WE BUILD CONSORTIUM

Deliverable D4.2

Initial Source Code Repositories Setup and Contribution Guidelines

Version: 4.2
December 2025

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Project and document data

Item	Description
Project title	**W**allet **E**cosystem for **B**usiness and payments **U**se cases on **I**dentification, **L**egal representation and **D**ata sharing
Grant Agreement no	Project 101224751 — WE BUILD
Deliverable title	D4.2 Initial Source Code Repositories Setup and Contribution Guidelines
Deliverable type	OTHER
Responsible party	NATIONAL INFRASTRUCTURES FOR RESEARCH AND TECHNOLOGY (999937887)
Authors	Thodoris Papadopoulos, Spyros Ligouras
Contributing parties	GRNET
Reviewers	
Dissemination level	PU - Public

History of changes

Version	Date	Changes	Author
1.0	October 2025	Initial version	GRNET
2.0	October 2025	Enhanced with content from 7 policy documents: GitHub Policies and Guidelines, Contribution Guidelines, Onboarding Procedures, Repository Management, Licensing Guidelines, Security and Compliance, and Branching and Workflow	GRNET
3.0	October 2025	Incorporated insights from best practices documents covering branching strategies, version control workflows, secrets management, repository management, and general GitHub best practices	GRNET
4.0	October 2025	Replaced ASCII diagrams with Mermaid diagrams and added additional visual diagrams for improved clarity and understanding	GRNET
4.1	November 2025	Updated to reflect actual WE BUILD repository structure, licenses, and communication channels	GRNET
4.2	December 2025	Improved structure, eliminated redundancies, and enhanced readability	GRNET

Table of contents

List of figures

List of tables

1. Executive summary

This deliverable establishes the foundational framework for source code repository management within WE BUILD consortium, a European Union co-funded project focused on developing a wallet ecosystem for business and payment use cases. The document addresses the critical need for standardized practices in collaborative technical development across multiple partner organizations working on digital identity solutions.

The deliverable serves three primary purposes. First, it documents the establishment of a centralized GitHub organization at https://github.com/webuild-consortium/, which functions as the authoritative platform for all technical collaboration within the project. This organizational structure provides centralized access management, unified visibility across Work Packages, consistent branding for consortium outputs, and simplified archival capabilities for long-term preservation of project deliverables. Second, the document defines comprehensive policies and guidelines that govern how consortium members utilize GitHub, ensuring consistent practices across all Work Packages while maintaining security, compliance, and quality standards. Third, it establishes clear contribution workflows that enable effective collaboration among partners from different organizations and technical backgrounds.

At the time of publication, WE BUILD GitHub organization hosts ten repositories supporting various Work Packages and specialized technical groups. These repositories encompass architecture documentation, interoperability testing frameworks, technology standards, and group-specific resources for trust infrastructure, qualified trust service providers, semantics, and wallet providers. The repository structure reflects a distributed ownership model that recognizes the autonomy required by different technical areas while maintaining the benefits of shared organizational infrastructure. The policies outlined in this deliverable address critical aspects of collaborative development including branching strategies & workflows, pull request procedures requiring cross-partner review, comprehensive access management with role-based permissions, security practices and secrets management, licensing requirements, and compliance with European Union regulations including GDPR.

Document Structure. This deliverable is organized into four main chapters. Chapter 2 provides the introduction, establishing the scope and objectives of the deliverable, defining its target audiences, explaining the central role of GitHub in consortium operations, and describing the document's versioning approach as a living reference. Chapter 3 addresses the initial code repositories setup, documenting the scope of Task 4.8 activities, explaining the rationale for the chosen organizational structure, and describing the initial set of repositories established across Work Packages and technical groups. Chapter 4 presents the comprehensive GitHub policies and guidelines that form

the operational core of the document, covering governance structures, access management, working practices and workflows, repository management procedures, licensing and intellectual property considerations, security and compliance requirements, contribution guidelines, onboarding and offboarding procedures, and best practices summary with supporting resources. The appendices provide quick reference materials including checklists, common commands, troubleshooting guidance, a glossary of technical terms, and required European Union funding acknowledgment language. This structure enables readers to quickly locate relevant information based on their role and immediate needs while providing comprehensive coverage of all aspects of the consortium's GitHub management framework.

2. Introduction

2.1 Scope and goal

This document addresses multiple audiences within WE BUILD consortium, each with distinct needs and perspectives on the repository management framework.

The primary audience consists of technical contributors including software developers, researchers, and engineers who actively create code, specifications, and technical documentation. These individuals require clear guidance on development workflows, contribution procedures, and quality standards. The document provides them with practical instructions for common tasks such as creating branches, submitting pull requests, conducting code reviews, and managing issues. Technical contributors benefit from understanding not only the "how" of these processes but also the "why" behind policy decisions, enabling them to work more effectively within the established framework.

Work Package Leads and Group Leads represent a secondary audience with management and oversight responsibilities. These individuals need to understand how to organize repositories within their domains, manage team access, coordinate contributions from multiple partners, and ensure that deliverables meet quality and compliance requirements. The document provides them with guidelines for repository creation, access management, and quality assurance that support their coordination responsibilities while maintaining consistency with consortium-wide standards.

The WP1 Project Management team requires a comprehensive view of the entire GitHub infrastructure to support their governance and coordination roles. This document provides them with the information needed to understand organizational settings, resolve conflicts between teams, ensure compliance with best practices, and support the evolution of policies as the project progresses. The clear documentation of roles, responsibilities, and decision-making authority enables effective escalation and resolution of issues that arise during collaborative work.

The European Commission and external evaluators represent an important audience for demonstrating transparency and accountability in the management of publicly funded research. This document serves as evidence of the consortium's systematic approach to collaborative technical development, its commitment to quality assurance and security, and its compliance with relevant regulations and best practices. The comprehensive nature of the policies and procedures documented here supports the consortium's credibility and facilitates external evaluation of project processes.

Finally, new consortium members joining the project at later stages benefit from this document as a comprehensive onboarding resource that explains the entire WE BUILD GitHub collaboration framework. Rather than learning through trial and error or

fragmented communications, new members can quickly understand the established practices and begin contributing effectively within the existing framework.

2.2 Role of GitHub in WE BUILD Consortium

GitHub serves as the central platform for technical collaboration across WE BUILD consortium, providing essential infrastructure that enables partners from different organizations to work together seamlessly on complex technical challenges. The selection of GitHub as the primary collaboration platform reflects careful consideration of the consortium's needs for version control, transparency, collaborative development, and long-term preservation of technical outputs.

The platform functions as the authoritative repository for all technical artifacts produced by the consortium. This includes source code for software components and services being developed to support digital identity use cases, formal specifications defining interfaces, protocols, and standards that ensure interoperability between implementations, architecture documentation and decision records that capture the evolution of technical approaches, test frameworks and environments that enable validation of implementations against specifications, and supporting documentation including tutorials, guides, and explanatory materials that facilitate understanding and adoption of consortium outputs.

Beyond simple file storage, GitHub provides sophisticated version control capabilities through its integration with Git. Every change to technical artifacts is tracked with complete history, including who made the change, when it was made, and why it was necessary. This comprehensive versioning supports multiple critical functions within the project. It enables traceability required for project governance and compliance with funding requirements. It facilitates collaboration by allowing multiple partners to work on different aspects of the same technical artifacts simultaneously. It provides accountability by maintaining clear records of contributions from each partner organization. It supports quality assurance by enabling review of changes before they are incorporated into stable versions.

The collaborative features of GitHub are particularly valuable for a multi-partner consortium. The pull request mechanism provides structured workflows for proposing, discussing, reviewing, and incorporating changes to technical artifacts. This ensures that changes undergo appropriate peer review before being accepted, that technical discussions are documented and accessible to all consortium members, and that contributions from different partners are integrated in a controlled and traceable manner. The issue tracking system enables systematic management of bugs, feature requests, and technical questions, ensuring that nothing falls through the cracks during complex multi-partner development efforts.

GitHub also serves important functions for project management and visibility. Project boards enable Work Packages and Groups to organize and track their activities, providing transparency into progress and helping identify bottlenecks or areas requiring additional attention. The platform's access control mechanisms support the principle of least privilege while enabling appropriate collaboration, ensuring that sensitive work remains protected while facilitating broad participation in appropriate areas. Integration capabilities with continuous integration and deployment systems enable automated testing and quality checks that maintain technical standards without requiring manual intervention for routine validations.

The choice of GitHub also reflects considerations of sustainability and broader impact. As a widely used platform in the open source community and industry, GitHub ensures that consortium outputs are accessible to potential adopters and contributors beyond the immediate consortium membership. This supports the European Commission's objectives for broad dissemination and exploitation of research results. The choice of the GitHub platform provide confidence that consortium outputs will remain accessible well beyond the project's completion date.

2.3 Versioning

This deliverable represents a snapshot of a living documentation that will continue to evolve throughout the WE BUILD project lifecycle. The policies, procedures, and guidelines documented here are designed to be practical and sustainable, but the consortium recognizes that adjustments may be necessary as technical work progresses, new challenges emerge, or better practices are identified.

The authoritative, continuously updated version of the GitHub policies and guidelines is maintained in the WE BUILD-github-policies repository within the WE BUILD GitHub organization. This repository serves as the single source of truth for all policy matters related to GitHub usage within the consortium. All changes to policies and procedures are implemented through pull requests to this repository, ensuring that modifications undergo the same review and approval processes as other technical work. This approach maintains transparency and accountability in policy evolution while enabling timely responses to emerging needs.

Major revisions to the policies that significantly impact consortium operations will be communicated through project-wide channels and, where appropriate, published as updated versions of this deliverable in the project's shared repository. Minor clarifications, corrections, or additions that do not fundamentally change existing policies will be tracked through the repository's commit history without necessarily triggering formal document updates. This balanced approach ensures that consortium members always have access to current guidance while avoiding unnecessary administrative burden from frequent formal revisions.

