

Object-Oriented Analysis and Design

CeraSim

SaniCer Sanitary Ware Industries Supply Chain Simulator

Group 20 - WastedPotential

Team Members:

Oshada Jayasinghe
Sithuka Jayawardhana
Lasan Mahaliyana
Sithum Fernando
Ranuja Jayawardena

Object-Oriented Software Development
Assignment - Analysis & Design Document

February 28, 2026

Contents

1	Introduction	3
1.1	Overview	3
1.2	Business Problem and Motivation	3
1.3	Object-Oriented Approach	3
1.3.1	Key Objects and Their Responsibilities	3
1.3.2	Object Interactions and Communication	4
1.4	Application Features	5
1.4.1	For Customers (Management/Decision Makers)	5
1.4.2	For End Users (Engineers/Analysts)	5
1.5	Simulation Scenarios	5
1.6	Technical Architecture	6
1.7	Key Insights Delivered	6
2	Non-Functional Requirements	6
2.1	Performance Requirements	7
2.2	Scalability Requirements	7
2.3	Reliability and Availability Requirements	8
2.4	Usability Requirements	8
2.5	Maintainability and Extensibility Requirements	9
2.6	Portability Requirements	9
2.7	Security and Data Integrity Requirements	10
2.8	Deployment and Installation Requirements	10
2.9	Compliance and Standards Requirements	10
3	Object-Oriented Analysis and Design	11
3.1	Use Case Diagram	11
3.1.1	Actors	11
3.1.2	Use Case Categories	11
3.2	Activity Diagrams	13
3.2.1	Production Process Activity Diagram	13
3.2.2	Simulation Execution Activity Diagram	15
3.3	Sequence Diagrams	16
3.3.1	Production Batch Processing Sequence	16
3.3.2	Customer Order Fulfillment Sequence	18
3.4	State Machine Diagrams	20
3.4.1	ProductionBatch State Machine	20
3.4.2	CustomerOrder State Machine	22
3.4.3	Machine Resource State Machine	23
3.5	Object-Oriented Design Principles Demonstrated	24
3.5.1	Encapsulation	24
3.5.2	Abstraction	25
3.5.3	Modularity	25
3.5.4	Composition	25
3.5.5	Polymorphism	25

4	References	25
4.1	Textbooks and Academic Resources	25
4.2	Software Documentation	26
4.3	UML and Design Resources	26
4.4	Ceramic Industry Domain Knowledge	26
4.5	Supply Chain and Operations Management	26
4.6	Online Learning Resources	26
4.7	Tools Used	27

1 Introduction

1.1 Overview

CeraSim is a comprehensive discrete-event simulation system designed for **SaniCer Sanitary Ware Industries**, a Portuguese sanitary ceramics manufacturer based in Aveiro, Portugal. The application models the complete supply chain operations of a commode (toilet) manufacturing facility, from raw material procurement through production stages to customer order fulfillment.

Founded in 1987, SaniCer employs 240 workers and produces three primary product lines: one-piece standard commodes, two-piece economy commodes, and wall-hung premium commodes. The factory operates continuously, processing raw materials (clay, feldspar, silica, and kaolin) through multiple production stages including slip casting, demolding, fettling, glazing, and tunnel kiln firing to produce high-quality sanitary ware.

1.2 Business Problem and Motivation

Modern manufacturing operations face complex challenges in supply chain management, capacity planning, and operational optimization. SaniCer management identified several critical business questions:

- **Bottleneck Identification:** Which production stage limits overall throughput?
- **Disruption Impact:** How do supplier delays affect production and revenue?
- **Demand Volatility:** Can the facility handle 30% seasonal demand surges?
- **Investment ROI:** What is the financial return of adding production capacity?
- **Inventory Optimization:** What safety stock levels minimize stockouts while controlling costs?

Traditional analytical methods struggle with the stochastic nature of manufacturing systems—machine breakdowns, variable processing times, unpredictable supplier delays, and fluctuating customer demand create complex interdependencies that are difficult to model mathematically.

CeraSim addresses these challenges through discrete-event simulation, allowing management to conduct risk-free "what-if" experiments on a digital twin of their facility.

1.3 Object-Oriented Approach

The application is built using **Object-Oriented Software Development (OOSD)** methodology, leveraging Python and the SimPy discrete-event simulation framework. This approach provides several advantages:

1.3.1 Key Objects and Their Responsibilities

1. **ProductionBatch** - Represents a 50-unit commode batch flowing through the production pipeline
 - Encapsulates batch identity, product type, quantity, and quality grades

