15 C4 Model Diagrams

2025-10-19

Contents

1 C4 Model Diagrams

1.1 Vision-Based Pick and Place Robotic System

Document Version: 1.0 **Last Updated:** 2025-10-18 **Status:** Complete

---

1.2 Table of Contents

1.3 1. Introduction

1.3.1 1.1 Purpose

This document presents the system architecture of the Vision-Based Pick and Place Robotic System using the **C4 model** (Context, Containers, Components, Code). The C4 model provides a hierarchical way to visualize software architecture at different levels of abstraction.

1.3.2 1.2 C4 Model Benefits

- **Hierarchical abstraction:** Zoom in/out from system context to code details
- **Audience-appropriate:** Different stakeholders focus on different levels
- **Communication:** Clear, unambiguous diagrams for technical and non-technical audiences
- **Documentation:** Living documentation that evolves with the system

1.3.3 1.3 Notation Legend

```
Person/System       ← External entity (users, external systems)
[Type]
```

```
    Container           ← Application/data store
    [Technology]



    Component           ← Logical grouping of functionality
    [Details]


   >  Relationship (synchronous)
- - - >  Relationship (asynchronous)
```

1.4 2. C4 Model Overview

1.4.1 2.1 The Four C's

Level	Name	Audience	Abstraction	Purpose
C1	**Context**	All stakeholders	Highest	System scope and external dependencies
C2	**Container**	Technical leaders, architects	High	Runtime applications and data stores
C3	**Component**	Developers, architects	Medium	Logical components within containers
C4	**Code**	Developers	Lowest	Classes, interfaces, data structures

1.4.2 2.2 Our C4 Documentation Structure

```
C1: System Context
     > Robot System interacts with Operator, Manager, Engineer, MES, ERP

C2: Containers
     > Vision Pipeline (Docker container)
     > Motion Planning (Docker container)
     > Control System (Docker container)
     > Orchestrator (Docker container)
     > Web Backend (FastAPI container)
     > Web Frontend (React container)
     > Database (PostgreSQL container)
     > Monitoring (Grafana/Prometheus containers)
     > Message Bus (ROS2 DDS)
```

```
C3: Components (per container)
   > Vision Pipeline: Camera Driver, Object Detector, Pose Estimator
   > Motion Planning: MoveIt2 Planner, Collision Checker, Trajectory Generator
   > Control: ros2_control Manager, Joint Controllers, Gripper Controller
   > Orchestrator: Task Manager, Behavior Tree, State Machine

C4: Code (selected critical components)
   > YoloDetector class
   > GraspSynthesizer class
   > PickPlaceServer class
```

1.5 3. Level 1: System Context Diagram

1.5.1 3.1 Context Diagram

```
                                   Operator
                                   [Person]
                                 Runs system,
                                 monitors status

                                          Uses web UI
                                          monitors



   Manager
   [Person]    Reviews       Vision Pick & Place
 Views KPIs,     metrics        Robot System
 ROI                            [Software System]

                                Automated pick-and-place
                        using AI vision
   Engineer    Develops     30 picks/min, 99% accuracy
   [Person]      & debugs
 Maintains,
 optimizes
                               Controls
                                      reads sensors


                                 UR5e Robot
                                 [Hardware]
                                6-DOF arm,
                                gripper, camera
   MES
   [External      Sends status
   System]          pick data
 Manufacturing
 Execution



   ERP
   [External      Sends metrics
   System]          inventory
 Enterprise
 Resource
 Planning



   Safety PLC     E-stop signal
   [External        safety status
   System]
 Emergency
 stop
```

1.5.2 3.2 Context Description

1.5.2.1 3.2.1 People

Person	Role	Interaction
Operator	Runs daily operations	Uses web UI to start/stop system, monitor status, handle errors
Manager	Oversees performance	Reviews KPIs, ROI metrics via dashboard
Engineer	Develops & maintains	Deploys code, debugs issues, optimizes performance via ROS2 tools
Maintenance Technician	Repairs & services	Diagnoses faults, performs preventive maintenance
Data Scientist	Trains AI models	Collects training data, updates object detection models

1.5.2.2 3.2.2 External Systems

System	Purpose	Integration
MES (Manufacturing Execution System)	Work orders, production tracking	REST API (bidirectional)