All consortium members are encouraged to contribute to the improvement of these policies and guidelines. Feedback based on practical experience is invaluable for ensuring that the framework remains effective and appropriate for the consortium's needs. Partners who identify areas where policies could be clarified, simplified, or enhanced are invited to open issues in the WE BUILD-github-policies repository or contact the GitHub Technical Coordinator directly. This participatory approach to policy development ensures that the framework reflects the collective wisdom of the consortium and adapts appropriately to changing circumstances

3. Initial Code Repositories Setup

3.1 Task Context and Objectives

This deliverable directly addresses Task 4.8 within Work Package 4, which focuses on the setup, organization, and maintenance of source code repositories for WE BUILD consortium. Task 4.8 represents a foundational activity that enables all subsequent technical collaboration across the consortium, providing the infrastructure and frameworks upon which partners build their collective technical outputs. It's primary objective is to establish and maintain a comprehensive repository system that supports collaborative development between partners, facilitates documentation versioning and management, accommodates multiple development environments with appropriate separation and protection, and enables efficient deployment of key components and services related to the implementation of use cases defined across the project's Work Packages. These objectives reflect the complex nature of multi-partner European research projects, where technical work must be coordinated across organizational boundaries while maintaining appropriate quality, security, and compliance standards.

The task encompasses several key activities that have been completed for this initial deliverable.

- Repository setup and initial configuration involved creating the GitHub organization, establishing naming conventions and organizational structures, configuring organization-level settings including security requirements and access controls, and setting up initial repositories for early active Work Packages and technical Groups.
- Definition of repository structure required analyzing the project's technical architecture and Work Package organization, identifying appropriate boundaries for repositories that balance autonomy with coordination, establishing relationships between repositories where components or specifications depend on each other, and defining standards for repository organization that ensure consistency across the consortium.
- Development of contribution guidelines and standards represented a major focus of the task, resulting in the comprehensive policies documented in Chapter 4 of this deliverable. This work involved researching best practices from successful open source projects and industry standards, adapting generic practices to the specific needs and constraints of WE BUILD consortium, ensuring alignment with European Union requirements for publicly funded research, and creating practical, actionable guidance that technical contributors can follow without excessive burden.
- Access management and security considerations required defining roles and permission levels appropriate for different types of contributions, establishing processes for granting and revoking access as partners join or leave the consortium, implementing mandatory security measures including two-factor authentication

requirements, and ensuring compliance with data protection regulations including GDPR.

- The collaborative development support dimension of the task focused on enabling effective cooperation between partners. This included establishing workflows for common development activities such as branching, pull requests, and code review, creating mechanisms for technical discussion and decision-making that span organizational boundaries, providing guidance on resolving conflicts and disagreements that naturally arise in collaborative work, and fostering a culture of constructive peer review and shared responsibility for quality.

For the remaining course of the project, the task will continue its work as ongoing maintenance and support will be required throughout the project lifecycle, including monitoring repository usage and addressing issues as they arise, adapting policies and procedures based on practical experience, providing support to consortium members who encounter difficulties or uncertainties, and ensuring that the repository infrastructure continues to meet the consortium's evolving needs.

3.2 GitHub Organization Structure and Rationale

The decision to establish a dedicated GitHub organization for WE BUILD consortium, rather than relying on repositories distributed across partner organizations, reflects careful consideration of the project's needs and the lessons learned from other collaborative European research initiatives. The unified organizational approach provides several critical benefits that significantly enhance the consortium's ability to work together effectively.

Centralized access management represents perhaps the most important practical benefit of the organizational approach. With repositories scattered across different partner organizations, managing access rights becomes a complex and error-prone process requiring coordination with multiple administrative systems and procedures. Each partner organization may have different policies, approval processes, and timing for access changes, creating friction and delays that impede collaborative work. The unified organization simplifies this dramatically by providing a single point of control where access can be granted or revoked quickly and consistently. When new partners join the consortium or existing partners bring additional personnel onto the project, the Technical Coordinator can provide appropriate access immediately without navigating multiple organizational bureaucracies. Similarly, when the consortium structure changes or individuals leave the project, access can be revoked comprehensively and immediately, maintaining security without requiring coordination across multiple systems.

Unified visibility across all technical activities enables consortium leadership to maintain appropriate oversight and support for the project's technical work. With a

distributed repository approach, leadership would need to track numerous repositories across different organizations, each potentially with different visibility settings, organizational structures, and naming conventions. This fragmentation makes it difficult to understand the overall state of technical work, identify areas where support may be needed, or recognize dependencies and potential conflicts between different activities. The unified organization provides a single place where leadership can view all technical activities, understand how different pieces of work relate to each other, and ensure that the overall technical program remains coherent and well-coordinated.

Consistent branding and identity present important benefits for external perception and long-term impact. The WE BUILD project represents a significant investment of European Union funds and aims to produce outputs that will have lasting impact on digital identity infrastructure. A unified organizational presence on GitHub demonstrates professionalism and coherence, making it easier for external stakeholders including potential adopters, other research projects, standardization bodies, and the broader open source community to understand and engage with the consortium's work. Rather than encountering a fragmented collection of repositories across different organizations with varying levels of documentation and discoverability, external stakeholders find a cohesive collection of work that clearly represents the WE BUILD project's contributions.

Simplified archival and long-term preservation considerations become particularly important as the project approaches completion. European research projects have obligations to ensure that outputs remain accessible for specified periods after project completion. With repositories distributed across partner organizations, this creates complex dependencies where the consortium's ability to meet its preservation obligations depends on each partner organization maintaining their repositories appropriately. Partner organizations may have changes in priorities, resource constraints, or even organizational restructuring that could threaten the preservation of repositories they host. The unified organization provides a single entity with clear responsibility for long-term preservation, making it straightforward to ensure that all outputs remain accessible regardless of changes in individual partner organizations' circumstances.

Technical benefits of the organizational approach include simplified continuous integration and deployment configuration, consistent security scanning and vulnerability management across all repositories, unified access to organizational-level features such as GitHub Actions and storage, and the ability to implement organization-wide standards and automations that reduce repetitive configuration work. These technical advantages compound over time, making the development infrastructure more maintainable and reliable.

The organizational structure chosen for WE BUILD balances centralized coordination with distributed ownership and autonomy. While the organization itself is centrally managed, individual repositories remain under the control of the Work Packages and technical Groups that own the respective technical domains. This distributed ownership model recognizes that different technical areas require autonomy to make decisions appropriate to their specific contexts while still benefiting from the shared organizational infrastructure and consistent policies.

3.2 Initial Repositories

At the time of this deliverable's publication, WE BUILD organization hosts ten repositories serving different purposes and technical domains.

Repository	Description	Visibility
`.github`	Organization-level GitHub configuration	Public
`eudi-wallet-rulebooks-and-schemas`	Rulebooks and data schemas for WE BUILD LSP Use Cases	Public
`webuild-github-policies`	GitHub Policies and Guidelines for WE BUILD Consortium	Private
`wp3-technology-standards`	WP3 Technology & Standards Working Group	Private
`wp4-architecture`	Drafts and ADRs for Large Scale Pilot Use Cases	Public
`wp4-interop-test-bed`	Interoperability Test Bed implementing conformance	Public
`wp4-qtsp-group`	Public resources shared within WP4 QTSP Group	Public
`wp4-semantics-group`	Semantics Group resources	Public
`wp4-trust-group`	Public resources shared within WP4 Trust Infrastructure Group	Public
`wp4-wallets-group`	Repository for WP4 Wallet Providers Group	Public

Table 1: Initial WE BUILD Consortium GitHub Repositories Overview

This initial set of repositories reflects the active areas of technical work at this stage of the project. Additional repositories will be created as new technical activities launch and as existing work evolves to require separate spaces. The repository creation process defined in Chapter 4 ensures that new repositories are established systematically with appropriate configuration, documentation, and access controls rather than proliferating ad hoc.

4. GitHub Policies and Guidelines

This chapter consists of a snapshot of the online GitHub Policies and Contribution Guidelines maintained online in the dedicated WE BUILD GitHub organization repository.

4.1 Purpose and Objectives

These policies and guidelines establish the foundation for effective, secure, and compliant use of GitHub within WE BUILD Consortium. By following these practices, partners can collaborate effectively across organizational boundaries while maintaining the quality, security, and integrity of technical work.

Success in collaborative technical work requires more than just tools—it requires shared understanding of how those tools should be used. These guidelines provide that shared understanding, enabling partners to focus on the technical challenges at hand rather than on process questions.

As the consortium's work evolves, these policies may be refined based on practical experience and changing needs. Partners are encouraged to provide feedback on these guidelines to help ensure they remain practical and effective.

The Policies and Guidelines for GitHub collaboration are maintained in the `webuild-policies` *repository. For questions or suggestions, please open an issue or contact the Technical Coordinator (email: webuild-github-support@grnet.gr).*

4.1.1 Primary Objectives

The policies and guidelines presented in the following sections serve multiple critical purposes within WE BUILD Consortium:

- **Establish a Common Framework.** Create a unified approach to using GitHub that all consortium partners can follow. This common framework reduces friction in cross-organizational collaboration and ensures that all partners understand the expected workflows and procedures.
- **Support Collaborative Technical Work.** Enable seamless collaboration between partners on technical deliverables. GitHub's collaborative features—including pull requests, code reviews, and issue tracking—are leveraged to facilitate meaningful technical discussions and joint development efforts.
- **Maintain Version Control Excellence.** Ensure that all technical artifacts are properly versioned, with clear histories of changes, contributors, and decision points. This version control discipline is essential for maintaining the integrity of technical work and understanding how solutions evolved over time.

- **Enable Comprehensive Documentation.** Provide a structured environment where technical documentation lives alongside the artifacts it describes. This co-location of code and documentation ensures that information remains current and accessible to all partners.
- **Ensure Traceability and Accountability.** Maintain clear records of who contributed what, when, and why. This traceability is essential for project governance, quality assurance, and demonstrating compliance with consortium agreements and funding requirements.
- **Uphold Compliance Standards.** Ensure that all GitHub activities comply with project governance requirements, data protection regulations (including GDPR), and consortium-wide policies regarding intellectual property and information security.

