Systems Development Guidelines

## Table of Contents

1. **Introduction**
2. **Purpose and Scope**
3. **Technology Stack Overview**
4. **Architectural Principles**
5. **Task Management and Issue Tracking**
6. **Coding Guidelines and Best Practices**
7. **API Design and Management**
8. **Database Guidelines**
9. **Security Guidelines**
10. **DevOps and Continuous Integration/Continuous Deployment (CI/CD)**
11. **Testing Guidelines**
12. **Deployment and Environment Management**
13. **Monitoring and Logging**
14. **Version Control Practices**
15. **Documentation and Knowledge Sharing**
16. **Continuous Improvement**
17. **Exemptions and Alternative Tools**
18. **Future State and Strategic Initiatives**
19. **Conclusion**

## 1. Introduction

In the rapidly evolving technological landscape, maintaining consistency, quality, and efficiency across all software development projects is essential. This document outlines the system development guidelines that all teams working with or for the British Columbia Energy Regulator (BCER) are expected to follow. By adhering to these guidelines, the BCER ensure our software solutions are robust, maintainable, and scalable, aligning with our business objectives and delivering value to BCER stakeholders.

## 2. Purpose and Scope

The purpose of this document is to provide a comprehensive set of guidelines covering all aspects of the software development lifecycle (SDLC). These guidelines are designed to promote best practices in software development and apply to all projects within the organization, regardless of size or complexity.

## 3. Technology Stack Overview

The BCER standardized technology stack ensures interoperability, maintainability, and scalability across all projects. This section provides an overview of the key technologies and tools that form the foundation of our development efforts.

* **Backend Development**:
	+ **Data Access Strategy (DAL)**: Tibco Data Virtualization (TDV) should be used wherever possible to access operational data in all applications.
		- Where TDV is not appropriate, implement Microsoft Entity Framework Core to manage data interactions with the Oracle database.
	+ **Application Architecture:**  .NET 8 should be used for developing RESTful APIs, adhering to OpenAPI specifications to ensure standardization and interoperability.
* **Frontend Development**: Employing Angular for creating dynamic and responsive user interfaces.
* **Database Management**: Oracle Database is the standard relational database system.
	+ Alternate technologies such as PostgreSQL are acceptable where appropriate and approved by the BCER.
* **Database Migrations**: Flyway is used for version-controlled database migrations, ensuring consistency across environments.
* **Authentication and Authorization**: Keycloak is used for identity and access management across all applications.
* **API Management**: Red Hat 3Scale is used for managing APIs, providing centralized OAuth2 flows, and exposing APIs through safely through apicast.
* **Containerization and Orchestration**: OpenShift serves as the platform for deploying and managing containerized applications. By default, all new BCER applications should be containerized, using OpenShift or Docker Compose (OpenShift preferred).
* **Continuous Integration/Continuous Deployment**: Azure DevOps Server (on-premises) is used for build and release pipelines, always incorporating business owner and IS approvals when moving to staging and production.
* **Testing Frameworks**:
	+ **Unit Testing**: xUnit for .NET backend and Karma for Angular frontend.
	+ **Integration Testing**: Tavern is used for API integration testing.
	+ **End-to-End Testing**: Cypress is utilized for lightweight E2E testing of web applications.
* **Frontend Tools**: TypeScript configuration (tsconfig.json), ESLint, and Prettier are required for Angular projects.
* **Shared Components**: Angular applications are required to use the custom BCER Angular shared components (<https://cicdhost.bcogc.ca/OGC/Core/_packaging?_a=feed&feed=OGC-Core>).
* **Hosting**: Angular Single Page Applications (SPAs) are hosted within ASP.NET 8 runtime environments.

## 4. Architectural Principles

Adhering to sound architectural principles ensures that our software systems are scalable, maintainable, and aligned with business objectives. This section outlines the key architectural guidelines that underpin our development practices.

