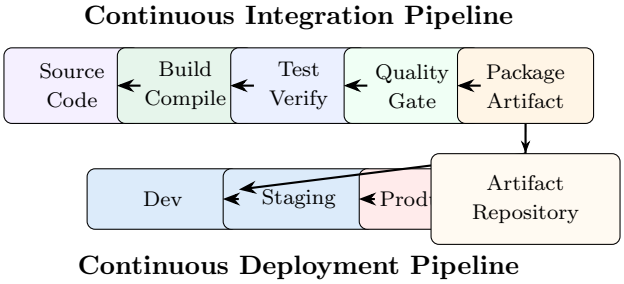
Builder's View

Architecture Viewpoint Specification

CI/CD Pipelines, Build Systems, Artifacts & Quality Gates



Version:	2.0
Status:	Release
Classification:	ISO/IEC/IEEE 42010 Compliant
Last Updated:	December 12, 2025

Based on the Views and Beyond approach to software architecture documentation

Contents

1 Viewpoint Name

Viewpoint Identification

Name:	Builder's View
Synonyms:	Build View, CI/CD View, Pipeline View, Integration View, Automation View, DevOps View
Identifier:	VP-BLD-001
Version:	2.0

1.1 Viewpoint Classification

The Builder's View addresses the concerns of build engineers, DevOps practitioners, and release managers who need to understand how source code is transformed into deployable artifacts. This viewpoint focuses on automation, continuous integration, continuous delivery, and the quality gates that ensure software readiness.

Table 1: Viewpoint Classification Taxonomy

Attribute	Value
Style Family	Allocation (Build and CI/CD)
Primary Focus	Build Processes, Pipelines, and Automation
Abstraction Level	Process / Automation
Temporal Perspective	Build-time and Integration-time
Related Styles	Development View, Deployment View
IEEE 42010 Category	Development Viewpoint (build aspects)
DevOps Alignment	Continuous Integration, Continuous Delivery

1.2 Viewpoint Scope

The Builder's View encompasses the following aspects:

- **Build Systems:** Tools and configurations for compiling and packaging code.
- **CI/CD Pipelines:** Automated workflows for integration and delivery.
- **Quality Gates:** Automated checks ensuring code quality and standards.
- **Artifact Management:** Storage, versioning, and distribution of build outputs.
- **Dependency Management:** External library and package handling.
- **Environment Configuration:** Build and test environment specifications.
- **Release Management:** Versioning, tagging, and release processes.

- **Build Performance:** Optimization and efficiency of build processes.

2 Overview

The Builder's View provides DevOps engineers and developers with comprehensive documentation of how software is built, tested, and prepared for deployment. It ensures consistent, repeatable, and automated software delivery.

2.1 Purpose and Scope

The primary purpose of this viewpoint is to document the complete build and integration process, ensuring that anyone can understand, maintain, and improve the CI/CD infrastructure.

Viewpoint Definition

The Builder's View documents the build systems, CI/CD pipelines, quality gates, artifact management, dependency handling, and release processes that transform source code into deployable software. It provides the knowledge needed to maintain reliable, efficient, and automated software delivery infrastructure.

2.2 Key Characteristics

The Builder's View exhibits several distinctive characteristics:

Automation-Centric: Emphasizes automated, repeatable processes over manual steps.

Pipeline-Oriented: Organizes build activities as sequential or parallel pipeline stages.

Quality-Focused: Integrates quality gates and automated verification throughout.

Reproducible: Ensures builds are deterministic and reproducible.

Observable: Provides visibility into build status, metrics, and history.

2.3 Relationship to Other Viewpoints

The Builder's View connects to other architectural viewpoints:

Table 2: Relationships to Other Viewpoints

Viewpoint	Relationship
Development	Build processes compile module structure. Dependencies defined in development are resolved during build.
Deployment	Build produces deployment artifacts. Pipelines trigger deployment processes.
Operational	Build metrics feed operational dashboards. Build failures trigger alerts.
Designer	Code standards enforced by build quality gates. Patterns verified by static analysis.
Information	Data migrations may be part of release pipeline. Schema changes versioned with artifacts.
Security	Security scanning integrated in pipeline. Secrets management for build processes.