4.1.2 Scope of Application

These policies apply to:

- All repositories under the `webuild-consortium` GitHub organization
- All consortium members with GitHub access
- All Work Packages and technical Groups
- All types of content (code, documentation, specifications, tests)
- All development environments (DEV, TEST, PROD)

4.1.3 Relationship to Other Policies

These GitHub policies complement and must be read in conjunction with:

- WE BUILD Consortium Agreement
- Data Management Plan
- Intellectual Property Rights Agreement
- Security and Privacy Policies
- EU Grant Agreement requirements

4.2 Governance and Organizational Structure

4.2.1 Consortium GitHub Organization Hierarchy

The WE BUILD Consortium maintains a dedicated GitHub organization at `https://github.com/webuild-consortium/`. This organization serves as the authoritative location for all technical collaboration within the project.

The governance structure follows a clear hierarchy:

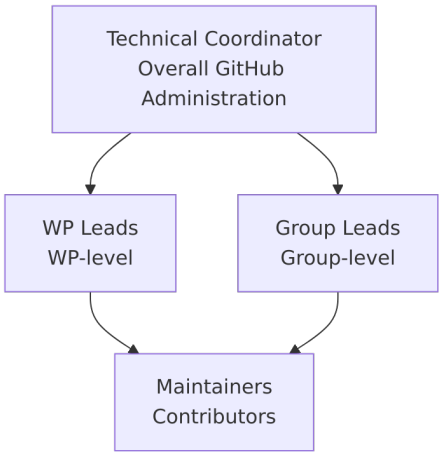

Figure 1: WE BUILD GitHub Organization Governance Hierarchy and Structure

Repositories typically contain:

- **Technical Specifications:** Formal definitions of interfaces, protocols, and standards
- **API Definitions**: Machine-readable and human-readable API documentation
- **Test Tools and Environments**: Software and configurations for testing implementations
- **Example Implementations**: Reference code demonstrating how to implement specifications
- **Supporting Documentation**: Guides, tutorials, and explanatory materials

4.2.2 Roles and Responsibilities

4.2.2.1 Technical Coordinator

Responsibilities: - Manage GitHub organization settings and security - Add and remove members from the organization - Approve new repositories and major changes - Update policies as needed - Resolve conflicts between teams

GitHub Permission: Organization Owner

Contact: webuild-github-support@grnet.gr

4.2.2.2 Work Package (WP) Leads

Responsibilities: - Oversee repositories within their Work Package - Approve repository creation requests - Manage team member access within their WP - Ensure quality of WP deliverables

GitHub Permission: Admin on WP team

4.2.2.3 Group Leads

Responsibilities: - Manage day-to-day repository operations - Add and remove team members within their group - Coordinate code reviews - Manage issues and project boards - Maintain up-to-date documentation

GitHub Permission: Admin on repositories they manage

4.2.2.4 Maintainers

Responsibilities: - Review and merge pull requests - Triage and manage issues - Create releases - Update documentation - Support contributors

GitHub Permission: Write access with merge rights

4.2.2.5 Contributors

Responsibilities: - Submit code and documentation changes - Report issues and bugs - Participate in discussions - Review pull requests from other contributors

GitHub Permission: Write or Read (depending on repository)

4.2.3 Decision-Making Authority

Decision	Who Decides	Who to Consult
Organization settings	Technical Coordinator	WP Leads
Create new repository	WP/Group Lead	Technical Coordinator (for approval)
Delete repository	Technical Coordinator	WP Lead, Group Lead
Add team members	Group Lead	-
Make repository public	WP Lead + Technical Coordinator	-
Change license	Technical Coordinator	WP Lead
Update policies	Technical Coordinator	WP Leads

Table 2: Decision-Making Authority Matrix for Repository Management

4.2.4 Essential Repository Components

Every repository within the consortium organization must include these foundational documents:

README File – The README file serves as the entry point for anyone accessing the repository. It should clearly explain the repository's purpose, provide orientation for new contributors, and link to other relevant documentation. A well-crafted README significantly reduces the barrier to entry for partners joining ongoing technical work.

LICENSE File – The LICENSE file specifies the legal terms under which the repository's contents can be used, modified, and distributed. The default license is Apache License 2.0, which is used by most WE BUILD repositories. Clear licensing is essential for ensuring that consortium outputs can be properly utilized by partners and, where appropriate, by the broader community.

CONTRIBUTING Guide – The CONTRIBUTING guide explains how partners can contribute to the repository. It should detail the expected workflow for proposing changes, coding standards or documentation conventions to follow, and the review process that contributions will undergo.

Supporting Documentation – Additional documentation may include architecture diagrams, decision records, meeting notes, or any other materials that help partners understand and contribute to the technical work effectively.

4.3 Access Management and Role Definitions

4.3.1 Account Requirements

All consortium members participating in GitHub activities must have their own GitHub accounts.

You may use your personal GitHub account. There is no requirement to create a separate work account or use a work email address in your GitHub account.

Configure your Git client to ensure commits are properly attributed:

```
git config --global user.name "Your Full Name"
git config --global user.email "your.email@organization.com"
```

4.3.2 Two-Factor Authentication (2FA)

Two-factor authentication is **required for all consortium members**. This is a mandatory security requirement that significantly enhances account security.

Setup Steps

1. Go to GitHub Settings → Password and authentication
2. Click "Enable two-factor authentication"
3. Choose authentication method:
   - **Recommended:** Authenticator app (Google Authenticator, Authy, 1Password)
   - **Alternative:** Security key (YubiKey, etc.)
   - **Fallback:** SMS (least secure)
4. Save recovery codes securely
5. Verify 2FA is enabled

Recovery Codes: - Store in secure password manager - Keep offline backup - Never share with anyone - Generate new codes if compromised

4.3.3 Access Request Process

4.3.3.1 For New Consortium Members

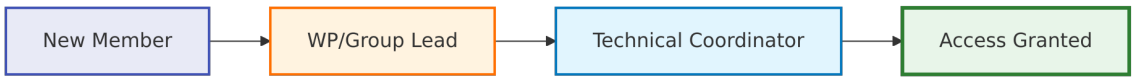

Figure 2: Access Request Process Flow for New Consortium Members

Process:

1. New member completes onboarding
2. WP/Group Lead submits access request to Technical Coordinator
3. Technical Coordinator adds member to organization
4. WP/Group Lead adds member to appropriate team(s)
5. Member receives welcome email with guidelines

Access Request Template:

```
Subject: GitHub Access Request - [Member Name]

## Member Information
- Name: [Full Name]
- GitHub Username: [@username]
- Email: [work.email@organization.com]
- Organization: [Partner Organization]
- Work Package: [WP Number]
- Group: [Group Name]
- Role: [Developer/Researcher/etc.]

## Access Requirements
- Organization Access: Yes
- Teams: [@webuild-consortium/[team-name]]
- Repositories: [List specific repos if limited access]
- Permission Level: [Read/Write/Admin]

## Duration
- Start Date: [YYYY-MM-DD]
- End Date: [YYYY-MM-DD or "Ongoing"]
```

4.3.4 Permission Levels

Level	Capabilities	Typical Role
Read	View code, clone, download	External collaborators, observers
Triage	Read + manage issues/PRs	Community managers
Write	Triage + push to branches	Contributors
Maintain	Write + manage repo settings	Maintainers
Admin	Full control except deletion	Group Leads
Owner	Full organizational control	Technical Coordinator

Table 3: GitHub Permission Levels and Associated Capabilities

4.3.5 Access Review and Maintenance

Access rights are reviewed as needed to ensure that:

- Permissions remain appropriate for each member's current role in the consortium
- Former partners or members who have left the consortium have their access revoked
- New partners receive appropriate access in a timely manner
- The principle of least privilege is maintained (members have only the access they need)

Review Process

WP/Group Leads review team membership when changes occur and submit access updates to the Technical Coordinator as needed.

4.3.6 Access Revocation

Access must be revoked immediately when: - Member leaves the consortium - Member changes role (no longer needs access) - Security incident involving the account - Violation of policies or code of conduct

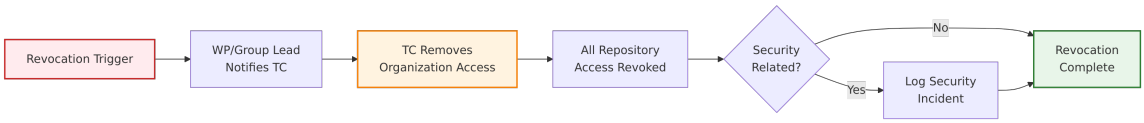

Figure 3: Access Revocation Process and Workflow

Revocation Process

1. WP/Group Lead notifies Technical Coordinator
2. Technical Coordinator removes organization access
3. All repository access automatically revoked
4. Incident logged if security-related

4.4 Working Practices and Workflows

4.4.1 Branching Strategy

The consortium employs the **Feature Branch Workflow** as the standard branching strategy. This approach balances stability with the need for ongoing development:

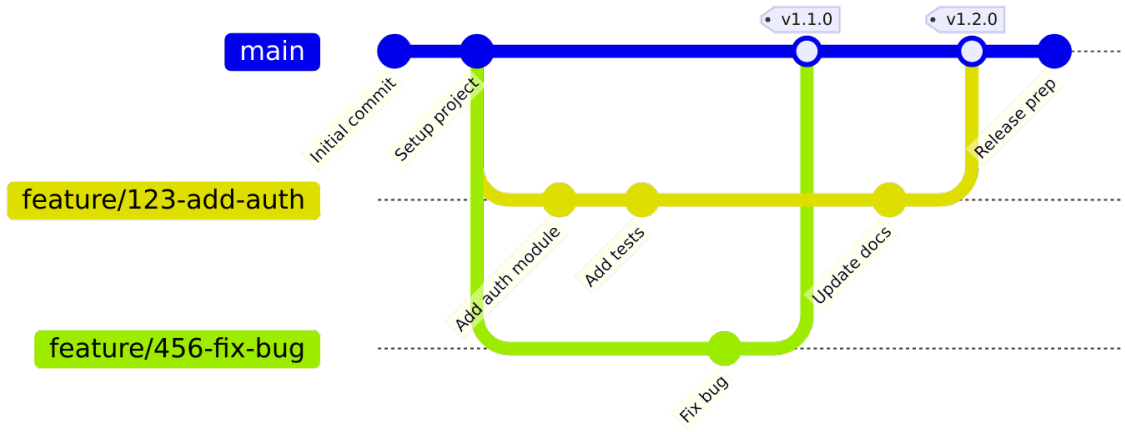

Figure 4: Feature Branch Workflow and Main Branch Integration

Main Branch – The `main` branch represents the stable, production-ready state of the repository. Content in the main branch should be thoroughly reviewed, tested, and approved. This branch serves as the authoritative source for released versions of specifications, stable APIs, and production-ready code.