* **Microservices Architecture**: Adopt a microservices architecture where appropriate, using principles similar to Domain-Driven Design (DDD) to align services with business domains.
* **Database Schema per Microservice:** For business-focused microservices, having a dedicated database schema is generally recommended to ensure loose coupling and independent scalability. However, the approach can be determined on a project-by-project basis, considering factors such as microservice size, complexity, scalability, and requirements. For microservices focused on non-functional requirements, a dedicated schema may not be necessary if persistent state management is not involved.
* **Separation of Concerns**: Design applications to separate concerns across different layers (e.g., presentation, business logic, data access).
* **Reusability and Modularity**: Encourage the development of reusable components and services to promote efficiency and consistency.
* **Data Access**: Wherever possible, access to BCER data should leverage TDVdata services.
* **Data Locking:** Inter-system or database field locking requirements for applications must be defined if required.  Consider impacts to IN PROGRESS/IN REVIEW amendments or operational activities, for example.
* **Data Retention:** Data retention requirements must be gathered for the associated data generated or stored with any new development.
	+ <https://www2.gov.bc.ca/gov/content/governments/services-for-government/information-management-technology/records-management/information-schedules/arcs>
	+ <https://www2.gov.bc.ca/gov/content/governments/services-for-government/information-management-technology/records-management/information-schedules/special-schedules/transitory-records>
	+ <https://www2.gov.bc.ca/assets/gov/british-columbians-our-governments/services-policies-for-government/information-management-technology/records-management/orcs/oil_and_gas_regulation_orcs.pdf>
	+ https://www2.gov.bc.ca/assets/gov/british-columbians-our-governments/services-policies-for-government/information-management-technology/records-management/orcs/human-resources.pdf
	+ Where possible, removal of data based on these rules should be core to the development.

## 5. Task Management and Issue Tracking

Effective task management and issue tracking are crucial for project transparency, accountability, and collaboration. This section details the guidelines for managing tasks and issues within our projects.

**5.1 Usage of Jira**:

* All projects must utilize Jira for task management and issue tracking to ensure consistent workflows and communication.

**5.2 Ticket-Based Development**

* **Code Origin**: All code changes should originate from a Jira ticket or a Service Desk Plus request, ensuring traceability and context.
* **Issue Types**: Use appropriate Jira issue types (e.g., Story, Bug, Task) to categorize and prioritize work effectively.

**5.3 Workflow Management**

* **Standard Workflows**: Adhere to standardized Jira workflows to ensure consistency in issue progression.
* **Status Updates**: Regularly update the status of Jira tickets to reflect the current state of work.

**5.4 Collaboration and Communication**

* **Comments and Attachments**: Use Jira comments and attachments to document discussions and decisions.
* **Notifications**: Configure Jira notifications to keep stakeholders informed about progress and changes.

**5.4 Reporting and Metrics**

* **Dashboards**: Create Jira dashboards to monitor project progress, team performance, and key metrics.
* **Reports**: Use Jira's reporting features to track sprint progress and identify bottlenecks.

## 6. Coding Guidelines and Best Practices

Consistent coding guidelines and best practices enhance code readability, maintainability, and quality. This section outlines the guidelines for both backend and frontend development.

**6.1 General Guidelines**

* **Ongoing and continuous Updates**: Logical and Conceptual data models must be updated in a consistent, ongoing manner instead of a ‘one-time’ approach on delivery of the product.
* **Consistent Naming Conventions**: Use clear, descriptive names for variables, methods, classes, and other code entities.
* **Code Formatting**: Adhere to standardized formatting rules; utilize automated formatting tools where applicable.
* **Comments and Documentation**: Provide meaningful comments and inline documentation to explain complex logic and design decisions.
* **Version Control**: All code must be committed to our Git repositories on Azure DevOps Server, following the agreed-upon branching strategy.
* **Automated Review**: We strive to use tools to automate code reviews using tools such as Sonarqube, linters, StyleCop wherever possible, ensuring compliance and efficiency.
* **iBCER Management**: Any use of enumerated coded values in database tables in all BCER applications must allow for administration via iBCER
* **Error Handling:** Error messages with steps to correct the failure or issue are presented to the user when known business rule, data validation or system errors occur, so the user can correct the issue
	+ Error messages displayed to user must be in plain English (target grade 7 comprehension)
	+ Error messages captured in logs must provide clear indications of the cause and inform operational support.
* **System Time Out Definition**: Define the period of inactivity after which logged-in users will be automatically logged out of the system.
	+ Unless otherwise specified, default time out should be 240 minutes.