2.4 Build Pipeline Overview

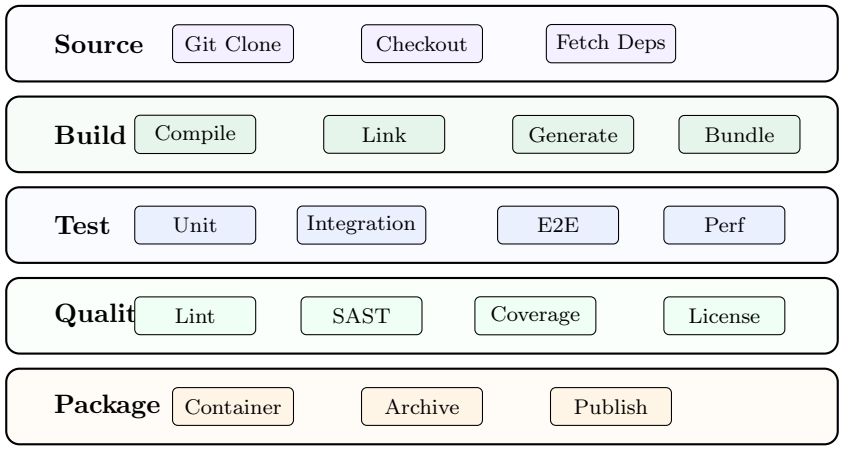


Figure 1: CI Pipeline Stage Overview

3 Concerns

This section enumerates the build and integration concerns that the Builder's View addresses.

3.1 Primary Concerns

C1: Build System Configuration

- What build tools are used?

- How is the build configured?
- What are the build targets?
- How are build variants managed?
- What build optimizations exist?

C2: CI/CD Pipeline Design

- What pipeline stages exist?
- How do stages relate and depend?
- What triggers pipeline execution?
- What parallel execution exists?
- How are failures handled?

C3: Quality Gate Definition

- What quality checks are automated?
- What thresholds must be met?
- What happens when gates fail?
- How are gate results reported?
- How are exceptions handled?

C4: Artifact Management

- What artifacts are produced?
- Where are artifacts stored?
- How are artifacts versioned?
- What retention policies exist?
- How are artifacts promoted?

C5: Dependency Management

- What dependencies are required?
- How are versions specified?
- How is dependency resolution handled?
- What security scanning exists?
- How are updates managed?

C6: Environment Configuration

- What build environments exist?
- How are environments provisioned?
- What environment variables are needed?
- How is configuration managed?
- How is environment parity ensured?

C7: Release Management

- How are releases versioned?

- What is the release branching strategy?
- How are release notes generated?
- What approval processes exist?
- How are hotfixes handled?

C8: Build Performance

- What is the current build time?
- What caching strategies exist?
- How is incremental building done?
- What parallelization exists?
- What are performance targets?

C9: Security and Secrets

- How are secrets managed?
- What security scanning is done?
- How is access controlled?
- What audit trails exist?
- How are vulnerabilities handled?

C10: Build Observability

- What metrics are collected?
- How is build status communicated?
- What dashboards exist?
- How are trends analyzed?
- What alerting exists?

3.2 Concern-Quality Attribute Mapping

Table 3: Concern to Quality Attribute Mapping

Concern	Reliabil.	Effictic.	Security	Maintain.	Traceab.	Reproduc.	Scalabil.	Automat.
Build System	●	●	○	●	○	●	○	●
Pipelines	●	●	○	●	○	○	●	●
Quality Gates	●	○	●	●	●	–	–	●
Artifacts	○	○	○	○	●	●	○	○
Dependencies	●	○	●	●	●	●	–	○
Environment	●	○	○	○	○	●	●	○
Release Mgmt	○	○	○	○	●	○	–	○
Performance	–	●	–	○	–	–	●	○
Security	○	–	●	○	●	–	–	●
Observability	●	○	○	●	●	–	○	●

● = Primary impact, ○ = Secondary impact, – = Minimal impact

4 Anti-Concerns

Understanding what the Builder's View is *not* appropriate for helps stakeholders avoid misapplying this viewpoint.

4.1 Out of Scope Topics

AC1: Runtime System Behavior

- Application logic and behavior
- Runtime performance characteristics
- User interactions
- Business process flows
- Runtime error handling

AC2: Production Operations

- Production monitoring
- Incident management
- Capacity planning
- Production troubleshooting
- SLA management

AC3: Code Design Details

- Design patterns implementation

- Algorithm specifics
- API design
- Data structures
- Business logic

AC4: Infrastructure Management

- Production infrastructure
- Network configuration
- Storage management
- Disaster recovery
- Infrastructure scaling

AC5: Business Requirements

- Feature specifications
- User stories
- Business rules
- Acceptance criteria
- Product roadmap

Common Misapplications

Avoid using the Builder's View for:

- Runtime behavior documentation (use C&C View)
- Production operations (use Operational View)
- Code structure (use Development View)
- Infrastructure details (use Deployment View)
- Design decisions (use Designer's View)

5 Typical Stakeholders

The Builder's View serves stakeholders involved in software build and delivery automation.

