.NET Web Control Naming Guidelines

In .NET it is preferred NOT to use prefixes (or Hungarian notation) in variable names, however, we make an exception when we are building ASP.NET pages. It becomes important to easily identify the type of UI control since different kinds of UI elements might have similar names. Note that we always use camel case for control names. The following table outlines the control and its corresponding naming prefix.

Control Type	Prefix	Example
ActionButton	abtn	abtnAddToCart
BlockHeader	bh	bhProducts
Button	btn	btnUpdateOrder
CheckBox	chk	chkIncludeService
CheckBoxList	chkList	chkListServices
CompareValidator	cv	cvPasswords
CustomValidator	custv	custvPricing
DataGrid	dg	dgRatePlans
DropDownList	ddl	ddlPhones
HyperLink	hl	hlFeatures
Label	lbl	lblInstructions
LargeBlockHeader	lbh	lbhPlans
LinkButton	lbtn	lbtnActivate
LinkList	ll	llMyTMobileOffers
List	list	listRingTones
ListBox	lb	lbAvailableItems
PageTitle	pt	ptGetMore
Panel	pnl	pnlPageSection
PlaceHolder	ph	phPlanContainer
PopupHyperLink	phl	phlAgreement
PrimaryLinkIndicator	pli	pliRatePlanDetails

Control Type	Prefix	Example
QuestionMarkLinkIndicator	qli	qliQuestions
RadioButton	rbtn	rbtnOptionalService
RadioButtonList	rbtnList	rbtnListOptionalServices
RangeValidator	rv	rvAge
RegularExpressionValidator	rxv	rxvCreditCardNumber
RequiredFieldValidator	rfv	rfvUserName
SecondaryLinkIndicator	sli	sliOptions
TextBox	txt	txtPassword
UserControl	Uc	ucNavigation
ValidationSummary	vs	vsErrors

Appendix H – IT Security requirements

Required

Application development must be in compliance with all applicable Policies and Standards. All developers are required to comply with the following Bellevue College IT Security Policies and Standards:

- a. Information Technology (IT) Security Policy
- b. Acceptable Use of Bellevue College Networks and Bellevue College Data Management Systems Policy
- c. Acceptable Use of Bellevue College Computers Policy
- d. Acceptable Use of State Resources Policy
- e. Software Licensing Compliance Policy
- f. Bellevue College Records Storage and Disposal Policy
- g. Bellevue College Family Education Rights and Privacy Act (FERPA): Disclosure of Student Information Policy
- h. Bellevue College Copyright and the Right of Fair Use Policy
- i. Bellevue College Access to Public Records Policy
- j. Bellevue College IT Security Standard: Application Development
- k. Bellevue College IT Security Standard: Password Management
- l. Bellevue College IT Security Standard: on Web Information Accessibility

Optional

In addition, the following Bellevue College Policies and IT Security Policies and Standards may also apply to specific development projects. It is the responsibility of developers to comply with the policies and standards which apply to their specific application development project:

- a. General Privacy Act: Disclosure of Social Security Numbers Policy
- b. Bellevue College IT Security Standard: Database Management
- c. Bellevue College IT Security Standard: Data and Information Security
- d. Bellevue College IT Security Standard: E-Commerce
- e. Bellevue College IT Security Standard: Encryption
- f. Bellevue College IT Security Standard: External Data Transfer
- g. Bellevue College IT Security Standard: Internet Software Security
- h. Bellevue College IT Security Standard: Network Data Storage
- i. Bellevue College IT Security Standard: Web Servers
- j. Bellevue College IT Security Standard: Web Space Usage

In accordance with the Bellevue College IT Security Standard addressing “Intrusion Detection and Incident Response”, IR reserves the right to remove or disable any developed applications deployed on the Bellevue College network which may compromise or disrupt campus computing resources.

Appendix I – Intellectual Property Agreement Form



3000 Landerholm Circle SE • Bellevue, WA 98007-6484 • www.bellevuecollege.edu

Information Resources, N215
Telephone: (425) 564-4200 Fax: (425) 564-6199

INTELLECTUAL PROPERTY AGREEMENT

In accordance with Washington state statutes, all materials, processes, and inventions produced by the employee or contractor solely for Bellevue College and/or Bellevue College clients are the property of Bellevue College and shall be copyrighted or patented, if at all, in Bellevue College's name. All copyrightable work produced by an employee or contractor within the scope of his or her employment or engagement shall be deemed a "work made for hire" as defined under the U. S. Copyright Act of 1976.

I have read and understand the preceding paragraph.

Print name: _____

Signature: _____

Date: _____