**6.2 Backend Development Guidelines**

* **Framework Usage**: Leverage the most current LTS .NET release features effectively, following best practices outlined by Microsoft.
* **Data Access Strategy (DAL)**:
	+ Tibco Data Virtualization (TDV) should be used where derived table data is being consumed.
	+ Microsoft Entity Framework Core must be used to manage data interactions with the Oracle database.
* **Application Architecture:** The most recent version of.NET LTS must be used for developing RESTful APIs, adhering to OpenAPI specifications to ensure standardization and interoperability.
* **Data Persistence:** Non-ephemeral data[[1]](#footnote-1) must be managed to ensure data preservation (backup and restoration).

*e.g. Non ephemeral data cannot be stored in databases within the OpenShift cluster.*

* **Input Data Validation**: Ensure the quality and validity of data inputs into the system are enforced:
	+ **Quality**
		- Define list of finite values where possible
		- Define range of acceptable values
		- Verify plausibility of user-entered value
	+ **Definition**
		- Define data type
		- Define if NULL is acceptable value
	+ **Integrity**
		- Ensure key data cannot be overwritten or edited unless required
	+ **Date & Time Format, Time Zone**
		- Ensure date and time format is consistent:  MMM/DD/YYYY HH:MM:SS
		- All systems must be implemented to leverage Coordinated Universal Time. Conversion of UTC to local time zone must be managed at the client level.
			* ***NOTE:*** *Previous to 2024, most systems were built using the Pacific Daylight Time (PDT) time zone. All systems going forward should use the Coordinated Universal Time (UTC) time zone on the backend, with time being converted to local machine time for end users. Existing systems must be updated to align to this standard when redevelopment or enhancement is undertaken, and it is deemed appropriate after consultation with the BCER.*
* **Confidential Data Requirements:** Systems managing or presenting 'confidential data' must define requirements that ensure that data is managed appropriately:
	+ For externally available systems, reports, etc. ensure confidential data (security) requirements limit access to those who should be able to view this data; or need to ensure no confidential data requirements are presented.
	+ Develop a register of confidential data fields used to ensure it is treated appropriately.
* **Code Style Enforcement:** Use StyleCop to enforce consistent coding styles and conventions.
* **Design Principles:** Apply [SOLID](https://en.wikipedia.org/wiki/SOLID) principles to create modular, scalable, and maintainable codebases.
* **Asynchronous Programming**: Implement async/await patterns to enhance performance and responsiveness.
* **Dependency Injection**: Use built-in dependency injection mechanisms to manage dependencies efficiently.
* **Exception Handling**: Implement a consistent approach to exception handling, including logging and error reporting.
* **Static Code Analysis**: Integrate SonarQube into the development process to enforce code quality.
* **Data Management**: All systems must be designed to facilitate data retention rules and records management requirements.

**6.3 Frontend Development Guidelines**

All applications must be developed for Chromium based browsers – users of browsers other than Chromium-based should be warned they are using an unsupported browser.

* **UI/UA Standards**: All applications developed for the BCER must follow UI/UX guidelines as outlined in the [User Experience Standards Guide v2.4](https://bcogcca.sharepoint.com/%3Aw%3A/s/msteams_739d8b_991501-CMIS-WorkingDocuments/ETS7vq8eRBxHsJ8poU_KewMBea1WxYIPQtKcPxTfB9xLqA?e=iSZGPm).
* **Mobile Friendly**: Applications must automatically scale to support mobile friendly browsers.
* **Framework Conventions**: Follow the Angular Style Guide for project structure and coding conventions.
* **Component-Based Architecture**: Develop reusable components to promote code reuse and simplify maintenance.
* **State Management**: Utilize RxJS for managing asynchronous data streams and state within Angular applications.
* **Configuration Files**: Include tsconfig.json for TypeScript configuration in all Angular projects.
* **Linting and Formatting**: Use ESLint for code linting and Prettier for code formatting.
* **Shared Components**: All Angular applications must use the custom BCER Angular shared components.
* **Responsive Design**: Ensure applications are responsive and adhere to accessibility standards.
* **Time zone:** All times must be presented to end users in local machine time.

## 7. API Design and Management

APIs are critical for enabling communication between services and with external consumers. This section provides guidelines for designing, documenting, and managing APIs.

* All services must conform to the BC Government API guidelines as published at <https://github.com/bcgov/api-guidelines>
* **RESTful API Design**: Design APIs following RESTful principles with logical resource-oriented endpoints and appropriate HTTP methods.
* **OpenAPI Specification**: Document APIs using OpenAPI (Swagger) to facilitate easy integration and understanding by consumers.