Feature Branches – Feature branches are created for specific pieces of work—whether implementing a new capability, fixing a bug, or drafting a section of a specification. Feature branches should be short-lived (ideally less than 2 weeks) and are deleted after their changes have been merged. Keeping branches short-lived reduces integration conflicts and ensures code remains synchronized with the main branch.

Branch Naming Convention

```
<type>/<issue-number>-<description>

Examples:
feature/123-add-jwt-authentication
bugfix/456-fix-memory-leak
hotfix/789-patch-security-vulnerability
docs/012-update-api-documentation
```

Types: - `feature/` - New features - `bugfix/` - Bug fixes - `hotfix/` - Production hotfixes - `docs/` - Documentation only - `refactor/` - Code refactoring - `test/` - Test additions/changes

This branching strategy provides clear separation between stable outputs and individual contributions, making it easier to manage complex collaborative efforts. The strategy isolates work in progress from stable code, enables parallel development, and facilitates easier testing and rollback if needed.

4.4.2 Pull Request Workflow

All contributions to repositories must follow the pull request (PR) workflow. This requirement is fundamental to maintaining quality and fostering collaboration:

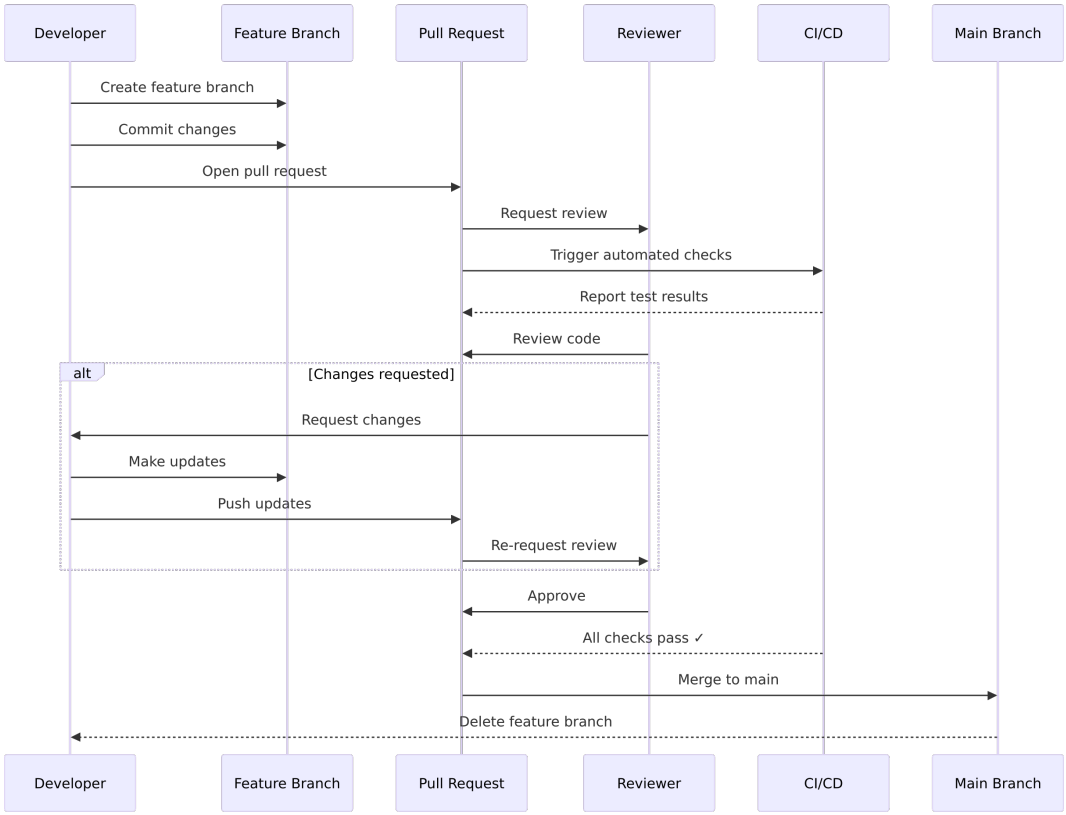

Figure 5: Pull Request Workflow from Creation to Merge

Why Pull Requests Matter

Pull requests serve multiple critical functions. They provide a structured mechanism for proposing changes, create a forum for technical discussion and review, maintain a permanent record of what changed and why, and ensure that at least two people (the author and a reviewer) have examined every change.

Pull Request Requirements

Every pull request should:

- Be linked to a relevant issue that explains the motivation for the change
- Include a clear description of what is being changed and why
- Be reviewed and approved by at least one other consortium partner before merging
- Pass any automated checks or tests that have been configured for the repository
- Address any feedback or concerns raised during the review process
- Be reviewed within 2 business days to maintain development momentum

Pull Request Template

```
## Description
Brief description of changes.

## Related Issues
Closes #[issue-number]
```

```
## Type of Change
- [ ] Bug fix
- [ ] New feature
- [ ] Breaking change
- [ ] Documentation update

## Testing
- [ ] Tests pass locally
- [ ] New tests added
- [ ] Manual testing completed

## Checklist
- [ ] Code follows style guidelines
- [ ] Self-review completed
- [ ] Documentation updated
- [ ] No new warnings
```

Cross-Partner Review

The requirement that pull requests be reviewed by another partner (not just another person from the same organization) is particularly important. This cross-partner review ensures that technical decisions consider multiple perspectives and that knowledge is shared across organizational boundaries.

Code Review Best Practices

For Authors: - Perform self-review before requesting review - Keep pull requests focused and reasonably sized - Provide context in the description - Respond to feedback constructively - Update the PR based on review comments

For Reviewers: - Review promptly (within 2 business days) - Be respectful and constructive - Focus on important issues - Explain your reasoning - Test changes when possible - Approve when satisfied with the quality

4.4.3 Commit Guidelines

Commit Message Format

```
<type>(<scope>): <subject>

<body>

<footer>
```

Commit Types

Type	Description	Example
`feat`	New feature	`feat: add user registration`
`fix`	Bug fix	`fix: resolve login timeout`
`docs`	Documentation only	`docs: update API guide`

Type	Description	Example
`style`	Code style changes	`style: format code with prettier`
`refactor`	Code refactoring	`refactor: simplify auth logic`
`test`	Test additions/changes	`test: add integration tests`
`chore`	Maintenance tasks	`chore: update dependencies`

Table 4: Git Commit Type Classification and Usage Examples

Example

```
feat(auth): implement JWT authentication

- Add JWT token generation
- Implement token validation middleware
- Add refresh token support
- Update API documentation

Closes #123
```

Best Practices

- Write clear, descriptive messages in imperative mood
- Use present tense (“add” not “added”)
- Keep subject line under 50 characters
- Explain what and why, not how
- Reference issues and PRs
- Make atomic commits (one logical change per commit)
- Commit frequently to avoid losing work
- Each commit should represent a complete, working state

Atomic Commits: Keep commits atomic – each commit should represent a single unit of work. This makes it easier to: - Understand the purpose of each change - Review code more effectively - Revert specific changes without side effects - Track down bugs using git bisect

4.4.4 Issue Tracking

GitHub Issues serve as the primary mechanism for tracking technical work, documenting decisions, and managing discussions:

Issue Purposes

Issues are used to:

- **Track Technical Progress:** Break down large technical efforts into manageable, trackable pieces of work

- **Document Discussions:** Capture technical discussions and decisions in a searchable, permanent format
- **Manage Updates:** Coordinate changes to specifications, APIs, or implementations
- **Report Problems:** Document bugs, inconsistencies, or areas needing improvement
- **Propose Enhancements:** Suggest new features or improvements to existing work

Issue Labels

Standard Labels:

Label	Description
`bug`	Something isn't working
`enhancement`	New feature or request
`documentation`	Documentation improvements
`question`	Further information requested
`help wanted`	Extra attention needed
`good first issue`	Good for newcomers
`security`	Security-related
`priority: high`	Critical, needs immediate attention
`priority: medium`	Important, should be addressed soon
`priority: low`	Nice to have, can wait

Table 5: Standard GitHub Issue Labels and Definitions

Issue Best Practices

Effective issue management requires:

- Clear, descriptive titles that summarize the issue at a glance
- Detailed descriptions that provide sufficient context for others to understand the issue
- Appropriate labels to categorize and prioritize issues
- Assignment to specific individuals when someone is actively working on the issue
- Regular updates to keep other partners informed of progress
- Closure with a summary of the resolution when work is complete
- Linking related issues and pull requests for traceability

4.4.5 Version Tagging and Releases

Version tags mark important milestones in the evolution of technical artifacts. Tags should follow semantic versioning principles:

Semantic Versioning

```
MAJOR.MINOR.PATCH

Example: 1.2.3
```

Version Increments

- **MAJOR (1.0.0 → 2.0.0):** Breaking changes
- **MINOR (1.0.0 → 1.1.0):** New features (backwards compatible)
- **PATCH (1.0.0 → 1.0.1):** Bug fixes (backwards compatible)

Pre-release Versions

- `1.0.0-alpha.1` - Alpha release
- `1.0.0-beta.1` - Beta release
- `1.0.0-rc.1` - Release candidate

When to Create Tags?