5.1 Primary Stakeholders

Table 4: Primary Stakeholder Analysis

Stakeholder	Role Description	Primary Interests
DevOps Engineers	Build and maintain CI/CD	Pipeline design, automation, reliability
Build Engineers	Manage build systems	Build optimization, tooling, caching
Release Managers	Coordinate releases	Version control, release process, artifacts
Platform Engineers	Provide build platforms	Environment provisioning, scaling, tools
Security Engineers	Secure the pipeline	Security scanning, secrets, compliance
QA Engineers	Define quality gates	Test automation, coverage, gate criteria

5.2 Secondary Stakeholders

Table 5: Secondary Stakeholder Analysis

Stakeholder	Role Description	Primary Interests
Developers	Write and commit code	Build feedback, local builds, debugging
Tech Leads	Guide technical decisions	Build standards, tooling choices
Architects	Define system structure	Build architecture, dependencies
Product Managers	Plan releases	Release timing, version management
Operations Team	Receive deployments	Artifact quality, deployment readiness
Compliance Officers	Ensure compliance	Audit trails, security compliance

5.3 Stakeholder Concern Matrix

Table 6: Stakeholder-Concern Responsibility Matrix

	Build Sys	Pipelines	Quality	Artifacts	Deps	Environ.	Release	Perform.	Security	Observ.
DevOps Eng	R	R	C	R	C	R	R	R	C	R
Build Eng	R	C	I	C	R	C	I	R	I	C
Release Mgr	I	C	C	R	I	I	R	I	C	C
Platform Eng	C	C	I	C	C	R	I	C	C	R
Security Eng	I	C	R	C	R	C	C	I	R	C
QA Engineer	I	C	R	I	I	I	C	I	I	C

R = Responsible, A = Accountable, C = Consulted, I = Informed

6 Model Types

The Builder's View employs several complementary model types to capture build and integration knowledge.

6.1 Model Type Catalog

MT1: Pipeline Definition

- *Purpose:* Document CI/CD pipeline structure
- *Primary Elements:* Stages, jobs, steps, triggers
- *Key Relationships:* Depends-on, triggers
- *Typical Notation:* YAML, pipeline diagrams

MT2: Build Configuration

- *Purpose:* Specify build system settings
- *Primary Elements:* Targets, dependencies, options
- *Key Relationships:* Depends-on, produces
- *Typical Notation:* Build files (Makefile, Gradle, etc.)

MT3: Artifact Manifest

- *Purpose:* Document build outputs
- *Primary Elements:* Artifacts, versions, locations
- *Key Relationships:* Produces, stored-in
- *Typical Notation:* Manifest files, tables

MT4: Dependency Graph

- *Purpose:* Show dependency relationships
- *Primary Elements:* Packages, versions, relationships

- *Key Relationships:* Depends-on, conflicts-with
- *Typical Notation:* Dependency diagrams, lock files

MT5: Quality Gate Specification

- *Purpose:* Define quality criteria
- *Primary Elements:* Checks, thresholds, actions
- *Key Relationships:* Requires, blocks
- *Typical Notation:* Gate definitions, tables

MT6: Environment Specification

- *Purpose:* Define build environments
- *Primary Elements:* Containers, tools, configurations
- *Key Relationships:* Runs-on, requires
- *Typical Notation:* Dockerfiles, environment configs

MT7: Release Plan

- *Purpose:* Document release process
- *Primary Elements:* Versions, branches, approvals
- *Key Relationships:* Promotes-to, requires
- *Typical Notation:* Release workflows, diagrams

MT8: Build Metrics Dashboard

- *Purpose:* Track build health and performance
- *Primary Elements:* Metrics, trends, alerts
- *Key Relationships:* Measures, indicates
- *Typical Notation:* Dashboards, charts

MT9: Security Scan Report

- *Purpose:* Document security findings
- *Primary Elements:* Vulnerabilities, severities, remediation
- *Key Relationships:* Affects, remediated-by
- *Typical Notation:* Security reports, tables