**7.1 API Management**

* **Mandatory API Manager Access**: All BCER API access must be controlled by an API Manager:
* **Current State**:
	+ Integrate backend APIs into Red Hat 3Scale API Manager for centralized management.
	+ **OAuth2 Flows**: Utilize Red Hat 3Scale API Manager to provide centralized OAuth2 authentication and authorization flows.
* **Versioning**: Implement API versioning strategies to maintain backward compatibility.
* **Error Handling**: Provide consistent and informative error responses in a standardized format.
* **Performance Optimization**: Implement caching strategies and optimize endpoints where necessary.

## 8. Database Guidelines

Proper database guidelines ensure data integrity, performance, and security. This section outlines the guidelines for database design, migrations, and optimization.

* **Collaboration with Data Architecture & Integration Team**: Work closely with the Data Architecture & Integration team to ensure compliance with organizational database guidelines.
* **Database Choice**: Oracle Database is the standard relational database system. Other databases (PostgresDB, SQL Server) can be used in specific circumstances given BCER IDS approval.
* **Schema per Microservice**: While having a dedicated database schema for each microservice is often recommended to ensure data encapsulation and independence, the approach can vary based on the project. Factors such as microservice size, complexity, and whether the microservice is business-focused or non-functional in nature should be considered when deciding on schema design. This decision is best made on a project-by-project basis.
* **Schema Design**: Schemas should generally follow normalization principles (up to the required normal forms) to ensure data integrity and reduce redundancy. However, exceptions can be made, such as denormalization, when justified by specific performance or scalability requirements. These decisions should be evaluated and documented on a case-by-case basis.
	+ Schemas are created such that it may be possible going forward to identify a business owner for a schema (in alignment with the business owner of applications)
	+ Schema separation encourages the use of data microservices going forward, will help reduce coupling between microservices, and should allow for improved data security
* **Naming Conventions**: Apply consistent naming conventions for database objects.
	+ Schemas are not named according to application names (except for COTS solutions)
	+ Schema naming should align to a business subject area (that may align with a specific application)
	+ Where a data model does not align to a specific subject area or the data model is for integration, a “common” schema such as PASR (permitted data) or OPS (operational data) can be leveraged
* Ensure that the data model generated as part of the solution adheres to the data standards as defined in the Commission's data standards document

**8.1 Database Migrations**

* **Flyway Usage**: Employ Flyway for managing database schema migrations. All changes must be scripted and version controlled.
* **Migration Scripts**: Write incremental, reversible migration scripts to ensure smooth transitions.
* **Version Control Integration**: Store migration scripts alongside application code in the version control system.

**8.2 Performance Optimization**

* **Indexing**: Regularly analyze and optimize indexes to improve query performance.
* **Query Optimization**: Utilize Oracle's performance tools for monitoring and optimization.

**8.3 Data Integrity and Security**

* **Data Integrity**: Enforce through primary keys, foreign keys, and constraints.
* **Access Controls**: Implement appropriate database access controls to ensure security.

**8.4 Data Quality and Guidelines**

* Data Quality definitions must be defined and documented for all critical data elements in a central repository.
* Data Governance processes must be followed.
* **Data Ownership:** New tables/schemas must have defined data owners before changes can be made to database environments.
	+ A subject area should be chosen at a level such that a single business owner can be identified

## 9. Security Guidelines

Security is paramount in all aspects of software development. This section details BCER guidelines for authentication, authorization, secure coding practices, and security testing.

All BCER systems must conform to the [OWASP](#_Appendix) standards to ensure the information system is secure.

**9.1 Authentication and Authorization**

* **Identity Management**:
	+ **Current State**: Utilize Keycloak for centralized authentication and authorization.
	+ **Future Consideration**: Exploring the potential transition to Microsoft Entra ID for enhanced integration with Microsoft services.
* **Single Sign-On (SSO)**: Implement SSO where applicable to enhance user experience and security.
* **Role-Based Access Control (RBAC)**: Define and manage user roles and permissions within the identity management system.
	+ New application security groups required, or existing groups that will be leverages
	+ New database security groups
	+ Access restrictions based on security group membership (functionality and data)
	+ Changes to authorizations and authentications are handled by the system in a timely manner (changes made to permissions are reflected in a reasonable amount of time in user access).
	+ All applications MUST have a READ ONLY role for use by IDS resources in supporting applications and security.  This role may be separate from a wider use READ ONLY role, but must exist for ease of permission management in KeyCloak
* **API Security**: Secure all APIs using OAuth2 flows provided by the API management platform.