System	Purpose	Integration
ERP (Enterprise Resource Planning)	Inventory, analytics	REST API (robot → ERP)
Safety PLC	Emergency stop, safety zones	Digital I/O, Modbus TCP
Robot Hardware (UR5e)	Physical manipulation	EtherCAT, URScript
Camera (RealSense D435i)	Vision sensing	USB 3.0, librealsense2 SDK
Gripper (Robotiq 2F-85)	Grasping	Modbus RTU over RS-485

1.6 4. Level 2: Container Diagram

1.6.1 4.1 Container Diagram

```
                       Robot System



     Vision Pipeline           Motion Planning
     [Docker/GPU]           [Docker/CPU]
                      Object
    - Camera Driver   poses    - MoveIt2
    - YOLOv8                   - OMPL Planner
    - Pose Estimator           - Collision Check
    - TensorRT
                                ROS2 C++/Python
     Python/C++


                                        Trajectory
             ROS2 topics



                                Control System
                                [RT Docker]

                               - ros2_control
                               - Joint Ctrl
                               - HW Interface

                                C++/RT-Linux



                                        EtherCAT
```

```
 Task                       Gripper Control
 Orchestrator           [Docker]
 [Docker]          Grasp
                   cmd     - Robotiq Driver
- Behavior Tree            - Force Control
- State Machine
- Task Manager              Python

 C++/Python
                           Modbus RTU

         ROS2 actions




          ROS2 DDS           (CycloneDDS message bus)






 Web Backend
 [Docker]

- FastAPI
- REST/gRPC API
- SQLAlchemy

 Python 3.11

         HTTPS/WS


 Web Frontend
 [Docker/Nginx]

- React 18
- Next.js
- Chart.js

 TypeScript/JS


         HTTPS
```

```
  Operator               PostgreSQL
  [Browser]              [Container]

Chrome/Firefox        Operational
              data (picks,
                       configs)

                       v15.3

 Grafana
 [Container]
                       InfluxDB
- Dashboards          [Container]
- Alerts
                      Time-series
v10.0                 metrics

                        v2.7



 Prometheus              Metrics
 [Container]           Collector
                         [Container]
Scrapes metrics
from all nodes        Publishes to
                      Influx/Prom
v2.45
              Python
```

1.6.2 4.2 Container Descriptions

1.6.2.1 4.2.1 Vision Pipeline Container

Attribute	**Details**
Technology	Python 3.10, C++17, ROS2 Humble, PyTorch 2.0, TensorRT 8.5
Runtime	Docker with NVIDIA GPU support
Responsibilities	- Capture RGB-D images- Detect objects (YOLOv8)- Estimate 6DoF poses (PCA/PVNet)- Publish object poses via ROS2

Attribute	Details
Data In	Raw camera frames (USB 3.0)
Data Out	`/vision/object_poses` (ROS2 topic)
Dependencies	RealSense D435i camera, CUDA runtime
Scaling	Single instance (stateless, can scale horizontally with multiple cameras)

1.6.2.2 4.2.2 Motion Planning Container

Attribute	Details
Technology	C++17, ROS2 Humble, MoveIt2 2.5, OMPL 1.6
Runtime	Docker (CPU-only)
Responsibilities	- Plan collision-free trajectories- Inverse kinematics- Cartesian path planning- Collision checking
Data In	Target poses (grasp/place) via ROS2 actions
Data Out	Joint trajectories
Dependencies	Robot URDF, collision meshes
Scaling	Single instance (stateful due to planning scene)

1.6.2.3 4.2.3 Control System Container

Attribute	Details
Technology	C++17, ROS2 Humble, ros2_control 2.27, RT-Linux kernel (PREEMPT_RT)
Runtime	Docker with real-time capabilities (–cap-add=SYS_NICE, –ulimit rtprio=99)
Responsibilities	- 1kHz control loop- Execute joint trajectories- Low-level robot communication (EtherCAT)- Safety monitoring
Data In	Joint trajectories from motion planner
Data Out	Joint states (positions, velocities, efforts)
Dependencies	UR5e robot (EtherCAT interface)
Scaling	Single instance per robot (not scalable)

1.6.2.4 4.2.4 Task Orchestrator Container

Attribute	Details
Technology	C++/Python, ROS2 Humble, BehaviorTree.CPP 4.0
Runtime	Docker