6.2 Model Type Relationships

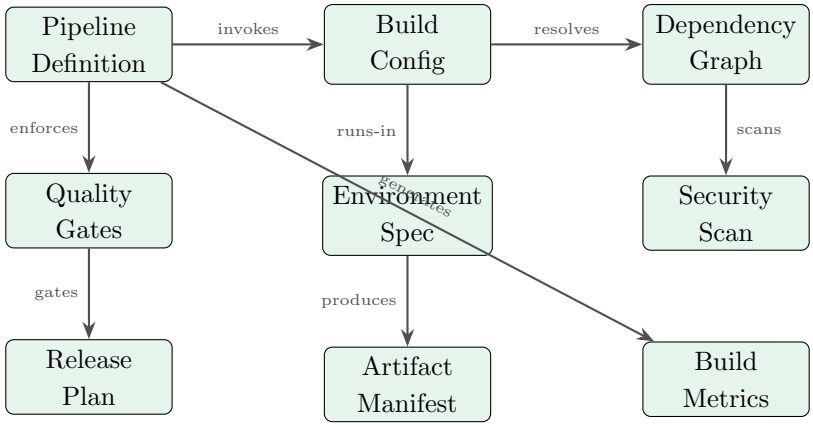


Figure 2: Model Type Dependency Relationships

7 Model Languages

For each model type, specific languages, notations, and techniques are prescribed.

7.1 Pipeline Element Notation

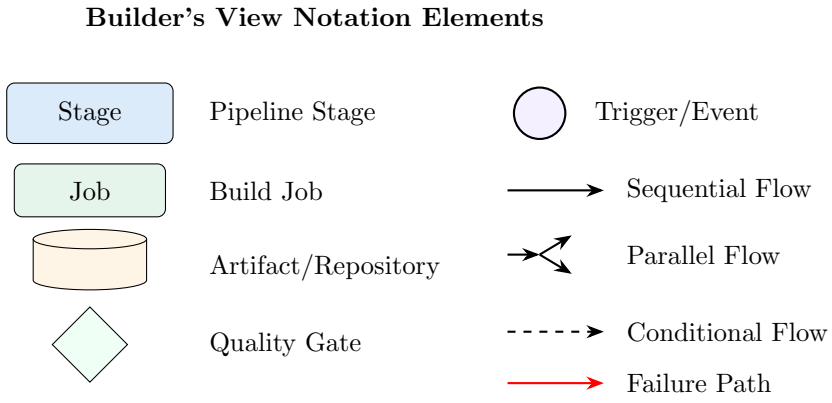


Figure 3: Builder's View Notation Legend

7.2 Pipeline Stage Types

Table 7: Pipeline Stage Classification

Stage Type	Purpose	Typical Activities
Source	Obtain source code	Git clone, checkout, submodules
Build	Compile and link	Compilation, linking, code generation
Test	Verify correctness	Unit tests, integration tests, E2E tests
Quality	Enforce standards	Linting, SAST, coverage, complexity
Security	Scan for vulnerabilities	DAST, dependency scan, secrets scan
Package	Create artifacts	Container build, archive, signing
Publish	Store artifacts	Push to registry, artifact upload
Deploy	Release to environment	Deployment to dev/staging/prod

7.3 Quality Gate Types

Table 8: Quality Gate Categories

Gate Type	What It Checks	Typical Threshold	Tools
Code Coverage	Test coverage percentage	$\geq 80\%$	JaCoCo, Istanbul
Static Analysis	Code quality issues	0 critical issues	SonarQube, ESLint
Security Scan	Known vulnerabilities	0 high/critical CVEs	Snyk, Trivy
License Check	License compliance	Approved licenses only	FOSSA, Black Duck
Performance	Performance regression	$< 10\%$ degradation	JMeter, k6
Complexity	Code complexity metrics	Cyclomatic < 10	SonarQube

7.4 Artifact Types

Table 9: Build Artifact Classification

Artifact Type	Description	Storage	Examples
Container Image	Docker/OCI images	Container registry	Docker Hub, ECR
Package	Language packages	Package registry	npm, Maven, PyPI
Binary	Compiled executables	Artifact storage	S3, Artifactory
Archive	Compressed bundles	Artifact storage	tar.gz, zip
Documentation	Generated docs	Static hosting	GitHub Pages
Test Results	Test reports	CI storage	JUnit XML, HTML

7.5 Pipeline Configuration Languages

Table 10: CI/CD Platform Configuration

Platform	Config Format	Key Features
GitHub Actions	YAML	Workflows, jobs, steps, matrix builds, reusable workflows
GitLab CI	YAML	Stages, jobs, rules, includes, DAG pipelines
Jenkins	Groovy/YAML	Declarative/scripted, shared libraries, plugins
Azure Pipelines	YAML	Stages, jobs, templates, environments, approvals
CircleCI	YAML	Workflows, orbs, executors, caching
Tekton	YAML/K8s	Tasks, pipelines, triggers, Kubernetes-native

8 Viewpoint Metamodels

This section defines the conceptual metamodel underlying the Builder's View.