**9.2 Secure Coding Practices**

* **Input Validation**: Sanitize and validate all user inputs to prevent injection attacks and other vulnerabilities.
* **Data Protection**: Encrypt sensitive data in transit using TLS (Transport Layer Security). The most current version of TLS configured with BCER’s security certificate must be used.
	+ TLS must be used for all traffic
* **Secrets Management**: Avoid hard-coding secrets; use secure secrets management systems integrated with OpenShift.
* **Logging and Monitoring**: Implement security logging to detect and respond to potential threats.
* **Prevent Supply Chain Attack**: Code and third-party component libraries must be free of any known vulnerabilities and patched to the current version of technologies

**9.3 Security Testing**

* **Static Analysis**: Use SonarQube and Qualys to detect security vulnerabilities during development.
* **Dynamic Analysis**: Perform regular security testing on running applications.
* **Security Updates**: Keep all dependencies and libraries up to date.

**9.4 General Security Requirements**

* **Reverse Proxy:** All public facing web apps and websites are protected by the BCER reverse proxy.
* **Mandatory Security Scan:** Moderate updates to public facing apps require web app vulnerability scanning.
* **Penetration Testing:** Major updates to public facing apps require penetration testing.

## 10. DevOps and Continuous Integration/Continuous Deployment (CI/CD)

Efficient DevOps practices enhance collaboration and streamline deployment processes. This section outlines BCER guidelines for CI/CD pipelines and deployment configurations. Unless specifically approved, all development initiatives will leverage existing BCER CI/CD infrastructure.

Approval processes must be in place for each environment – consultation with IDS will provide appropriate Business and IDS individuals for approval through each environment.

**10.1 Continuous Integration**

* **Automated Builds**: Configure Azure DevOps pipelines to compile code, run unit tests, and perform static code analysis upon pull requests and merges.
* **Code Quality Checks**: Integrate SonarQube analysis into the CI process.
* **Image Security Scanning**: Incorporate Advanced Cluster Security (ACS) and Quay image scanning into the CI/CD pipeline.
* **Artifact Management**: Store build artifacts and container images in secure, centralized repositories.

**10.2 Continuous Deployment**

* **Release Pipelines**: Utilize Azure DevOps Server for managing release pipelines, incorporating business owner and IS approvals when moving to staging and production.
* **Environment Promotion**: Automate deployments across Development, Test, UAT, Staging, and Production environments.
* **Release Control**: Implement approval steps to ensure compliance with organizational policies.
* **Rollback Strategies**: Implement mechanisms to revert to previous stable versions when necessary.

**10.3 Deployment Configurations**

* **Configuration Management**: Manage application deployment configurations using Dockerfiles and OpenShift deployment files.
* **Containerization**: Package applications into Docker containers for consistency.
* **Platform Standardization**: Deploy all applications on OpenShift to leverage its features.

## 11. Testing Guidelines

Comprehensive testing ensures software reliability and quality. This section details BCER guidelines for unit testing, integration testing, end-to-end testing, and continuous testing.

All applications must be developed, implemented and testing in all BC-ER environments (DEV/TEST/UAT/STAGE/PROD) unless a specific exemption is provided.

* Unit Testing (DEV)
* System Testing (TEST)
* User End Testing (UAT)
* Smoke Testing/Migration confirmation (STAGE)
* Smoke Testing (PROD) – as possible

**11.1 Unit Testing**

* **Backend**: Use xUnit for unit testing .NET applications.
* **Frontend**: Use Karma for unit testing Angular components.
* **Coverage Goals**: Aim for a minimum of 60% code coverage on unit tests.

**11.2 Integration Testing**

* **Framework**: Use Tavern and Postman for API integration testing.
* **Database Testing**: Use test databases or mock databases to verify data interactions.

**11.3 End-to-End Testing**

* **Tool**: Utilize Cypress for front end testing, lightweight E2E testing of web applications.
* **Test Strategy**: Focus on critical user flows to reduce fragility.