Tags should be created for:

- **Major Releases** – Significant versions of specifications or software that represent substantial milestones
- **Stable Snapshots** – Points in time when the repository contents are known to be stable and suitable for reference
- **Deliverable Submissions** – Versions that correspond to formal project deliverables
- **Breaking Changes** – Versions that introduce incompatible changes to APIs or interfaces

Tag Documentation Each tag should be accompanied by release notes that explain what has changed since the previous version, highlight any breaking changes or important updates, and acknowledge contributors to that release.

Creating Tags

```bash
# Create annotated tag
git tag -a v1.0.0 -m "First production release"

# Push tag to remote
git push origin v1.0.0

# List all tags
git tag -l
```

4.3 Repository Management

4.5.1 Repository Lifecycle

Repositories progress through distinct lifecycle stages:

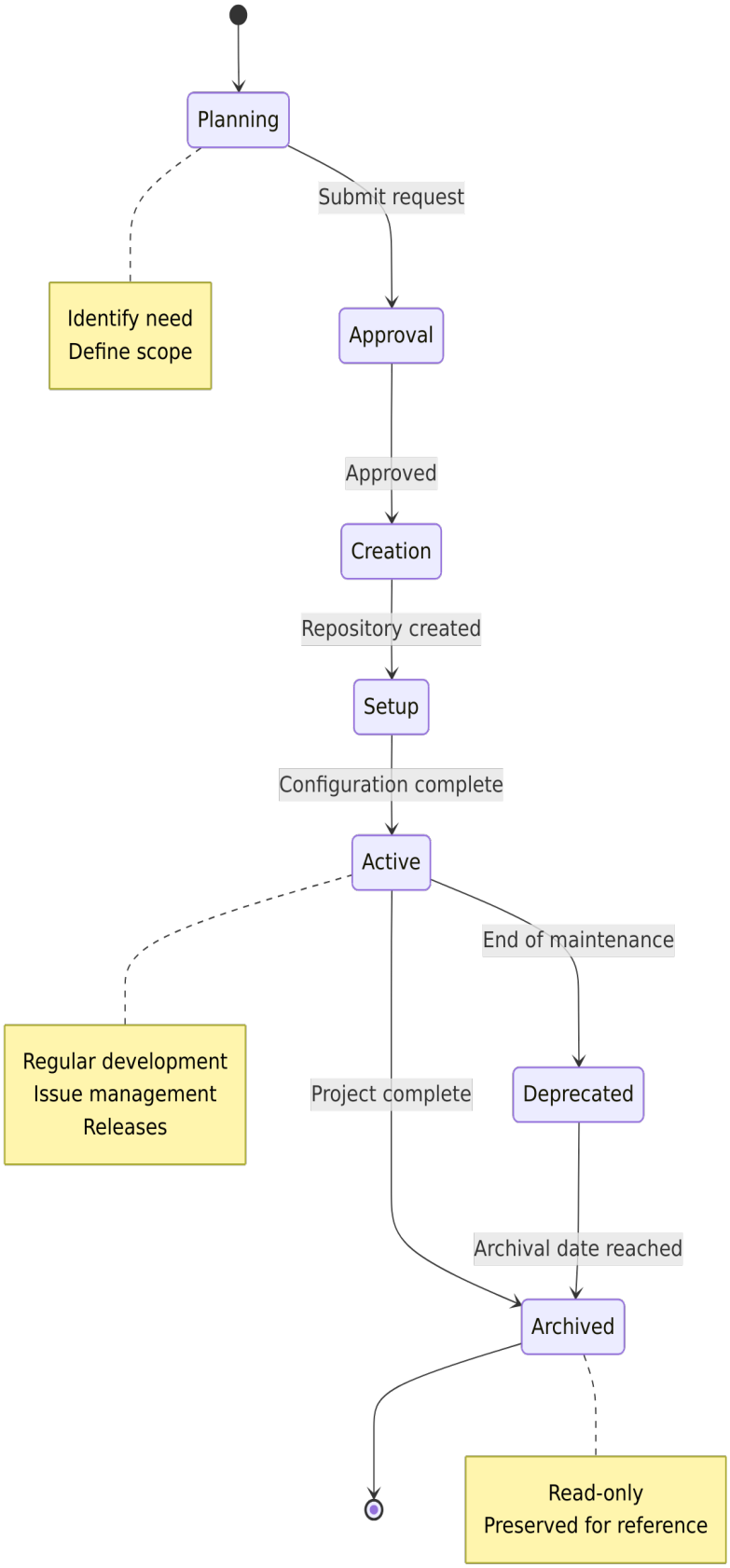

Figure 6: Repository Lifecycle Management Stages

Lifecycle Stages

1. **Planning:** Identify need, define purpose and scope
2. **Approval:** Submit creation request, review by WP/Group Lead, approval by Technical Coordinator
3. **Creation:** Repository created in organization, initial structure set up
4. **Setup:** Complete configuration, add required files, configure branch protection
5. **Active:** Regular development activity, issue management, releases
6. **Deprecated:** No longer actively maintained, security updates only
7. **Archived:** Read-only status, preserved for reference

4.5.2 Creating New Repositories

When to Create a New Repository

Create a new repository when:

- Starting a new component or service
- Separating concerns (e.g., API from UI)
- Creating standalone tools or libraries
- Establishing new group workspace
- Developing specifications or standards
- Setting up test frameworks

Do not create new repository for:

- Small features (use branches instead)
- Temporary experiments (use personal repos)
- Duplicate functionality
- Single files or scripts

Repository Naming Best Practices

Follow a clear naming convention: - Use descriptive, specific names - Include project or team prefix if applicable - Indicate technology stack when relevant - Use lowercase with hyphens (e.g., `wp4-trust-api`) - Avoid special characters - Keep names concise but meaningful

Repository Creation Request Template

```
## Repository Creation Request

### Basic Information
- **Repository Name:** [e.g., wp4-new-component]
- **Purpose:** [Brief description]
- **Type:** [Code/Documentation/Specification]
- **Work Package:** [e.g., WP4]
- **Group:** [e.g., Trust Infrastructure]
- **Visibility:** Public (default)
```

```
### Justification
Explain why this repository is needed and how it fits into the project.

### Scope
- What will be included
- What will NOT be included
- Boundaries and interfaces

### Team
- **Owner:** [Group Lead Name]
- **Maintainers:** [List names]
- **Initial Contributors:** [List names]

### Approvals
- [ ] Group Lead: [Name]
- [ ] WP Lead: [Name]
- [ ] Technical Coordinator: [Pending]
```

4.5.3 Repository Configuration

4.5.3.1 Required Files

Every repository MUST include:

- **README.md** - Project overview and documentation
- **LICENSE** - Apache License 2.0 (default)
- **CONTRIBUTING.md** - Contribution guidelines
- **.gitignore** - Files to exclude from version control
- **CODE_OF_CONDUCT.md** - Community standards
- **SECURITY.md** - Security policy

4.5.3.2. Repository Topics

Classify repositories with topics to improve discoverability and organization:

- Navigate to repository Settings → About
- Add relevant topics (e.g., `api`, `python`, `authentication`, `wp4`)
- Topics help with searching, filtering, and organizing repositories
- Use consistent topics across similar repositories

4.5.3.3 Branch Protection Rules

For `main` branch:

- Require pull request reviews (minimum 1 approval)
- Require status checks to pass before merging
- Require conversation resolution before merging
- Include administrators in restrictions
- Restrict force pushes (disabled)
- Restrict deletions (disabled)

- Require linear history (optional, for cleaner history)

4.5.3.4 Security Settings

Enable:

- Dependency graph
- Dependabot alerts
- Dependabot security updates
- Secret scanning
- Code scanning (if applicable)

4.5.4 Repository Maintenance

Regular Maintenance Tasks

Daily/Weekly: - Monitor and triage new issues - Review pull requests within 2 days - Check CI/CD status - Fix failing builds

Monthly: - Review Dependabot PRs - Update dependencies - Check security alerts - Update documentation - Review and clean up stale branches

Quarterly: - Review team access - Remove inactive users - Analyze repository metrics - Plan improvements

4.5.5 Archiving and Deprecation

When to Archive?

- Project completed
- No longer maintained
- Replaced by newer version
- End of project phase
- Historical reference only

Deprecation Process

1. Add deprecation notice to README
2. Create migration guide
3. Update documentation
4. Notify users
5. Set deprecation timeline
6. Plan archival date

Deprecation Notice Template

```
# DEPRECATED

**This repository is deprecated and no longer maintained.**
```

```
**Reason:** [Explain why deprecated]

**Alternative:** Please use [new-repository](link) instead.

**Migration Guide:** See [MIGRATION.md](MIGRATION.md) for migration instructions.

**Support:** Security fixes only until [date]. No new features will be added.
```

4.6 Licensing and Intellectual Property

4.6.1 Default License Policy

Standard License: Apache License 2.0

Current Status in WeBuild: Most WeBuild repositories use Apache License 2.0, including: - `architecture` - `wp4-interop-test-bed` - `wp4-qtsp-group` - `wp4-trust-group` - `.github`

Note: Some repositories may not have license properly configured in GitHub metadata. All new repositories should explicitly include Apache 2.0 license.