8.1 Core Metamodel

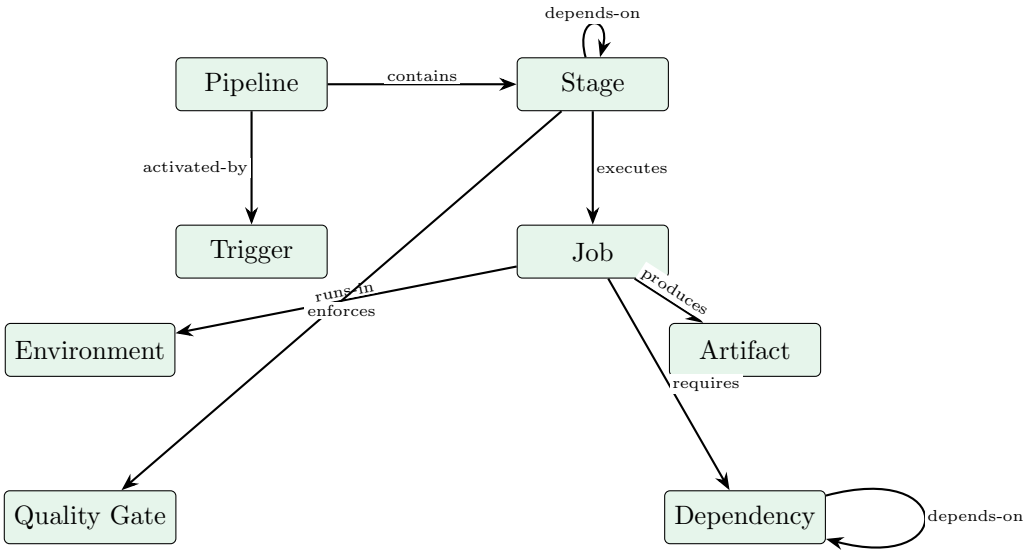


Figure 4: Builder's View Core Metamodel

8.2 Entity Definitions

Entity: Pipeline

Definition: An automated workflow that orchestrates the build, test, and delivery process.

Attributes:

- `pipelineId`: Unique identifier
- `name`: Pipeline name
- `description`: Pipeline purpose
- `triggers`: Events that start the pipeline
- `stages`: Ordered list of stages
- `timeout`: Maximum execution time
- `concurrency`: Parallel execution settings
- `variables`: Pipeline-level variables
- `caching`: Cache configuration
- `notifications`: Alert settings

Constraints:

- Pipeline should have at least one trigger
- Pipeline should have at least one stage
- Timeout should be reasonable

Entity: Stage

Definition: A logical grouping of jobs that execute together in the pipeline.

Attributes:

- `stageId`: Unique identifier
- `name`: Stage name
- `type`: Stage type (build, test, deploy, etc.)
- `jobs`: Jobs in this stage
- `dependencies`: Required previous stages
- `condition`: Execution condition
- `environment`: Target environment
- `approvals`: Required approvals
- `timeout`: Stage timeout
- `retryPolicy`: Retry configuration

Constraints:

- Dependencies must not create cycles
- Stages should have clear purposes
- Conditions should be deterministic

Entity: Job

Definition: A unit of work executed on a build agent, consisting of steps.

Attributes:

- `jobId`: Unique identifier
- `name`: Job name
- `steps`: Ordered list of steps
- `environment`: Execution environment
- `agent`: Build agent requirements
- `matrix`: Matrix build parameters
- `services`: Required services (databases, etc.)
- `artifacts`: Artifacts produced/consumed
- `cache`: Cache configuration
- `timeout`: Job timeout

Constraints:

- Jobs should be atomic and independent where possible
- Environment requirements should be explicit
- Timeouts should prevent hung jobs

Entity: Trigger

Definition: An event or condition that initiates pipeline execution.