**11.4 Continuous Testing**

* **Automation**: Integrate all tests into the CI/CD pipelines.
* **Test Environments**: Use dedicated environments that closely mirror production configurations.

**11.5 Confidential Data Specific Testing:** Test cases specific to confidential data must be included to test this on every external release that includes well (or any confidential) information - disclosure of confidential data must be prevented at all costs).

Examples of confidential data are provided below (Px specific) - this is not an exhaustive list:

|  |  |  |  |
| --- | --- | --- | --- |
| GCI Top | Pool Density | ReEntry Date | TVD to Completion Point |
| GCI Base | Commingling Effective Date | Incremental Distance Drilled | MD to Top of Pay |
| Field | Disposal Injection Approval Number | TVD to Top of Pay | Top Cut Depth |
| Pool | Historical Volumetric Flag | Completion Date | MD to Completion Point |
| Pool Confirmation Status | Crude Oil Density Type | TMD | Geol Approval Flag |

## 12. Deployment and Environment Management

Effective deployment and environment management ensure consistent application performance across all stages. This section outlines BCER guidelines for environment configurations and resource management.

* **Standard Environments**: All custom applications must be deployed across Development, Test, UAT, Staging, and Production environments.
* **Environment Consistency**: Maintain uniform configurations to prevent environment-specific issues.

**12.1 Configuration Management**

* **Secrets and Configurations**: Use OpenShift's secrets and configuration management features securely.
* **Immutable Infrastructure**: Treat infrastructure components as immutable to prevent configuration drift.

**12.2 Scalability and Resource Management**

* **Scalability**: Leverage OpenShift's capabilities to scale applications and ensure high availability.
* **Resource Management**: Define resource requests and limits for containers to optimize performance.

**12.3 Angular SPA Hosting**

* **Integration**: Host Angular Single Page Applications within ASP.NET 8 runtime environments for seamless integration.

## 13. Monitoring and Logging

Monitoring and logging are essential for maintaining system health and performance. This section details BCER guidelines for centralized logging and monitoring.

**13.1 Centralized Logging**

* **Logging Solution**: Implement a centralized logging system to collect and analyze logs. Logging must include (at a minimum) the following:
	+ Authorized Access
		- UserID
		- Date/Time
		- Event Type
		- Resource Access
	+ Privileged Operations
		- Admin Account access
		- System Start and Stop
		- Restricted Information Access
	+ Failed Access Alerts
		- Logon
		- Data Access
* **Structured Logging**: Use structured logging formats for efficient analysis. Use OpenShift based standards (Loki) for logging structure for all OpenShift based applications.

**13.2 Application Monitoring**

* **Performance Metrics**: Monitor key performance indicators such as response times and error rates.
* **Alerts and Notifications**: Configure alerts for critical issues to enable rapid response.

**13.3 Infrastructure Monitoring**

* **Resource Utilization**: Monitor infrastructure components for resource utilization and availability.

**13.4 Database Auditing**

* **Mandatory Audit:** Audit records are created in \_ADT tables whenever a user or the system adds, updates, or deletes a record saving the type of change, application, username (not proxy account) and date/time.
* All new tables must include ADT functionality to provide a history of data changes.  In cases where existing tables are being modified which presently do not have ADT functionality implemented, reasonable efforts must be made to add them.  This functionality should include the following fields:
	+ USER ID making change (where possible, applications using system accounts must also identify the user behind the change).
	+ Date/Time of change
	+ Type of change (UPDATE/INSERT/DELETE)

## 14. Version Control Practices

Effective version control practices enhance collaboration and code integrity. This section outlines BCER branching strategy, commit message conventions, and code review processes.

* **Branching Strategy**:
	+ **Feature and Bugfix Branches**: Use branches that relate to Jira tickets for development work.
	+ **Pull Requests**: Merge all feature and bugfix branches into the development branch via pull requests.
* **Development Branch**: Serves as the integration branch for ongoing development.

**14.1 Main Branch**

* **Release Management**: Live system code is merged into the main branch after go-live.
* **Stability**: The main branch reflects the current production state and must remain stable.

**14.2 Commit Messages**

* **Clarity**: Write clear and descriptive commit messages referencing the associated Jira tickets.
* **Traceability**: Ensure commit messages provide context to understand changes.