Rationale: - Permissive open-source license - Compatible with commercial use - Provides patent protection - Widely recognized and understood - Allows derivative works - Minimal restrictions on use

4.6.2 License Implementation

Every repository MUST include:

1. **LICENSE file** in repository root
2. **License headers** in source files (where applicable)
3. **README notice** stating the license
4. **NOTICE file** for attributions (if needed)

Copyright Notice

```
Copyright 2025 WeBuild Consortium

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

Source File Headers

```
# Copyright 2025 WeBuild Consortium
# SPDX-License-Identifier: Apache-2.0
```

4.6.3 Third-Party Dependencies

Before Adding Dependencies

- Check dependency license
- Verify license compatibility
- Review license obligations
- Check for patent clauses
- Assess security implications
- Document license in project

Acceptable Licenses

- Apache 2.0
- MIT
- BSD 2-Clause, 3-Clause
- ISC
- Python Software Foundation
- Unlicense / Public Domain

Problematic Licenses (avoid)

- GPL 2.0 (incompatible with Apache 2.0)
- AGPL (strong copyleft)
- Proprietary licenses
- Licenses with field-of-use restrictions

Dependency Management Best Practices

- Lock package versions in manifest files
- Align package versions across projects when possible
- Document all dependencies and their licenses
- Regularly review and update dependencies
- Use automated tools to track dependency vulnerabilities

4.6.4 Intellectual Property

Key Points

1. **Background IP:** Each partner retains ownership of pre-existing IP
2. **Foreground IP:** New IP created during project is jointly owned
3. **Access Rights:** Partners have access rights per consortium agreement
4. **Publication:** Open access publication required by EU grant
5. **Exploitation:** Partners may exploit results per agreement

Contributors retain copyright to their contributions, but grant rights to the consortium.

4.7 Security and Compliance

4.7.1 Security Principles

Core Principles

1. **Defense in Depth:** Implement multiple layers of security
2. **Least Privilege:** Grant minimum necessary permissions
3. **Secure by Default:** Security should be the default state
4. **Zero Trust:** Never trust, always verify

Mandatory Requirements

All members MUST:

- Enable two-factor authentication (2FA)
- Use strong, unique passwords
- Protect credentials and secrets
- Report security incidents immediately
- Follow secure coding practices
- Complete security training
- Review code for security issues
- Keep dependencies updated

Prohibited Activities

Never:

- Commit secrets or credentials
- Store sensitive data in repositories
- Share accounts or credentials
- Disable security features
- Ignore security warnings
- Use weak or reused passwords
- Access unauthorized resources
- Bypass security controls

4.7.2 Secrets Management

What Are Secrets?

- API keys and tokens
- Passwords and passphrases
- Private keys and certificates
- Database credentials
- OAuth tokens

- Encryption keys
- Service account credentials

Never Commit Secrets

Secrets should never be committed to version control. Use environment variables and secure secret management tools instead.

Use .gitignore

```
# Secrets and credentials
.env
.env.local
.env.*.local
secrets/
*.key
*.pem
*.p12
credentials.json
config/secrets.yml
```

If Secrets Are Committed

1. **Rotate the secret immediately**
2. **Remove from history** using BFG Repo-Cleaner or git filter-branch
3. **Notify** Technical Coordinator and security team
4. **Document** the incident

Secure Storage:

- **Local development:** Use environment variables and `.env` files (excluded from git)
- **CI/CD:** Use GitHub secrets for workflow automation
- **Production:** Use dedicated secret management services (AWS Secrets Manager, Azure Key Vault, HashiCorp Vault)

GitHub Secrets Best Practices

- Store secrets at appropriate level (repository, environment, or organization)
- Use environment secrets for deployment-specific credentials
- Limit access to secrets through environment protection rules
- Rotate secrets regularly
- Never print secrets in logs or output
- Use secret scanning to detect accidental commits

4.7.3 Code Security

Secure Coding Practices

Input Validation: - Always validate and sanitize input - Use parameterized queries - Validate data type, length, and format - Whitelist allowed values - Encode output properly

Authentication and Authorization: - Use strong password hashing (bcrypt, Argon2) - Implement rate limiting - Use secure session management - Implement proper authorization checks - Use HTTPS for all communications

Error Handling: - Log detailed errors server-side - Return generic messages to users - Don't expose stack traces - Don't reveal system information - Handle errors gracefully

Cryptography: - Use established libraries (cryptography, libsodium) - Use strong algorithms (AES-256, RSA-2048+) - Generate secure random keys - Protect encryption keys - Never implement your own crypto

4.7.4 Dependency Security

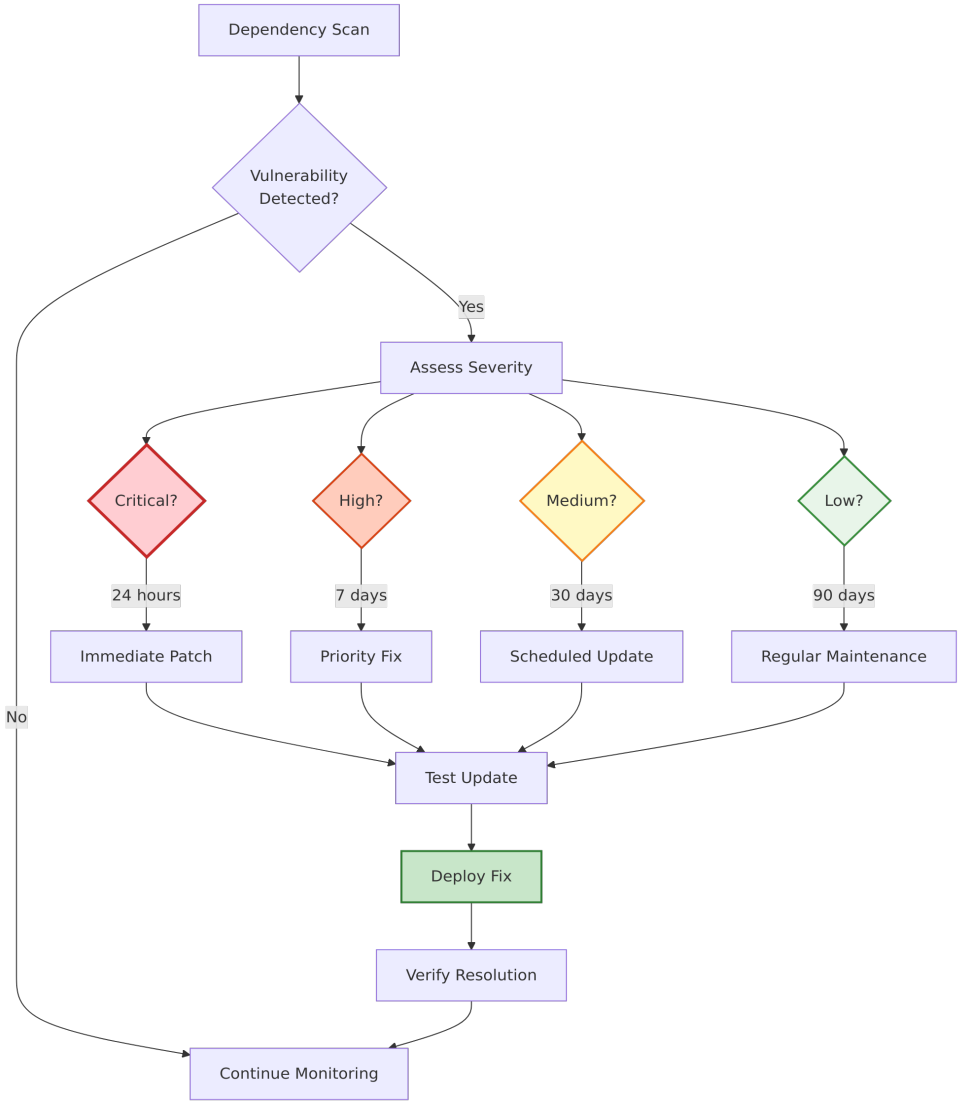

Figure 7: Dependency Security Management and Update Workflow

Dependency Management

Enable GitHub Features:

1. Dependency graph
2. Dependabot alerts
3. Dependabot security updates

Vulnerability Response

Severity	Response Time	Action
Critical	24 hours	Immediate patch
High	7 days	Priority fix
Medium	30 days	Scheduled update
Low	90 days	Regular maintenance

Table 6: Dependency Vulnerability Response Times by Severity Level

Update Strategy

- **Patch versions:** Update automatically (1.2.3 → 1.2.4)
- **Minor versions:** Review and test (1.2.0 → 1.3.0)
- **Major versions:** Careful review and testing (1.0.0 → 2.0.0)

Best Practices

- Don't commit dependencies to source code (use package managers)
- Lock dependency versions to ensure reproducible builds
- Regularly review and update dependencies
- Monitor security advisories
- Test updates in non-production environments first

4.7.5 Data Protection and GDPR Compliance

Never store in repositories

- Names and contact information
- Email addresses
- Phone numbers
- Physical addresses
- IP addresses
- User IDs (if personally identifiable)
- Any other personal data

If personal data needed for testing

- Use synthetic/fake data
- Anonymize real data
- Use data generators
- Obtain explicit consent (if real data required)

Data Classification

Level	Examples	Handling
Public	Published docs, open source code	No restrictions
Internal	Internal docs, non-sensitive code	Consortium only
Confidential	Partner data, unpublished research	Restricted access
Restricted	Credentials, personal data	Never in repos

Table 7: Data Classification Levels and Repository Handling Requirements

4.7.6 Data Protection and GDPR Compliance

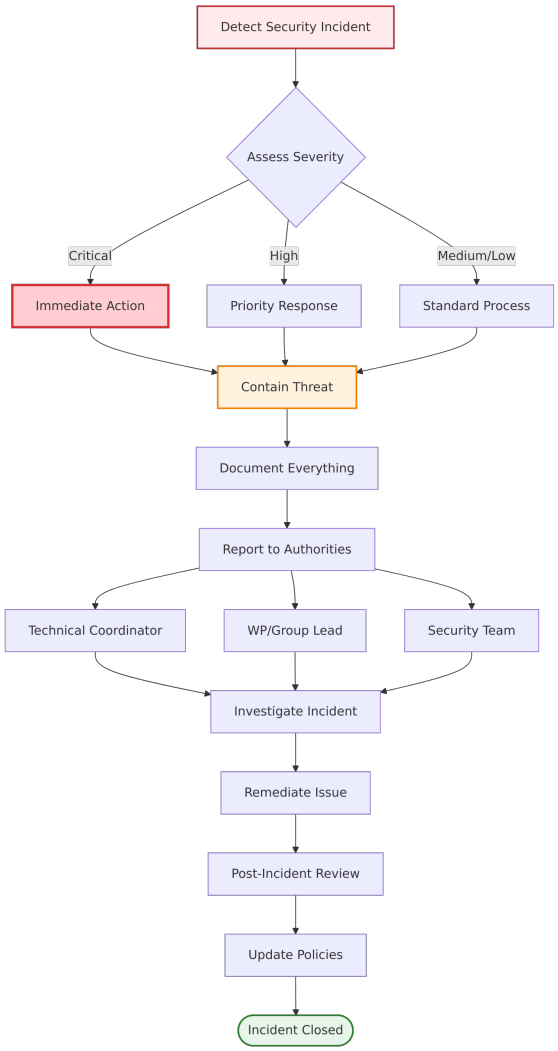

Figure 8: Security Incident Response Process Flow

What is a Security Incident?

- Unauthorized access to repositories
- Compromised credentials
- Data breach or exposure
- Malware or malicious code
- Denial of service attack
- Vulnerability exploitation

Reporting Incidents

Immediate Actions:

1. Don’t panic
2. Document everything
3. Report immediately to:
    - Technical Coordinator
    - WP/Group Lead
    - Security team

Include in report

- What happened
- When it happened
- What was affected
- Current status
- Actions taken
- Evidence collected

4.7.7 Compliance Requirements

EU Regulations

GDPR (General Data Protection Regulation):

- Lawful basis for data processing
- Explicit consent when required
- Data minimization
- Purpose limitation
- Storage limitation
- Appropriate security measures
- Accountability and documentation

Implementation

- Data protection impact assessment

- Privacy by design
- Data processing records
- Data breach procedures
- Data subject rights procedures

4.8 Contribution Guidelines

4.8.1 Getting Started

Prerequisites

Before contributing, ensure you have:

- GitHub account with 2FA enabled
- Access to WeBuild organization
- Appropriate repository permissions
- Git installed and configured
- Development environment set up
- Familiarity with Git workflows

4.8.2 Contribution Workflow

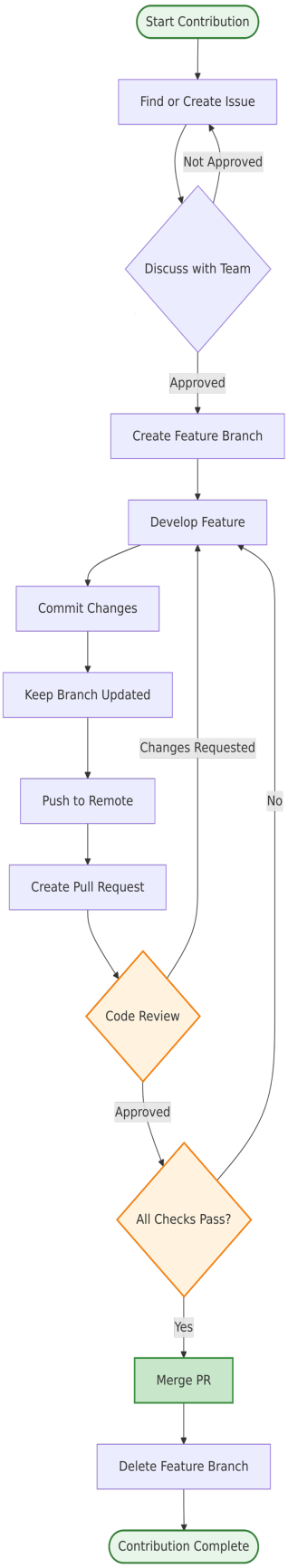

Figure 9: Complete Contribution Workflow from Issue to Merged Code

Step-by-Step Process

4.8.2.1 Step 1: Find or Create an Issue

- Search existing issues for similar work
- If no issue exists, create one describing the problem or feature
- Discuss approach with team
- Get approval from maintainer
- Get issue assigned to you

4.8.2.2 Step 2: Create Feature Branch