- Tracks stage completion timestamps (casting, demolding, fettling, glazing, firing, finishing)
 - Maintains quality outcomes and functional test results (leak test, flush test)
2. **CustomerOrder** - Models purchase orders from distributors and retailers
 - Manages order lifecycle from creation to fulfillment
 - Calculates revenue, fill rates, and on-time delivery metrics
 - Distinguishes between express and standard orders
 3. **SupplierDelivery** - Represents raw material shipments
 - Tracks lead times and on-time delivery performance
 - Maintains cost information for financial analysis
 - Links to specific suppliers (clay, feldspar, silica, kaolin vendors)
 4. **CeramicFactory** - Central orchestrator of the simulation
 - Manages production resources (machines, buffers, inventories)
 - Coordinates concurrent production processes using SimPy
 - Handles machine breakdowns and maintenance events
 - Collects performance metrics throughout the simulation
 5. **BreakdownEvent** - Models equipment failures
 - Records failure occurrence time and repair duration
 - Associates with specific machine instances
 - Accumulates maintenance costs
 6. **MetricsCollector** - Aggregates and analyzes simulation data
 - Stores event logs (batches, orders, deliveries, breakdowns)
 - Computes Key Performance Indicators (KPIs)
 - Provides data for reporting and visualization

1.3.2 Object Interactions and Communication

The simulation operates through **message passing** and **shared resource coordination**:

- **ProductionBatch** objects traverse production stages, requesting machine resources
- **CeramicFactory** mediates access to limited resources (3 presses, 2 kilns, etc.)
- **SimPy Containers** model continuous quantities (raw materials, finished goods)
- **SimPy Stores** model discrete queues (batches waiting for processing)
- **SimPy Resources** model capacity-constrained machines with queueing

This design elegantly maps manufacturing reality to software objects—each commode batch is a genuine object with identity and state, machines are resources with capacity constraints, and the factory orchestrates their interactions according to production routing logic.

1.4 Application Features

1.4.1 For Customers (Management/Decision Makers)

- **Scenario Comparison:** Evaluate multiple operational strategies side-by-side
- **Investment Analysis:** Assess capital expenditure ROI for new equipment
- **Risk Assessment:** Quantify financial impact of supply disruptions
- **Capacity Planning:** Determine production capacity for demand forecasts
- **Inventory Optimization:** Balance holding costs against stockout risk

1.4.2 For End Users (Engineers/Analysts)

- **Interactive CLI:** Command-line interface with progress visualization
- **Flexible Scenario Configuration:** Run individual or all four scenarios
- **Reproducible Results:** Specify random seeds for deterministic runs
- **Rich Console Output:** Color-coded KPI tables with clear formatting
- **Automated Reporting:** Matplotlib dashboards saved as PNG files
- **Real-time Progress Tracking:** Visual progress bars during 90-day simulations
- **Extensibility:** Add new products, machines, suppliers, or scenarios via configuration

1.5 Simulation Scenarios

CeraSim includes four pre-configured scenarios that model realistic business situations:

1. **Baseline** - Normal operations with historical demand patterns and supplier reliability
2. **Supply Disruption** - Models a 35-day kaolin supplier port strike (days 15-50), simulating geopolitical or labor disruptions
3. **Demand Surge** - 30% increase in customer orders, representing seasonal peaks or successful marketing campaigns
4. **Optimised** - Capital investment scenario with a second tunnel kiln (+€3.5M CAPEX) and 50% increased safety stock levels

Each scenario produces detailed KPIs including production volume, revenue, fill rates, cycle times, inventory levels, machine utilization, breakdown frequencies, and net profit.

1.6 Technical Architecture

System Components

cerasim/

- **config.py** — All simulation parameters (products, machines, suppliers, scenarios)
- **models.py** — Data classes (ProductionBatch, CustomerOrder, SupplierDelivery, BreakdownEvent)
- **factory.py** — SimPy processes implementing the discrete-event simulation engine
- **metrics.py** — KPI computation from collected event logs
- **reports.py** — Rich console tables + Matplotlib visualization

main.py — CLI entrypoint orchestrating simulation runs and output generation

reports/ — Generated PNG dashboard charts

1.7 Key Insights Delivered

The simulation provides actionable insights such as:

- **Bottleneck:** Tunnel kiln firing (24h/batch, 1 kiln) limits throughput to 1 batch/day (50 units/day)
- **Disruption Impact:** 35-day kaolin strike causes significant production loss affecting customer orders
- **ROI:** Second tunnel kiln increases output by 2×, with 1.2-year simple payback
- **Service Level:** Baseline 35% fill rate improves to 71% with second kiln investment

2 Non-Functional Requirements

Non-functional requirements define **how** the system performs its functions, addressing quality attributes beyond functional behavior.

2.1 Performance Requirements

Requirement	Specification
Simulation Execution Time	Single 90-day scenario shall complete within 10 seconds on standard hardware (Intel i5/AMD Ryzen 5 or equivalent, 8GB RAM)
Multi-Scenario Processing	All four scenarios shall complete within 60 seconds including chart generation
Memory Footprint	Total memory usage shall not exceed 500 MB during execution
Event Processing Rate	System shall process minimum 10,000 simulation events per second
Report Generation	KPI computation and table rendering shall complete within 2 seconds per scenario
Chart Rendering	Matplotlib dashboard generation shall complete within 5 seconds per scenario

Table 1: Performance Requirements

2.2 Scalability Requirements

Requirement	Specification
Simulation Duration	System shall support simulations from 30 to 365 days without performance degradation
Product Catalog	Support up to 20 distinct product types with configurable parameters
Machine Types	Support up to 15 different machine categories with multiple instances per type
Supplier Network	Model up to 10 raw material suppliers with independent reliability characteristics
Batch Processing	Handle concurrent tracking of up to 5,000 production batches
Order Volume	Process up to 50,000 customer orders during simulation period