**14.3 Code Reviews**

* **Mandatory Reviews**: Require code reviews for all pull requests to ensure code quality. Appropriate code review resourcing must be determined with BCER IDS.
* **Reviewers**: Involve appropriate team members or subject matter experts.

## 15. Documentation and Knowledge Sharing

Proper documentation and knowledge sharing promote transparency and continuous learning. This section emphasizes the importance of maintaining up-to-date documentation and fostering collaboration.

* **Code Documentation**: Maintain inline code documentation and generate API documentation from code annotations.
* **Technical Documentation**: Keep architectural diagrams, design documents, and setup guides updated and accessible.
* **Knowledge Base**: Utilize internal platforms (Azure DevOps GIT (markdown), SharePoint) for sharing best practices and troubleshooting guides.
* **Collaboration**: Encourage knowledge sharing through code reviews, peer programming, and regular team meetings.
* **Data Dictionary**: A central repository of Terminology must be maintained, with the business data owner being responsible for new terms being added as required.

## 16. Continuous Improvement

Continuous improvement ensures that the BCER adapt to changes and enhance its processes. This section highlights the Regulator’s commitment to ongoing evaluation and optimization.

* **Feedback Loops**: Incorporate feedback mechanisms throughout the development lifecycle.
* **Metrics and KPIs**: Track code quality metrics, build success rates, and deployment frequencies.
* **Process Optimization**: Regularly review and refine processes to improve efficiency.
* **Innovation**: Encourage teams to explore new technologies and methodologies.

## 17. Exemptions and Alternative Tools

While these guidelines promote consistency, The BCER recognizes that alternative tools may be suitable in certain scenarios. This section outlines the process for proposing and approving exemptions.

**17.1 Review Process**

* **Proposal Submission**: Teams must submit a detailed proposal outlining the alternative tool or technology.
* **Evaluation**: The Development Guidelines Committee will assess alignment with organizational goals and potential benefits.
* **Approval**: Exemptions may be granted based on the review.

**17.2 Considerations for Exemptions**

* **Business Value**: The alternative should provide significant business value.
* **Compatibility**: It should integrate well with existing systems.
* **Support and Maintenance**: There should be a plan for ongoing support.
* **Security and Compliance**: The alternative must meet security and compliance requirements.

## 18. Future State and Strategic Initiatives

To stay ahead in the industry, we are exploring enhancements to our technology stack and processes. This section outlines our strategic initiatives.

**18.1 API Management Evolution**

* **Transition to Red Hat 3scale**: Evaluating migration from WSO2 API Manager to Red Hat 3scale.
* **Potential Benefits**: Improved scalability, better integration with OpenShift, and advanced analytics.

**18.2 DevOps Platform Modernization**

* **Migration to Azure DevOps Cloud**: Considering moving from on-premises Azure DevOps Server to Azure DevOps Services in the cloud.
* **Potential Benefits**: Enhanced collaboration, scalability, and cost efficiency.

**18.3 Identity and Access Management Strategy**

* **Exploring Microsoft Entra ID**: Reviewing the potential transition from Keycloak to Microsoft Entra ID.
* **Potential Benefits**: Unified identity platform, enhanced security features, and improved user experience.

**18.4 Action Plan**

* **Assessment and Planning**: Conduct feasibility studies and involve key stakeholders.
* **Pilot Programs**: Implement pilot projects to validate benefits and identify challenges.
* **Roadmap Development**: Develop a phased implementation plan.
* **Communication**: Keep all teams informed about progress and changes.

## 19. Conclusion

Adhering to these system development guidelines ensures that BCER software solutions are robust, secure, and efficient. By following these guidelines, the Regulator fosters a culture of excellence and continuous improvement, enabling the BCER to meet business objectives and deliver value to its stakeholders.

These guidelines are subject to periodic review and updates to stay aligned with technological advancements and organizational needs. All internal and external resources are expected to understand and apply these guidelines in their daily work.

## Appendix

OWASP standards : <https://github.com/OWASP/secure-coding-practices-quick-reference-guide/releases/download/v2.0.1/OWASP_SCP_Quick_Reference_Guide.en-US.pdf>

1. Temporary data that exists only for a short duration and is not permanently stored [↑](#footnote-ref-1)