Attribute	Details
Responsibilities	- High-level task sequencing- State machine management- Error recovery logic- Workflow coordination
Data In	System triggers, sensor data
Data Out	ROS2 actions to subsystems (vision, motion, gripper)
Dependencies	All subsystem ROS2 action servers
Scaling	Single instance (stateful orchestration)

1.6.2.5 4.2.5 Web Backend Container

Attribute	Details
Technology	Python 3.11, FastAPI 0.104, SQLAlchemy 2.0, gRPC
Runtime	Docker with Gunicorn (4 workers)
Responsibilities	- REST/gRPC API for UI- Database CRUD operations- Authentication (OAuth2/JWT)- ROS2 bridge (rclpy client)
Data In	HTTP requests from frontend
Data Out	JSON responses, ROS2 messages
Dependencies	PostgreSQL, ROS2 nodes
Scaling	Horizontal (stateless, load balanced)

1.6.2.6 4.2.6 Web Frontend Container

Attribute	Details
Technology	TypeScript, React 18, Next.js 14, Chart.js, WebSockets
Runtime	Docker with Nginx (static file serving)
Responsibilities	- User interface- Real-time dashboard- System configuration forms- Live video feed
Data In	REST API responses, WebSocket streams
Data Out	HTTP requests to backend
Dependencies	Web Backend API
Scaling	Horizontal (CDN-ready static files)

1.6.2.7 4.2.7 PostgreSQL Container

Attribute	Details
Technology	PostgreSQL 15.3
Runtime	Docker with persistent volume

Attribute	Details
Responsibilities	- Store operational data (picks, configs, calibrations)- Transactional guarantees- Query analytics
Data In	SQL queries from Web Backend
Data Out	Query results
Dependencies	None (standalone)
Scaling	Vertical (read replicas for scaling reads)

1.6.2.8 4.2.8 InfluxDB Container

Attribute	Details
Technology	InfluxDB 2.7 (time-series database)
Runtime	Docker with persistent volume
Responsibilities	- Store time-series metrics (cycle time, latency, joint states)- High-write throughput- Retention policies
Data In	Metrics from Metrics Collector
Data Out	Query results for Grafana
Dependencies	None
Scaling	Horizontal (InfluxDB Enterprise clustering)

1.7 5. Level 3: Component Diagrams

1.7.1 5.1 Vision Pipeline Components

```
              Vision Pipeline Container


     Camera Driver       /camera/color/image_raw
     [ROS2 Node]

    - realsense_node             sensor_msgs/Image
    - USB streaming




     Image Processor       Image Buffer
     [ROS2 Node]            [Shared Memory]

    - Denoise (OpenCV)     Ring buffer (10)
    - Color correction

             /vision/preprocessed/image
```

```
 Object Detector     /vision/detections
 [ROS2 Node]

- YOLOv8 inference            vision_msgs/Detection2D
- TensorRT engine
- GPU acceleration



 Pose Estimator       Detection Sync
 [ROS2 Node]           [ApproxTimeSynch]

- PCA-based pose       Sync depth +
- Point cloud proc     detections
- TF2 broadcaster

        /vision/object_poses
        /tf (object frames)


 Visualization       /vision/debug/detections
 [ROS2 Node]

- Draw bboxes         For debugging in RViz2
- Overlay poses
```

1.7.2 5.2 Motion Planning Components

```
        Motion Planning Container


 Pick Place Server   /pick_place (action server)
 [ROS2 Action]

- Sequence pick/
  place motions


        Calls
 MoveGroup
 Interface
 [MoveIt2 API]

- setPoseTarget()
- plan()
- execute()


       Uses


 OMPL Planner           IK Solver
 [Plugin]               [KDL Plugin]

- RRTConnect           - Analytical IK
- RRT*                 - Numerical IK
- PRM                  - Multiple sols


       Queries


 Planning Scene
 [Shared State]

- Robot state
- Collision objects
- Allowed collis.


       Checks


 Collision Checker
 [FCL Library]

- Self collision
- Env collision
- Continuous check



 Trajectory           Publishes smoothed trajectory
 Processor
 [Time-optimal]
                      /joint_trajectory (topic)
- Velocity limits
```

```
    - Accel limits
    - Jerk limits
```