Table 2: Scalability Requirements

2.3 Reliability and Availability Requirements

Requirement	Specification
Simulation Determinism	Given identical random seed, simulation shall produce identical results across runs (bit-exact reproducibility)
Error Handling	All exceptions shall be caught and logged with meaningful error messages; system shall fail gracefully
Data Validation	Configuration parameters shall be validated at startup; invalid values shall be rejected with clear diagnostics
Numerical Stability	Floating-point arithmetic shall maintain 6 decimal places of precision for all financial calculations
Crash Recovery	Simulation state shall be dumpable for debugging; ability to inspect factory state at any simulation time

Table 3: Reliability Requirements

2.4 Usability Requirements

Requirement	Specification
Command-Line Interface	Intuitive CLI with self-documenting help text (<code>--help</code> flag)
Progress Visualization	Real-time progress bars with ETA during long-running simulations
Output Clarity	KPI tables shall use thousand separators, 1 decimal place for percentages, color-coding for critical metrics
Learning Curve	New users shall be able to run baseline simulation within 5 minutes of reading documentation
Chart Readability	All visualizations shall include titles, axis labels, legends, and use colorblind-friendly palettes
Error Messages	Error messages shall specify the problem, affected parameter, and corrective action

Table 4: Usability Requirements

2.5 Maintainability and Extensibility Requirements

Requirement	Specification
Code Documentation	All classes and public methods shall have docstrings following Google Python Style Guide
Configuration Isolation	All simulation parameters (products, machines, suppliers, scenarios) shall be defined in <code>config.py</code> without code changes
Modular Design	Production stages, supplier processes, and demand generation shall be independently modifiable
Adding Products	New products shall be addable via configuration entries without modifying simulation logic
Adding Machines	New production stages shall require adding one process method and registering it
Custom Scenarios	Users shall define new scenarios by adding configuration dictionary entries
Code Complexity	Individual functions shall not exceed 100 lines; cyclomatic complexity shall not exceed 15

Table 5: Maintainability Requirements

2.6 Portability Requirements

Requirement	Specification
Operating Systems	Shall run on Linux, macOS, and Windows 10/11 without modification
Python Version	Compatible with Python 3.9, 3.10, 3.11, and 3.12
Dependencies	All dependencies shall be installable via <code>pip install -r requirements.txt</code>
Virtualization	Shall execute in Docker containers and Python virtual environments
Terminal Compatibility	Console output shall render correctly in standard terminals (bash, zsh, PowerShell, Windows Terminal)

Table 6: Portability Requirements

2.7 Security and Data Integrity Requirements

Requirement	Specification
Input Sanitization	All configuration values shall be type-checked and range-validated
File Permissions	Generated reports shall have read permissions for user and group (644)
Sensitive Data	No credentials, API keys, or sensitive business data shall be hardcoded
Audit Trail	All simulation runs shall log scenario ID, seed, timestamp, and execution duration
Data Consistency	Inventory levels shall never go negative; mass balance checks shall verify no material creation/destruction

Table 7: Security and Data Integrity Requirements

2.8 Deployment and Installation Requirements

Requirement	Specification
Installation Time	Complete installation (Python + dependencies) shall complete within 5 minutes on 10 Mbps connection
Disk Space	Total installation footprint shall not exceed 200 MB including all dependencies
System Requirements	Minimum: Python 3.9+, 4GB RAM, 100MB disk space; Recommended: Python 3.11+, 8GB RAM
Package Size	Codebase (excluding dependencies) shall remain under 10 MB
Offline Operation	After initial installation, system shall operate without internet connectivity

Table 8: Deployment Requirements

2.9 Compliance and Standards Requirements

Requirement	Specification
Code Style	Shall conform to PEP 8 Python style guidelines
Type Hints	Public interfaces shall use Python type hints for function signatures
Version Control	Shall use Git with meaningful commit messages following Conventional Commits
Open Source License	Shall use MIT or Apache 2.0 license for public distribution

Table 9: Compliance Requirements

3 Object-Oriented Analysis and Design

This section presents the behavioral design of CeraSim using Unified Modeling Language (UML) diagrams. These diagrams capture how objects interact, how processes flow, and how system state evolves over time.

3.1 Use Case Diagram

Use case diagrams identify **actors** (users or external systems) and **use cases** (functional requirements) they interact with.