Attributes:

- `triggerId`: Unique identifier
- `type`: Trigger type (push, PR, schedule, manual, API)
- `source`: Event source (branch, tag, webhook)
- `filters`: Inclusion/exclusion patterns
- `schedule`: Cron expression for scheduled triggers
- `conditions`: Additional conditions
- `parameters`: Input parameters
- `enabled`: Whether trigger is active

Constraints:

- Triggers should be specific enough to avoid unnecessary builds
- Schedule expressions should be valid cron syntax
- Filters should cover intended scope

Entity: Artifact

Definition: A build output that is stored, versioned, and potentially deployed.

Attributes:

- `artifactId`: Unique identifier
- `name`: Artifact name
- `type`: Artifact type (container, package, binary, etc.)
- `version`: Version identifier
- `checksum`: Integrity hash
- `location`: Storage location
- `metadata`: Additional metadata
- `signature`: Digital signature
- `retention`: Retention policy
- `scanResults`: Security scan results

Constraints:

- Artifacts should be immutable once published
- Versions should follow semantic versioning
- Checksums should be verified on retrieval

Entity: Environment

Definition: The execution context where build jobs run, including tools and configuration.

Attributes:

- `environmentId`: Unique identifier
- `name`: Environment name
- `type`: Environment type (container, VM, bare metal)
- `image`: Container/VM image reference
- `tools`: Installed tools and versions
- `variables`: Environment variables
- `secrets`: Secret references
- `resources`: CPU, memory, storage
- `network`: Network configuration
- `caching`: Cache mounts

Constraints:

- Environments should be reproducible
- Secrets should never be logged
- Resources should be appropriate for workload

Entity: Quality Gate

Definition: An automated check that must pass before proceeding in the pipeline.

Attributes:

- `gateId`: Unique identifier
- `name`: Gate name
- `type`: Gate type (coverage, security, etc.)
- `metric`: What is measured
- `threshold`: Required value
- `operator`: Comparison operator
- `action`: What happens on failure
- `exceptions`: Allowed exceptions
- `evidence`: Documentation of results
- `trend`: Historical trend tracking

Constraints:

- Gates should be objective and measurable
- Thresholds should be achievable
- Exceptions should be documented and temporary

Entity: Dependency

Definition: An external package or library required by the build.

Attributes:

- `dependencyId`: Unique identifier
- `name`: Package name
- `version`: Version constraint
- `resolvedVersion`: Actual resolved version
- `source`: Package registry
- `scope`: Dependency scope (compile, test, runtime)
- `transitive`: Whether transitive or direct
- `license`: License information
- `vulnerabilities`: Known vulnerabilities
- `checksum`: Integrity verification

Constraints:

- Versions should be pinned or use lock files
- Licenses should be reviewed
- Vulnerabilities should be assessed

8.3 Relationship Definitions

Table 11: Metamodel Relationship Definitions

Relationship	Source	Target	Description
contains	Pipeline	Stage	Pipeline includes stages
activated-by	Pipeline	Trigger	Trigger starts pipeline
executes	Stage	Job	Stage runs jobs
runs-in	Job	Environment	Job uses environment
produces	Job	Artifact	Job creates artifact
enforces	Stage	Gate	Stage checks gate
requires	Job	Dependency	Job needs dependency
depends-on	Stage	Stage	Stage ordering
depends-on	Dependency	Dependency	Transitive dependencies

9 Model Correspondence Rules

Model correspondence rules define how build elements relate to elements in other views.

9.1 Development View Correspondence

Correspondence Rule CR-01: Module to Build Target Mapping

Rule: Each deployable module should have a corresponding build target.

Formal Expression:

$$\forall m \in DeployableModules : \exists t \in BuildTargets : builds(t, m)$$

Rationale: Ensures all modules can be built.

Verification: Build target coverage analysis.

9.2 Deployment View Correspondence

Correspondence Rule CR-02: Artifact to Deployment Mapping

Rule: Each deployment target should receive a build artifact.

Formal Expression:

$$\forall d \in DeploymentTargets : \exists a \in Artifacts : deploys(a, d)$$

Rationale: Ensures deployment receives build outputs.

Verification: Deployment artifact traceability.

9.3 Designer's View Correspondence

Correspondence Rule CR-03: Quality Gate to Standard Mapping

Rule: Design standards should be enforced by quality gates.

Formal Expression:

$$\forall s \in AutomatableStandards : \exists g \in QualityGates : enforces(g, s)$$

Rationale: Automates standards compliance.

Verification: Standards automation coverage.

10 Operations on Views

This section defines methods for creating, analyzing, and maintaining build views.