1.7.3 5.3 Control System Components

```
             Control System Container

     Controller          Main control loop (1 kHz)
     Manager
     [ros2_control]

    - Load controllers
    - Update loop
    - State publishing

           Updates (1 kHz)

     Joint Trajectory
     Controller
     [Plugin]

    - Interpolate traj
    - PID control
    - Feedforward

           Commands

     Hardware
     Interface
     [Custom Plugin]

    - read()  (state)
    - write() (cmd)

           EtherCAT

     UR Driver
```

```
 [Library]

- TCP/IP socket
- URScript
- Real-time client



 Joint State         Publishes current joint states
 Broadcaster
 [ros2_control]
                     /joint_states (topic, 100 Hz)
- Position
- Velocity
- Effort



 Safety Monitor      Monitors limits, publishes alerts
 [ROS2 Node]

- Joint limits       /safety/alerts (topic)
- Velocity limits
- Force limits
```

1.7.4 5.4 Task Orchestrator Components

```
        Task Orchestrator Container


 Task Manager        Main entry point
 [ROS2 Node]

- Start/stop tasks
- Error recovery


       Executes


 Behavior Tree
 Engine
 [BT.CPP]
```

```
- Load XML tree
- Tick nodes
- Blackboard


        Ticks


 Action Nodes           Condition Nodes
 [Plugins]              [Plugins]

- CaptureImage         - ObjectDetected
- DetectObjects        - GraspValid
- PlanGrasp            - WaitForTrigger
- ExecutePick
- ExecutePlace


        Calls ROS2 services/actions


 Service Clients
 [ROS2 Clients]

- /vision/detect
- /grasp/compute
- /pick_place



 Blackboard           Shared state between BT nodes
 [Memory Store]

- detections
- target_object
- grasp_pose
- error_count




 State Publisher      Publishes orchestrator state
 [ROS2 Publisher]

 /task/status         TaskStatus msg (10 Hz)
```

1.8 6. Level 4: Code Diagrams

1.8.1 6.1 YoloDetector Class Diagram

```
                  YoloDetector

 - model: torch.nn.Module
 - device: torch.device
 - bridge: CvBridge
 - confidence_threshold: float
 - iou_threshold: float
 - image_sub: Subscriber<Image>
 - detections_pub: Publisher<Detection2DArray>
 - debug_image_pub: Publisher<Image>

 + __init__()
 + load_model() -> None
 + image_callback(msg: Image) -> None
 + run_inference(image: np.ndarray) -> List[Detection]
 + parse_detections(results) -> Detection2DArray
 + draw_detections(image, detections) -> np.ndarray
 - preprocess_image(image: np.ndarray) -> torch.Tensor
 - postprocess_results(outputs) -> List[BBox]
 - apply_nms(boxes, scores) -> List[int]


        uses


                TensorRTEngine

 - engine: trt.ICudaEngine
 - context: trt.IExecutionContext
 - input_shape: Tuple[int, int, int]
 - bindings: List[int]

 + __init__(engine_path: str)
 + infer(input_tensor: torch.Tensor) -> torch.Tensor
 - allocate_buffers() -> None
```

1.8.2 6.2 GraspSynthesizer Class Diagram

```
               GraspSynthesizer
```

```
 - service: Service<ComputeGrasps>
 - num_candidates: int
 - gripper_max_width: float
 - friction_coeff: float
 - quality_evaluator: GraspQualityEvaluator

 + __init__()
 + compute_grasps_callback(req, res) -> Response
 + generate_box_grasps(req) -> List[Grasp]
 + generate_cylinder_grasps(req) -> List[Grasp]
 + generate_generic_grasps(req) -> List[Grasp]
 - compute_grasp_pose(obj_pose, approach, rot) -> Pose
 - check_force_closure(grasp) -> bool


          uses


             GraspQualityEvaluator

 - metric_type: str  # "ferrari_canny" | "volume"

 + evaluate(grasp: Grasp, object: Object) -> float
 - compute_ferrari_canny(contact_pts) -> float
 - compute_grasp_wrench_space(contacts) -> np.ndarray
```