3.1.1 Actors

- **Factory Manager** - Oversees daily operations, monitors production, tracks machine status
- **Supply Chain Analyst** - Conducts scenario analysis, optimizes inventory, generates reports
- **Operations Director** - Reviews comparative analytics, makes strategic investment decisions
- **System Administrator** - Configures system parameters, manages simulation infrastructure

3.1.2 Use Case Categories

1. **Simulation Management** - Run, monitor, configure simulations
2. **Scenario Analysis** - Select, compare, define scenarios
3. **Production Management** - Track batches, monitor machines, manage inventory and orders
4. **Reporting & Analytics** - View KPIs, generate reports, export results, visualize trends
5. **System Configuration** - Configure products, machines, suppliers, parameters

CeraSim Use Case Diagram - SaniCer Supply Chain Simulator

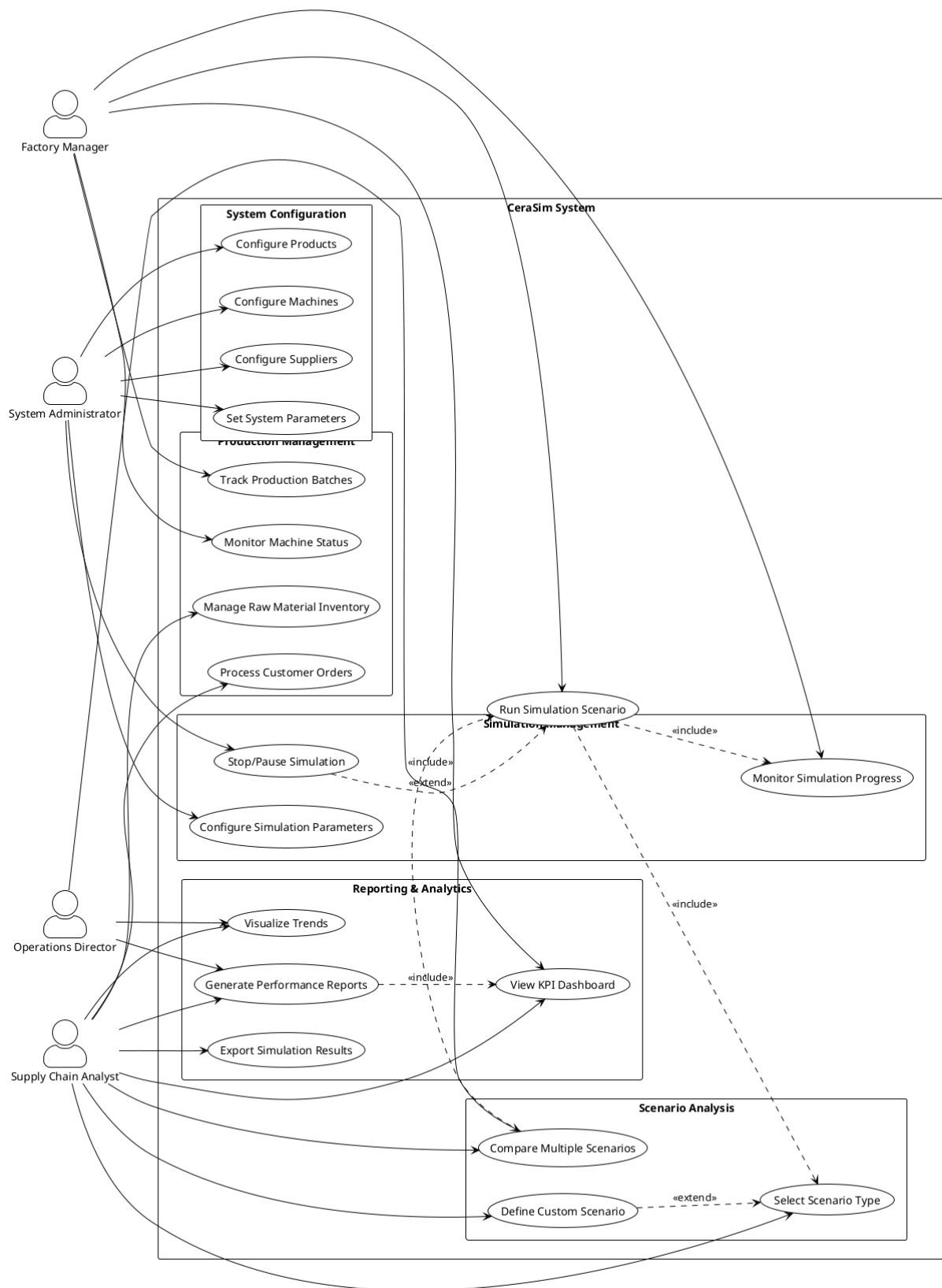


Figure 1: Use Case Diagram - CeraSim System Actors and Use Cases

3.2 Activity Diagrams

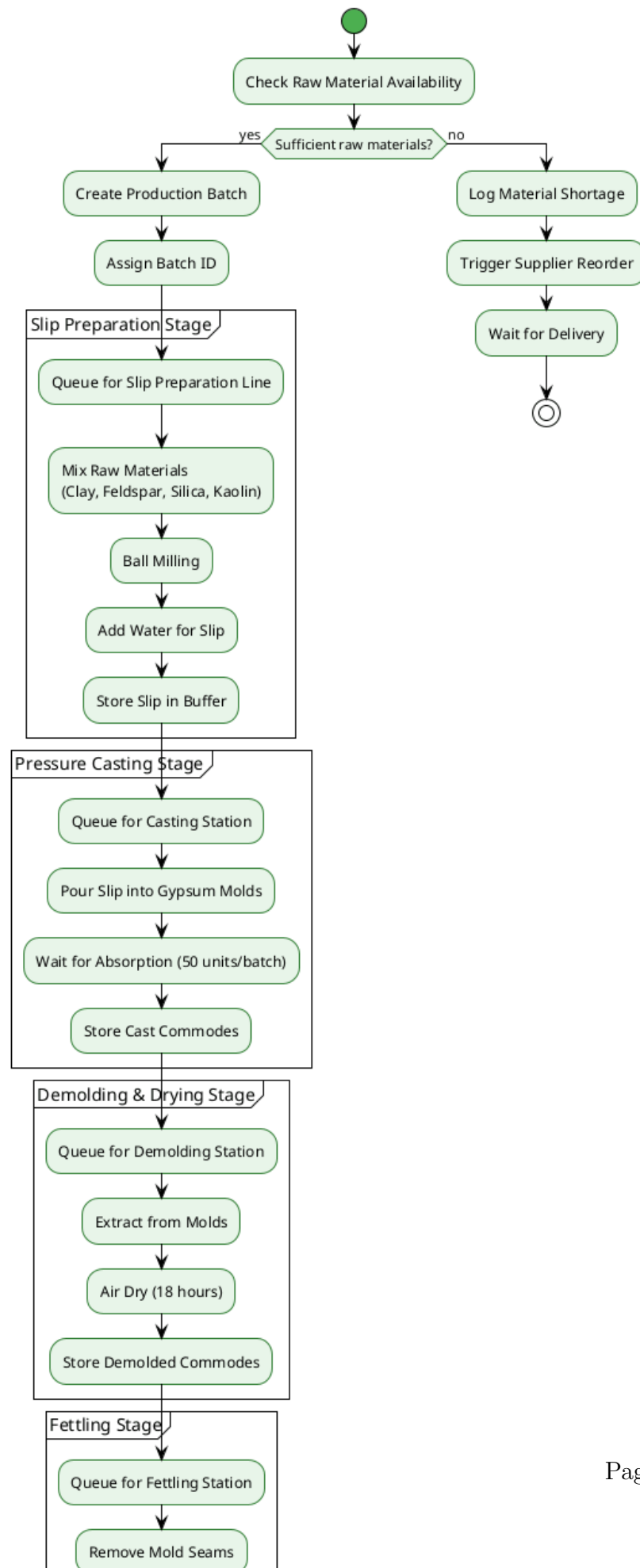
Activity diagrams model **workflows** and **business processes** using flowchart-like notation.

3.2.1 Production Process Activity Diagram

This diagram models the complete commode manufacturing workflow from raw materials to finished goods. It shows:

- **Decision Points:** Material availability check, functional test pass/fail
- **Sequential Stages:** Slip preparation → pressure casting → demolding → fettling → glazing → tunnel kiln firing → finishing with functional testing
- **Fork/Join:** Quality classification into Pass, Rejects after functional testing
- **Bottleneck Identification:** Tunnel kiln firing (24h cycle) highlighted as the severe constraining resource

Production Process Activity Diagram - Commode Manufacturing



3.2.2 Simulation Execution Activity Diagram

This diagram illustrates the high-level simulation execution flow:

- **Initialization:** Command-line argument parsing, scenario selection, environment setup
- **Concurrent Processes:** Fork showing parallel SimPy processes (production stages, suppliers, demand, fulfillment)
- **Iteration:** Day-by-day simulation advancement with progress updates
- **Post-Processing:** KPI computation, table generation, chart rendering