```
# Update local main
git checkout main
git pull origin main

# Create feature branch
git checkout -b feature/123-add-user-profile

# Verify branch
git branch
```

4.8.2.3 Step 3: Develop the Feature

```
# Make changes
# ... edit files ...

# Stage changes
git add src/profile.py tests/test_profile.py

# Commit with clear message
git commit -m "feat: add user profile management

- Implement profile CRUD operations
- Add profile validation
- Create profile API endpoints
- Add comprehensive tests

Closes #123"
```

4.8.2.4 Step 4: Keep Branch Updated

```
# Fetch latest changes
git fetch origin

# Rebase on main (preferred)
git rebase origin/main

# Resolve conflicts if any
# ... fix conflicts ...
git add .
git rebase --continue
```

4.8.2.5 Step 5: Push and Create PR

```
# Push feature branch
git push origin feature/123-add-user-profile
```

```
# If rebased, may need force push
git push origin feature/123-add-user-profile --force-with-lease
```

4.8.2.6 Step 6: Address Review Feedback

```
# Make requested changes
# ... edit files ...

# Commit changes
git add .
git commit -m "refactor: address review feedback"

# Push updates
git push origin feature/123-add-user-profile
```

4.8.2.7 Step 7: Merge

After approval

1. Ensure all checks pass
2. Resolve any conflicts
3. Squash commits if needed
4. Merge pull request
5. Delete feature branch

4.8.3 Code Standards

General Principles

- Write clean, readable code
- Follow SOLID principles
- Keep it simple
- Use meaningful variable names
- Add comments for complex logic
- Write self-documenting code where possible

Language-Specific Standards

Follow established style guides for each language:

- **Python:** PEP 8, use pylint, flake8, black
- **JavaScript/TypeScript:** Airbnb style guide, use ESLint, Prettier
- **Java:** Google Java Style Guide, use checkstyle

Security Considerations

Always:

- Validate all inputs
- Sanitize user data
- Use parameterized queries

- Implement proper authentication
- Follow principle of least privilege
- Keep dependencies updated

Never:

- Store passwords in plain text
- Commit secrets or credentials
- Trust user input
- Use deprecated security libraries
- Ignore security warnings

4.8.4 Testing Requirements

All code changes must include tests

- New features require new tests
- Bug fixes require regression tests
- Refactoring maintains test coverage
- Tests must pass before merging

Types of Tests

- **Unit Tests:** Test individual components in isolation
- **Integration Tests:** Test component interactions
- **End-to-End Tests:** Test complete user workflows

Minimum Requirements

- Unit test coverage: 80%
- Critical paths: 100%
- New code: Must include tests
- Bug fixes: Must include regression tests

4.8.5 Code Review Guidelines

For Authors

- Self-review first
- Provide clear PR description
- Link to relevant issues
- Explain design decisions
- Be responsive to feedback
- Make requested changes promptly

For Reviewers

- Review promptly (within 2 days)
- Be respectful and constructive
- Focus on important issues
- Explain reasoning
- Approve when satisfied
- Test if possible

Review Checklist

- Functionality: Does it work as intended?
- Code Quality: Is it well-written and maintainable?
- Tests: Are there adequate tests?
- Documentation: Is it properly documented?
- Security: Are there security concerns?
- Performance: Are there performance issues?
- Standards: Does it follow coding standards?

4.9 Onboarding and Offboarding Procedures

4.9.1 Onboarding Journey

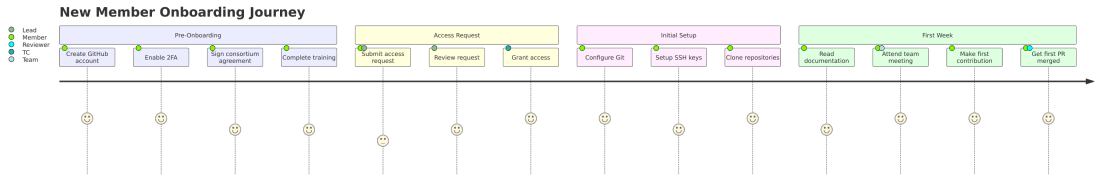

Figure 10: New Member Onboarding Journey and Milestones

4.9.2 Pre-Onboarding Requirements

Before requesting access, new members must:

Have a GitHub Account: - Create account at github.com - Use professional username (preferably real name) - Add profile picture - Enable two-factor authentication (2FA)

Complete Consortium Onboarding: - Sign Consortium Agreement - Receive partner organization approval

Understand Project Context: - Review project overview - Understand WP/Group objectives - Identify role and responsibilities - Know reporting structure

4.9.3 Initial Setup

Git Configuration

```
# Set your name and email
git config --global user.name "Your Full Name"
git config --global user.email "your.email@organization.com"

# Set default branch name
git config --global init.defaultBranch main

# Enable credential caching
git config --global credential.helper cache

# Verify configuration
git config --list
```

SSH Key Setup (Recommended)

```
# Generate SSH key
ssh-keygen -t ed25519 -C "your.email@organization.com"