1.8.3 6.3 PickPlaceServer Class Diagram

```
                  PickPlaceServer

 - action_server: ActionServer<PickPlace>
 - move_group: MoveGroupInterface
 - gripper_client: ActionClient<GripperCommand>
 - planning_scene: PlanningScene

 + __init__(node_options)
 + handle_goal(uuid, goal) -> GoalResponse
 + handle_cancel(goal_handle) -> CancelResponse
 + handle_accepted(goal_handle) -> None
 - execute(goal_handle) -> None
 - plan_pick(target, approach_dist) -> Plan
 - plan_place(target) -> Plan
 - execute_trajectory(plan) -> bool
 - close_gripper(width, force) -> bool
 - open_gripper() -> bool
 - retreat(distance) -> bool
```

```
  - publish_feedback(handle, status, progress) -> None


         uses


          MoveGroupInterface (MoveIt2)

  - robot_model: RobotModel
  - planning_scene_monitor: PlanningSceneMonitor
  - trajectory_execution_manager: TrajectoryExecutionMgr

  + setPoseTarget(pose: Pose) -> None
  + setJointValueTarget(joints: List[float]) -> None
  + plan(plan: Plan&) -> MoveItErrorCode
  + execute(plan: Plan) -> MoveItErrorCode
  + computeCartesianPath(waypoints) -> double
  + getCurrentPose() -> PoseStamped
  + getCurrentJointValues() -> List[float]
```

1.9 7. Cross-Cutting Concerns

1.9.1 7.1 Logging Architecture

```
              All ROS2 Nodes
   (Vision, Motion, Control, Orchestrator, etc.)



            rclcpp::Logger
            (stdout/stderr)



             Docker Log Driver
   (json-file or syslog)



            JSON logs



              Filebeat
   (Log shipper)



            Forwards
             Logstash
   (Log parsing, enrichment)
   - Parse JSON
   - Add metadata (hostname, container)
   - Filter by log level


            Indexes


            Elasticsearch
   (Log storage & search)
   - Index: logs-robot-YYYY.MM.DD
   - Retention: 30 days


            Queries


             Kibana
   (Log visualization)
   - Dashboards
   - Alerts (error rate > 10/min)
```

1.9.2 7.2 Monitoring & Metrics

```
        Metrics Collector (Python ROS2 Node)
  - Subscribes to /task/status, /joint_states, etc.
  - Publishes Prometheus metrics on :8000/metrics
  - Writes to InfluxDB


           Scrapes (15s interval)


            Prometheus
  (Metrics storage & alerting)
  - Time-series DB
  - Retention: 15 days
  - Alert rules (uptime < 99%, cycle_time > 3s)


           Queries (PromQL)
```

```
              Grafana
   (Dashboards & visualization)
   - Real-time dashboard (refresh 5s)
   - Historical trends
   - Alerts to Slack/email
```

1.9.3 7.3 Security Architecture

```
                 User (Browser)


             HTTPS (TLS 1.3)


              Nginx Reverse Proxy
   - TLS termination
   - Rate limiting (100 req/min per IP)
   - Firewall rules (deny all except 443)


             HTTP (internal network)


            Web Backend (FastAPI)
   - OAuth2/JWT authentication
   - RBAC (Operator, Engineer, Admin roles)
   - API key for MES/ERP


             SQL (prepared statements)


              PostgreSQL
   - TLS encryption
   - User permissions (least privilege)
   - Audit logging enabled
```

ROS2 Security:

```
              ROS2 DDS (CycloneDDS)
   - DDS Security (SROS2)
   - Encrypted topics (AES-256)
   - Authentication (X.509 certificates)
   - Access control lists (permissions.xml)
```

1.10 8. Deployment View

1.10.1 8.1 Physical Deployment (Production)

```
             Intel NUC (Main Compute)
CPU: Intel i7-12700H (12 cores)
RAM: 32 GB DDR4
Disk: 1 TB NVMe SSD
OS: Ubuntu 22.04 LTS + Docker


 Motion Planning     Orchestrator
 (Docker)            (Docker)




 Control System      Web Backend
 (Docker RT)         (Docker)




 PostgreSQL          Prometheus
 (Docker)            (Docker)




              Gigabit Ethernet



        Jetson Xavier NX (Edge Compute)
GPU: 384-core NVIDIA Volta
CPU: 6-core NVIDIA Carmel ARM64
RAM: 8 GB LPDDR4
OS: Jetson Linux (L4T) + Docker


 Vision Pipeline     Metrics
 (Docker GPU)        Collector
                     (Docker)
 - YOLOv8 +
   TensorRT
 - Pose Estimator
```

```
                USB 3.0



        RealSense D435i
        (Camera)





        EtherCAT                  Modbus RTU

  UR5e Robot                 Robotiq Gripper
  (6-DOF Arm)                (2F-85)
```