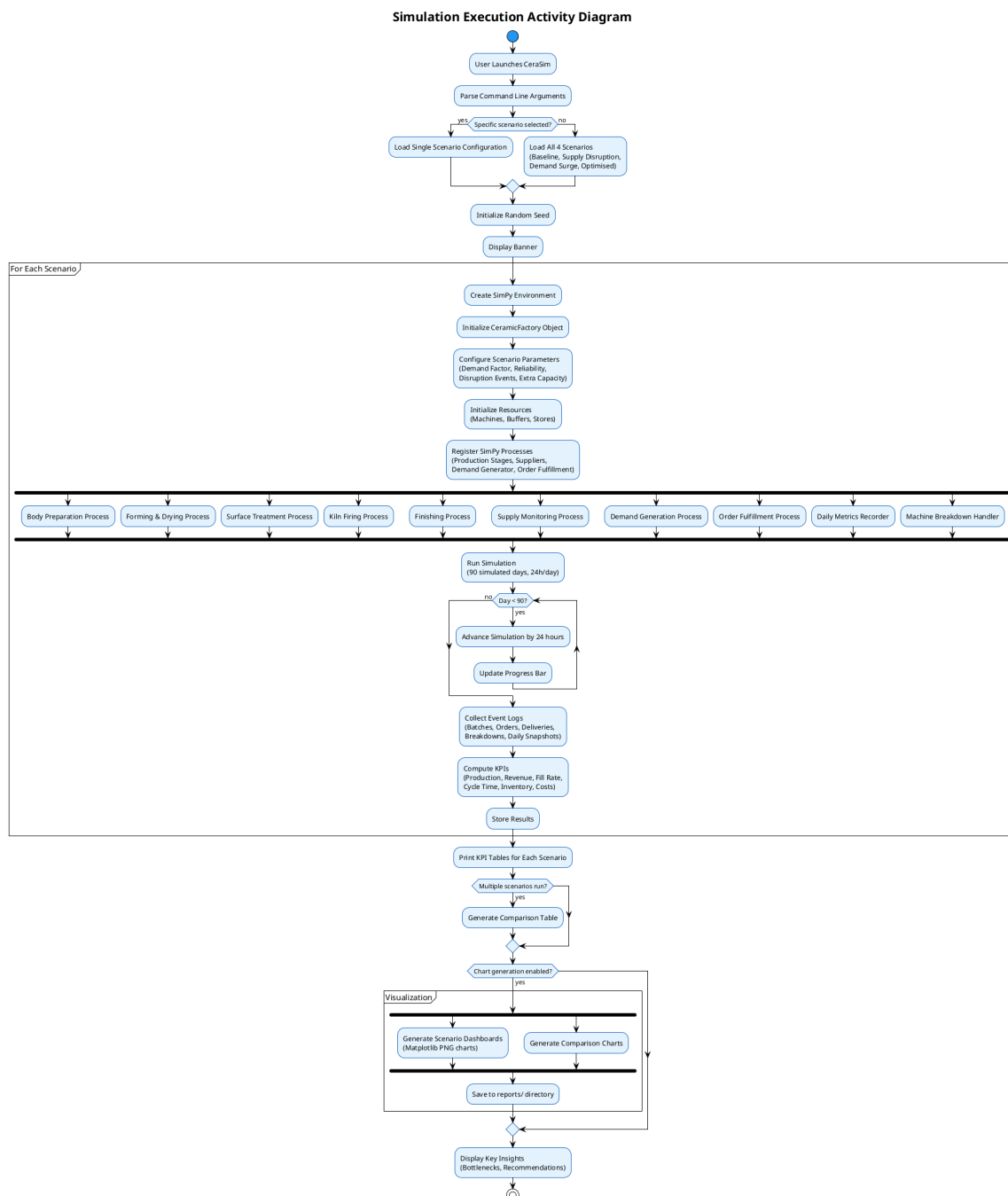


Figure 3: Activity Diagram - Simulation Execution Workflow

3.3 Sequence Diagrams

Sequence diagrams show **object interactions over time**, emphasizing message passing between objects.

3.3.1 Production Batch Processing Sequence

This diagram traces a single `ProductionBatch` object through the entire production pipeline:

- **Lifelines:** `SimPy Environment`, `Factory`, `ProductionBatch`, and production stage machines
- **Resource Requests:** `request()` and `release()` patterns for machine allocation
- **State Updates:** Timestamp recording at each stage completion
- **Conditional Flow:** Glazing stage bypassed for unglazed products
- **Final State:** Batch completion with quality grades, cycle time calculation, metrics logging