# Start SSH agent
eval "$(ssh-agent -s)"

# Add key to agent
ssh-add ~/.ssh/id_ed25519

# Copy public key and add to GitHub
cat ~/.ssh/id_ed25519.pub
```

4.9.4 First Week Goals

Week 1 Checklist

- ☐ Complete environment setup
- ☐ Clone relevant repositories
- ☐ Read all documentation
- ☐ Attend team meeting
- ☐ Make first contribution (even if small)
- ☐ Get first PR merged
- ☐ Introduce yourself to team
- ☐ Set up communication channels

4.9.5 Offboarding Procedures

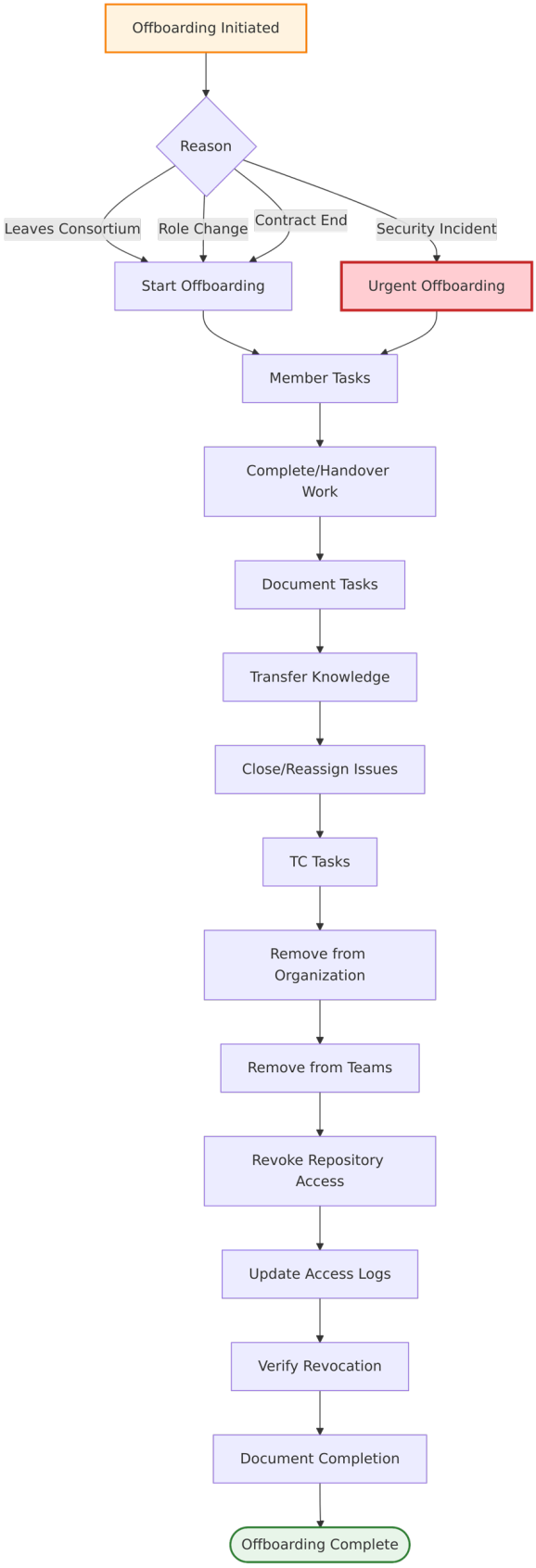

Figure 11: Member Offboarding Procedure and Checklist Flow

When Offboarding is Required

- Member leaves the consortium
- Partner organization exits project
- Contract/assignment ends
- Role changes (no longer needs access)
- Security incident requires access revocation

Offboarding Checklist

For Departing Member:

- [ ] Complete assigned work or hand over
- [ ] Document ongoing tasks
- [ ] Transfer knowledge to team
- [ ] Update documentation
- [ ] Close or reassign issues
- [ ] Finish open pull requests

For Technical Coordinator

- [ ] Remove from organization
- [ ] Remove from all teams
- [ ] Revoke repository access
- [ ] Update access logs
- [ ] Verify access revocation complete
- [ ] Document offboarding completion

4.10 Best Practices Summary

4.10.1 General Best Practices

Do

- Keep branches short-lived (< 2 weeks)
- Update from main frequently
- Write clear commit messages
- Make atomic commits
- Test before pushing
- Review your own code first
- Delete merged branches
- Use branch protection rules
- Follow naming conventions
- Document significant decisions
- Commit frequently to avoid losing work
- Pull before you push to avoid conflicts

Don’t

- Commit directly to main
- Force push to shared branches
- Leave branches unmerged for long
- Mix unrelated changes
- Commit broken code
- Ignore CI failures
- Skip code review
- Rewrite published history
- Commit dependencies or configuration files
- Hardcode secrets or credentials

4.10.2 Security Best Practices

Always

- Enable 2FA on all accounts
- Use strong, unique passwords
- Protect credentials and secrets
- Validate all inputs
- Use parameterized queries
- Keep dependencies updated
- Review code for security issues
- Report incidents immediately
- Rotate secrets regularly
- Use environment variables for configuration

Never

- Commit secrets or credentials
- Store sensitive data in repositorie
- Share accounts or credentials
- Disable security features
- Ignore security warnings
- Trust user input without validation
- Use deprecated security libraries
- Expose secrets in logs

4.10.3 Code Quality Best Practices

Write Clean Code

- Clear and readable
- Self-documenting where possible
- Consistent style
- Properly formatted
- Well-tested
- Adequately documented

Follow SOLID Principles

- Single Responsibility
- Open/Closed
- Liskov Substitution
- Interface Segregation
- Dependency Inversion

Keep It Simple

- Avoid over-engineering
- Use appropriate abstractions
- Prefer clarity over cleverness
- Make code maintainable

4.10.4 Repository Management Best Practices

Organization

- Use clear naming conventions
- Classify repositories with topics
- Maintain comprehensive README files
- Keep repositories focused and scoped appropriately
- Archive dead or unmaintained repositories

Collaboration

- Use pull requests for all changes
- Conduct thorough code reviews
- Leverage issue tracking effectively
- Use GitHub Projects for planning
- Communicate clearly and frequently

4.11 Support and Resources

4.11.1 Getting Help

For GitHub Issues

1. Check repository documentation
2. Search existing issues
3. Open an issue in the relevant repository
4. Contact repository maintainers
5. Escalate to WP/Group Lead

For Policy Questions

1. Review this documentation
2. Contact your WP/Group Lead
3. Reach out to Technical Coordinator

Communication Channels: - GitHub Issues (for technical questions) - Consortium internal channels (for policy/administrative questions) - Note: Additional support channels (e.g., Slack) may be established as needed

4.11.2 Training and Resources

Internal Resources

- Repository documentation
- Wiki pages
- Past pull requests and issues
- Team meeting notes
- Architecture diagrams

External Resources

- GitHub Documentation
- Git Documentation
- Pro Git Book
- Apache License 2.0
- SPDX License List

4.11.3 Contact Information

Technical Coordinator: webuild-github-support@grnet.gr

WP4 Lead: Contact via consortium internal channels

Group Leads: Contact via consortium internal channels or repository discussions

Support: - For technical issues: Open an issue in the relevant repository - For policy questions: Contact your WP/Group Lead - For access requests: Contact Technical Coordinator through consortium channels

4.12 Appendices

4.12.1 Appendix A: Quick Reference Checklist

New Repository Setup

- ☐ Repository created with appropriate name
- ☐ README.md with required content
- ☐ LICENSE file (Apache 2.0)
- ☐ CONTRIBUTING.md
- ☐ .gitignore configured
- ☐ Branch protection rules set
- ☐ Topics added
- ☐ Team access configured
- ☐ EU funding acknowledgment included
- ☐ Security settings enabled

Before Making Repository Public

- ☐ Pre-publication checklist completed
- ☐ No secrets or credentials
- ☐ No sensitive data
- ☐ Appropriate license
- ☐ Documentation complete
- ☐ Approvals obtained
- ☐ Announcement prepared

Pull Request Checklist:

- ☐ Code follows style guidelines
- ☐ All tests pass locally
- ☐ New tests added for new functionality
- ☐ Documentation updated
- ☐ Commit messages are clear
- ☐ No merge conflicts with main
- ☐ Self-review completed
- ☐ No debugging code or console logs

4.12.2 Appendix B: Common Git Commands

Branch Management

```
# Create branch
git checkout -b feature/123-name

# Update branch
git fetch origin
git rebase origin/main
```

```
# Delete branch
git branch -d feature/123-name
```

Commit and Push

```
# Commit
git add .
git commit -m "type: description"

# Push
git push origin feature/123-name

# Force push (use with caution)
git push origin feature/123-name --force-with-lease
```

Sync with Main

```
git checkout main
git pull origin main
git checkout feature/123-name
git rebase main
```

Stash Changes

```
# Save work in progress
git stash save "WIP: feature implementation"

# Apply stash
git stash pop
```

4.12.3 Appendix C: Troubleshooting

Common Issues

Issue: Merge Conflicts

```
# Update your branch
git fetch origin
git rebase origin/main

# Fix conflicts in files
git add .
git rebase --continue
```

Issue: Accidentally Committed to Main

```
# Move commit to new branch
git branch feature/accidental-commit
git reset --hard HEAD~1
git checkout feature/accidental-commit
```

Issue: Need to Change Last Commit Message

```
git commit --amend -m "New commit message"
git push origin branch-name --force-with-lease
```

Issue: Pushed Sensitive Data

1. Rotate the secret immediately

2. Remove from history using BFG Repo-Cleaner
3. Notify Technical Coordinator
4. Document the incident

4.12.4 Appendix D: Glossary

- **Consortium:** WeBuild project partners
- **WP:** Work Package
- **PR:** Pull Request
- **CI/CD:** Continuous Integration/Continuous Deployment
- **GDPR:** General Data Protection Regulation
- **IP:** Intellectual Property
- **EUDI:** European Digital Identity
- **2FA:** Two-Factor Authentication
- **SSH:** Secure Shell
- **API:** Application Programming Interface
- **Atomic Commit:** A commit containing a single logical change
- **Branch Protection:** Rules that prevent direct commits to protected branches
- **Code Owner:** Person or team responsible for reviewing changes to specific code
- **Dependency:** External library or package used by the project
- **Fork:** Personal copy of a repository
- **Merge Conflict:** When Git cannot automatically merge changes
- **Rebase:** Reapplying commits on top of another base
- **Semantic Versioning:** Version numbering scheme (MAJOR.MINOR.PATCH)
- **Tag:** Named reference to a specific commit

4.12.5 Appendix E: EU Funding Acknowledgment

All repositories MUST include EU funding acknowledgment

Required Text

```
## Funding

![Co-funded by the European Union](https://github.com/EWC-consortium/ewc-wiki/assets/455274/1ac9b4e3-06b9-4c3c-a2af-ec5fbf584517)

The WeBuild project is co-funded by the European Union. However, the views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the granting authority. Neither the European Union nor the granting authority can be held responsible.
```

Placement: In README.md and major documentation files