1.10.2 8.2 Logical Deployment (Docker Network)

```
              robot_net (Bridge Network)
              Subnet: 172.20.0.0/16

  Vision Pipeline      172.20.0.10
  Motion Planning      172.20.0.20
  Control System      172.20.0.30
  Orchestrator       172.20.0.40
  Web Backend       172.20.0.50
  Web Frontend       172.20.0.60
  PostgreSQL        172.20.0.70
  InfluxDB         172.20.0.71
  Prometheus        172.20.0.80
  Grafana          172.20.0.90
  Metrics Collector     172.20.0.100

  (All containers communicate via this internal network)
  (No external access except Web Frontend on port 443)


Host Network Ports:
  - 443 (HTTPS) → Nginx → Web Frontend
  - 3000 (Grafana UI) → Grafana
  - 5432 (PostgreSQL) → Blocked externally
  - 9090 (Prometheus) → Blocked externally
```

1.11 9. Dynamic Views

1.11.1 9.1 Pick and Place Sequence (Simplified C4 Dynamic)

```
Operator           Web UI        Backend       Orchestrator   Vision    Motion    Control
```

```
     Click "Start"
            >
                      POST /start
                            >
                                    StartTask()
                                          >
                                                   Capture()
                                                        >
                                                   Objects
                                                  <
                                                   PlanGrasp()
                                                              >
                                                   GraspPose
                                                  <
                                                   ExecutePick()
                                                                   >
                                                                           Done
                                                  <
                      200 OK
                     <
     Status update
    <
```

1.11.2 9.2 Error Recovery (Dynamic Behavior)

```
Orchestrator      Vision       Grasp Planner    Motion       Control

       DetectObjects
             >
       Empty[]
      <

       [Retry 1/3]
       DetectObjects
             >
       [cube]
      <

       ComputeGrasps
                      >
       Grasp
      <

       ExecutePick
                                   >
       FAILED
```

```
<

 [Retry 1/3 with adjusted force]
 ExecutePick(force=25N)
                               >
 SUCCESS
<
```

1.12 10. Summary

1.12.1 10.1 C4 Model Completeness

Level 1 (Context): System scope, external actors, external systems **Level 2 (Containers):** 11 runtime containers (Docker) with technologies **Level 3 (Components):** Detailed breakdown of Vision, Motion, Control, Orchestrator **Level 4 (Code):** Class diagrams for critical components (YoloDetector, GraspSynthesizer, PickPlaceServer)

1.12.2 10.2 Key Architectural Patterns

Pattern	Application
Microservices	Each subsystem is an independent Docker container
Pub/Sub	ROS2 DDS for asynchronous communication
Request/Reply	ROS2 services for synchronous operations
Action Pattern	Long-running tasks (motion planning, pick/place)
Layered Architecture	Clear separation: Hardware → Firmware → Middleware → Application → UI
Repository Pattern	Database access via SQLAlchemy ORM
Dependency Injection	Constructor injection for testability

1.12.3 10.3 Technology Summary

C4 Level	Diagram Count	Technologies Visualized
C1: Context	1	Operator, Manager, Engineer, MES, ERP, Safety PLC, UR5e, Camera, Gripper
C2: Containers	1	11 Docker containers (Vision, Motion, Control, Orchestrator, Backend, Frontend, PostgreSQL, InfluxDB, Prometheus, Grafana, Metrics)
C3: Components	4	20+ components (Camera Driver, YOLOv8, Pose Estimator, MoveIt2, ros2_control, Behavior Tree, etc.)
C4: Code	3	3 critical classes (YoloDetector, GraspSynthesizer, PickPlaceServer)

C4 Level	Diagram Count	Technologies Visualized
Cross-Cutting	3	Logging (ELK), Monitoring (Prometheus/Grafana), Security (OAuth2/TLS)
Deployment	2	Physical (NUC + Jetson) and Logical (Docker network)
Dynamic	2	Pick-place sequence, Error recovery

1.12.4 10.4 Next Steps

1. **Validation:** Review diagrams with stakeholders (operators, engineers, architects)
2. **Implementation:** Use C3/C4 diagrams as blueprints for coding
3. **Documentation:** Generate PlantUML or Structurizr DSL for automated rendering
4. **Updates:** Keep diagrams synchronized with code changes (living documentation)

Document Status: v1.0 Complete **Next Document:** Building Block Diagrams (module decomposition, data flow) **Dependencies:** High-Level Design (08), Low-Level Design (14), Technical Stack (05)