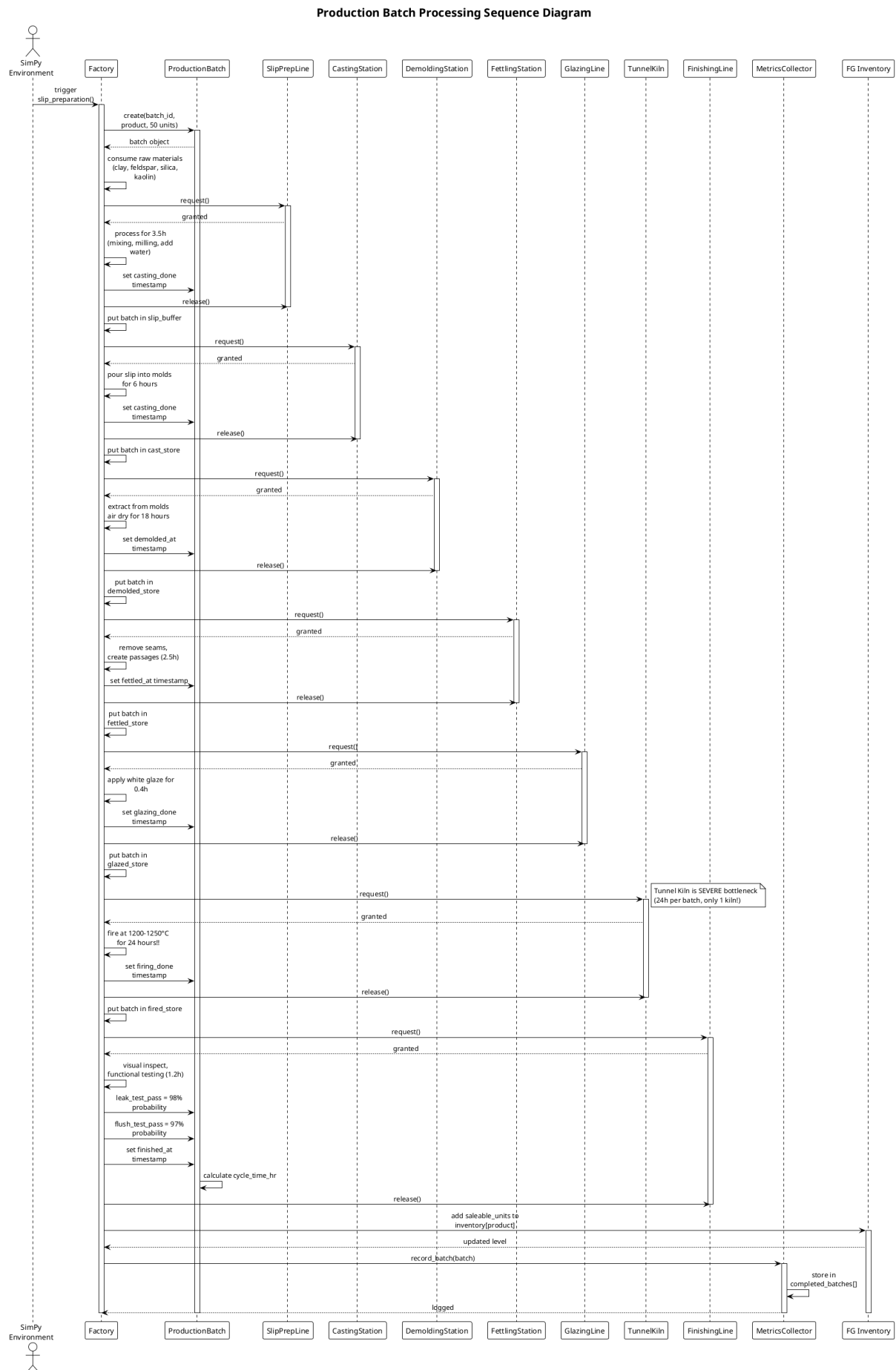


Figure 4: Sequence Diagram - Production Batch Processing

3.3.2 Customer Order Fulfillment Sequence

This diagram shows the order lifecycle from creation to fulfillment:

- **Order Creation:** DemandGenerator creates CustomerOrder and enqueues it
- **Inventory Check:** Factory queries finished goods inventory
- **Fulfillment Paths:** Full, partial, or zero fulfillment based on availability
- **Metrics Recording:** Fill rates, stockout events, lost sales tracking