10.1 Creation Methods

10.1.1 View Development Process

Step 1: Define Build Targets

1. Identify all deployable components
2. Define build outputs for each
3. Specify target platforms
4. Document build variants
5. Configure build tools

Step 2: Design Pipeline Structure

1. Identify pipeline stages
2. Define stage dependencies
3. Specify parallel execution opportunities
4. Configure triggers
5. Set up notifications

Step 3: Configure Quality Gates

1. Identify quality metrics
2. Set threshold values
3. Configure gate actions
4. Document exception process
5. Set up trend tracking

Step 4: Set Up Artifact Management

1. Define artifact types
2. Configure storage locations
3. Set versioning scheme
4. Define retention policies
5. Configure signing/verification

Step 5: Configure Environments

1. Define build environments
2. Specify required tools
3. Configure secrets management
4. Set up caching
5. Document environment requirements

Step 6: Implement Security

1. Configure dependency scanning
2. Set up SAST/DAST
3. Implement secrets management
4. Configure access controls
5. Set up audit logging

Step 7: Establish Monitoring

1. Define key metrics
2. Set up dashboards
3. Configure alerts
4. Plan trend analysis
5. Document SLOs

10.1.2 Pipeline Design Patterns

Pattern: Trunk-Based Pipeline

Context: Continuous integration with short-lived branches.

Structure:

- Single main branch (trunk)
- Feature flags for incomplete features
- Automated deployment on merge
- Fast feedback loops

Benefits: Reduced merge conflicts, faster integration, simpler branching.

Pattern: Environment Promotion

Context: Progressive deployment through environments.

Structure:

- Dev → Staging → Production
- Same artifact promoted through stages
- Automated gates between environments
- Manual approval for production

Benefits: Confidence building, risk reduction, audit trail.

Pattern: Matrix Build

Context: Testing across multiple configurations.

Structure:

- Define configuration matrix (OS, version, etc.)
- Parallel execution of combinations
- Aggregate results
- Fail fast on critical failures

Benefits: Comprehensive testing, efficient execution, early detection.

10.2 Analysis Methods

10.2.1 Build Performance Analysis

Build Time Optimization

Purpose: Identify and eliminate build time bottlenecks.

Metrics:

- Total pipeline duration
- Individual stage/job times
- Queue wait time
- Cache hit rates
- Parallelization efficiency

Techniques:

- Identify critical path
- Measure cache effectiveness
- Analyze parallelization opportunities
- Profile slow steps
- Optimize dependency resolution

11 Examples

11.1 Example 1: CI/CD Pipeline Diagram

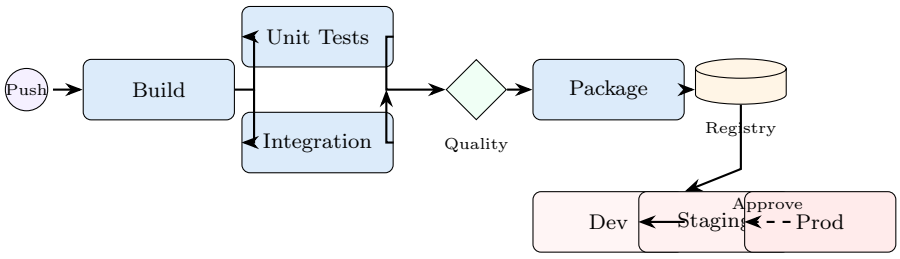


Figure 5: Complete CI/CD Pipeline

Description: This diagram shows a complete CI/CD pipeline. A push event triggers the build stage. Unit and integration tests run in parallel. A quality gate checks coverage and security. On success, artifacts are packaged and published to the registry. Deployment proceeds through Dev and Staging automatically, with manual approval required for Production.

11.2 Example 2: GitHub Actions Workflow

```yaml
name: CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '20'
          cache: 'npm'
      - run: npm ci
      - run: npm run build
      - uses: actions/upload-artifact@v4
        with:
          name: build-output
          path: dist/

  test:
    needs: build
    runs-on: ubuntu-latest
    strategy:
      matrix:
        test-type: [unit, integration]
    steps:
      - uses: actions/checkout@v4
      - run: npm ci
      - run: npm run test:${{ matrix.test-type }}

  quality:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
```

```
      - run: npm ci
      - run: npm run lint
      - run: npm run test:coverage
      - uses: SonarSource/sonarcloud-github-action@master
```

Listing 1: Example GitHub Actions Workflow

11.3 Example 3: Quality Gate Configuration

Table 12: Quality Gate Definition

Gate	Metric	Threshold	Current	Action
Coverage	Line coverage	$\geq 80\%$	85%	Block merge
Security	Critical CVEs	0	0	Block deploy
Complexity	Cyclomatic	< 15	12	Warning
Duplication	Duplicate code	$< 5\%$	4.2%	Warning
Debt	Tech debt ratio	$< 5\%$	3.1%	Informational

12 Notes

12.1 Pipeline Best Practices

CI/CD Best Practices

- **Fast Feedback:** Optimize for quick build times; developers need rapid feedback
- **Fail Fast:** Run quick checks first; don't waste time on obvious failures
- **Reproducibility:** Builds should be deterministic and reproducible
- **Immutable Artifacts:** Never modify artifacts after publishing
- **Version Everything:** Code, config, and infrastructure as code
- **Automate Everything:** Manual steps are error-prone and slow
- **Monitor and Alert:** Track build health and alert on degradation

12.2 Artifact Management Guidelines

Artifact Best Practices

- **Semantic Versioning:** Use SemVer for meaningful version numbers
- **Integrity Verification:** Sign and verify artifact checksums
- **Retention Policies:** Define how long artifacts are kept
- **Promotion Model:** Same artifact moves through environments
- **Metadata:** Include build info, commit hash, timestamp
- **Cleanup:** Automatically clean old/unused artifacts

12.3 Common Pitfalls

Build Anti-Patterns to Avoid

1. **Snowflake Builds:** Unreproducible builds that work only sometimes
2. **Slow Pipelines:** Long build times that discourage frequent commits
3. **Flaky Tests:** Tests that fail intermittently, eroding trust
4. **Secret Leakage:** Secrets appearing in logs or artifacts
5. **Missing Gates:** Skipping quality checks to meet deadlines
6. **Manual Steps:** Human intervention required in pipeline
7. **Environment Drift:** Build environments differing from production
8. **Unversioned Configs:** Pipeline configs not in version control

13 Sources

13.1 Primary References

1. Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley.
2. Kim, G., et al. (2016). *The DevOps Handbook*. IT Revolution Press.
3. Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps*. IT Revolution Press.
4. Richardson, C. (2018). *Microservices Patterns*. Manning Publications.
5. Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A Software Architect's Perspective*. Addison-Wesley.

13.2 Supplementary References

6. Morris, K. (2016). *Infrastructure as Code*. O'Reilly Media.
7. Davis, J., & Daniels, R. (2016). *Effective DevOps*. O'Reilly Media.
8. Nygard, M. (2018). *Release It!* (2nd ed.). Pragmatic Bookshelf.
9. Beyer, B., et al. (2016). *Site Reliability Engineering*. O'Reilly Media.
10. Martin, R. C. (2008). *Clean Code*. Prentice Hall.

13.3 Online Resources

- GitHub Actions Documentation: `https://docs.github.com/actions`
- GitLab CI/CD Documentation: `https://docs.gitlab.com/ee/ci/`
- Jenkins Documentation: `https://www.jenkins.io/doc/`

- DORA Metrics: https://dora.dev/

A Builder's View Checklist

Item	Complete?
Build System	
Build targets defined for all components	☐
Build configuration in version control	☐
Local build possible	☐
Build variants documented	☐
Pipeline	
Pipeline stages defined	☐
Triggers configured	☐
Parallel execution optimized	☐
Failure handling defined	☐
Notifications configured	☐
Quality	
Quality gates defined	☐
Thresholds set appropriately	☐
Security scanning enabled	☐
License compliance checked	☐
Artifacts	
Artifact types documented	☐
Versioning scheme defined	☐
Storage configured	☐
Retention policies set	☐
Operations	
Build metrics collected	☐
Dashboards created	☐
Alerts configured	☐
Documentation complete	☐

B Glossary

Artifact A build output such as a binary, container image, or package.

CI Continuous Integration; automatically building and testing code changes.

CD Continuous Delivery/Deployment; automatically releasing software.

DAST Dynamic Application Security Testing; testing running applications.

Job A unit of work in a pipeline, executed on a build agent.

Pipeline An automated workflow for build, test, and deployment.

Quality Gate	An automated check that must pass to proceed.
SAST	Static Application Security Testing; analyzing source code.
SemVer	Semantic Versioning; a versioning scheme (MAJOR.MINOR.PATCH).
Stage	A logical grouping of jobs in a pipeline.
Trigger	An event that initiates pipeline execution.