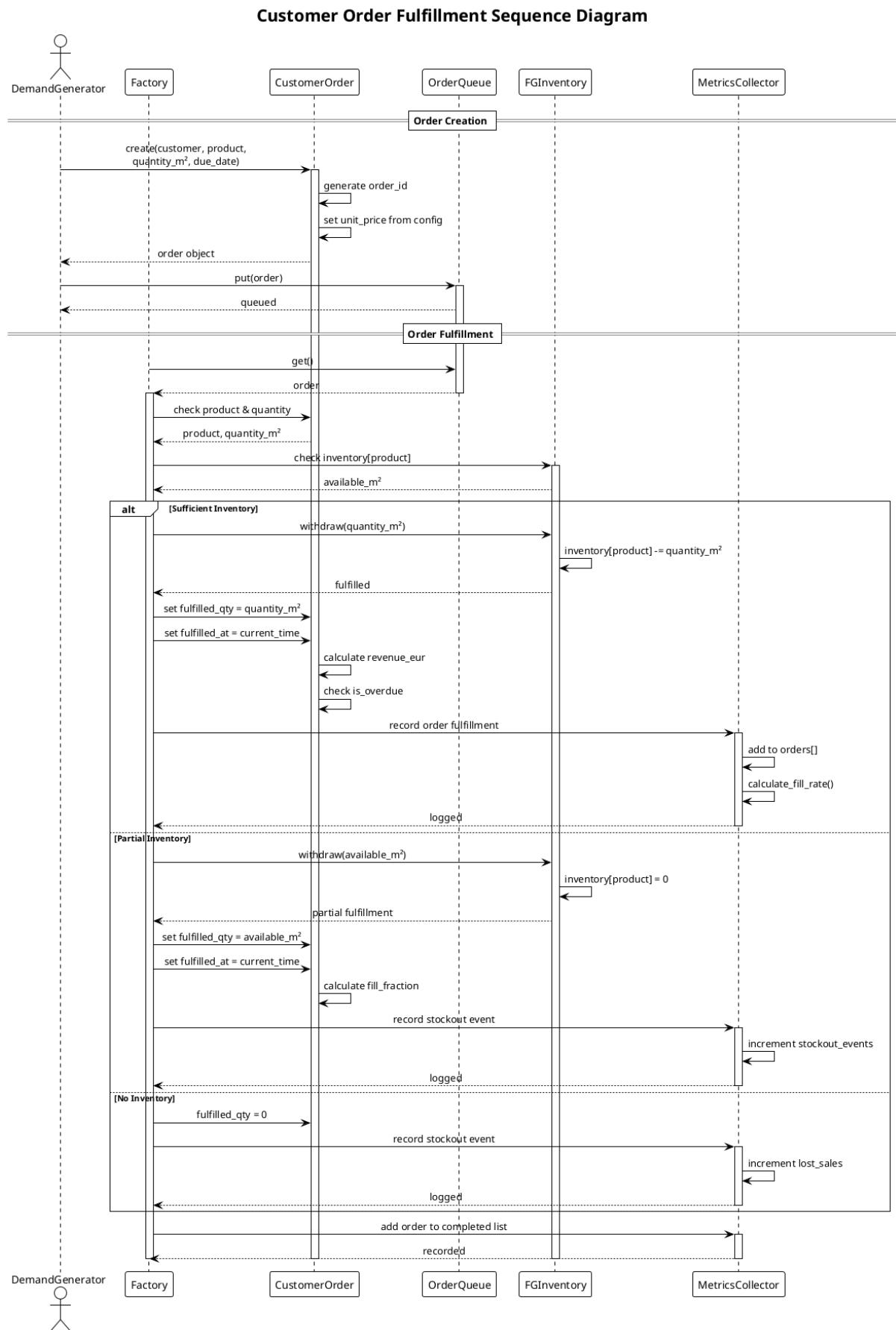


Figure 5: Sequence Diagram - Customer Order Fulfillment

3.4 State Machine Diagrams

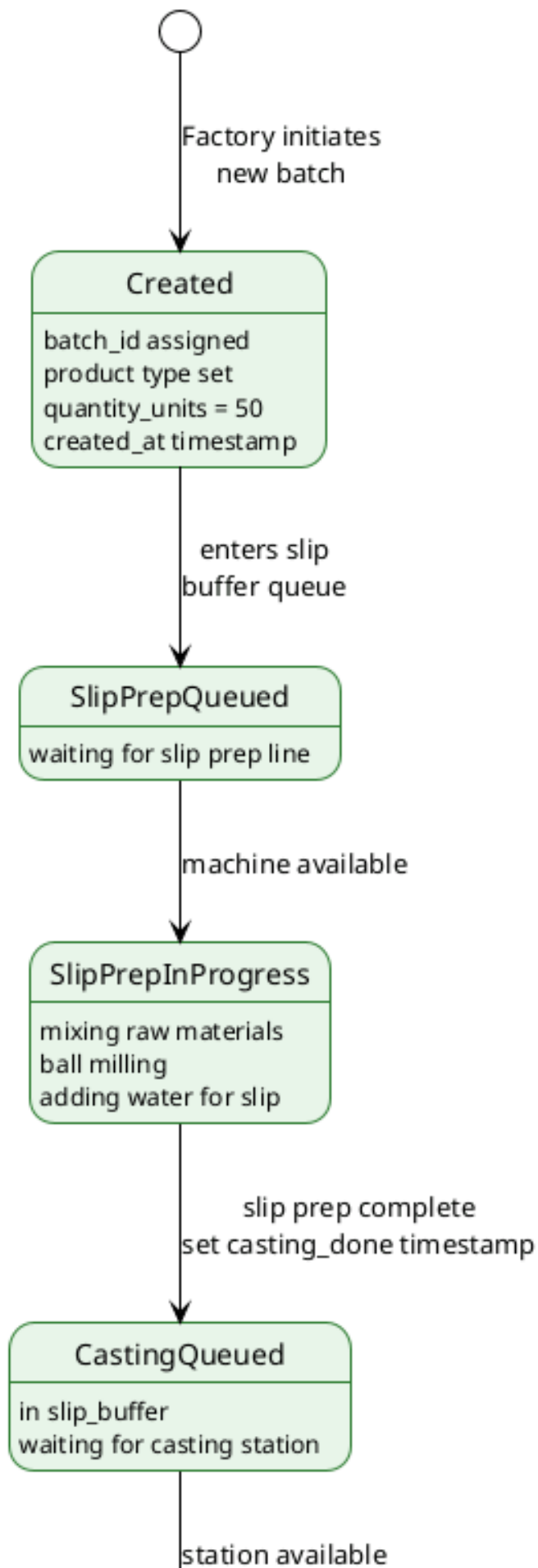
State machine diagrams model **object lifecycle** showing states and transitions triggered by events.

3.4.1 ProductionBatch State Machine

This diagram shows how a ProductionBatch transitions through production states:

- **States:** Created, SlipPrepQueued, SlipPrepInProgress, CastingQueued, CastingInProgress, DemoldingQueued, DemoldingInProgress, FettlingQueued, FettlingInProgress, GlazingQueued, GlazingInProgress, FiringQueued, FiringInProgress, FinishingQueued, FunctionalTesting, Completed
- **Transitions:** Triggered by machine availability and stage completion
- **Testing Transition:** Leak test and flush test determine pass/fail
- **Annotations:** Bottleneck state (FiringQueued - 24h cycle) and cycle time notes

ProductionBatch State Machine Diagram



3.4.2 CustomerOrder State Machine

This diagram models the order lifecycle:

- **States:** Created, Queued, Processing, FullyFulfilled, PartiallyFulfilled, Unfulfilled, On-Time, Overdue, Closed
- **Branching:** Three fulfillment outcomes based on inventory availability
- **Time-Based Transition:** OnTime vs. Overdue based on due date comparison
- **Metrics Impact:** Annotated effects on fill rate, stockouts, lost sales

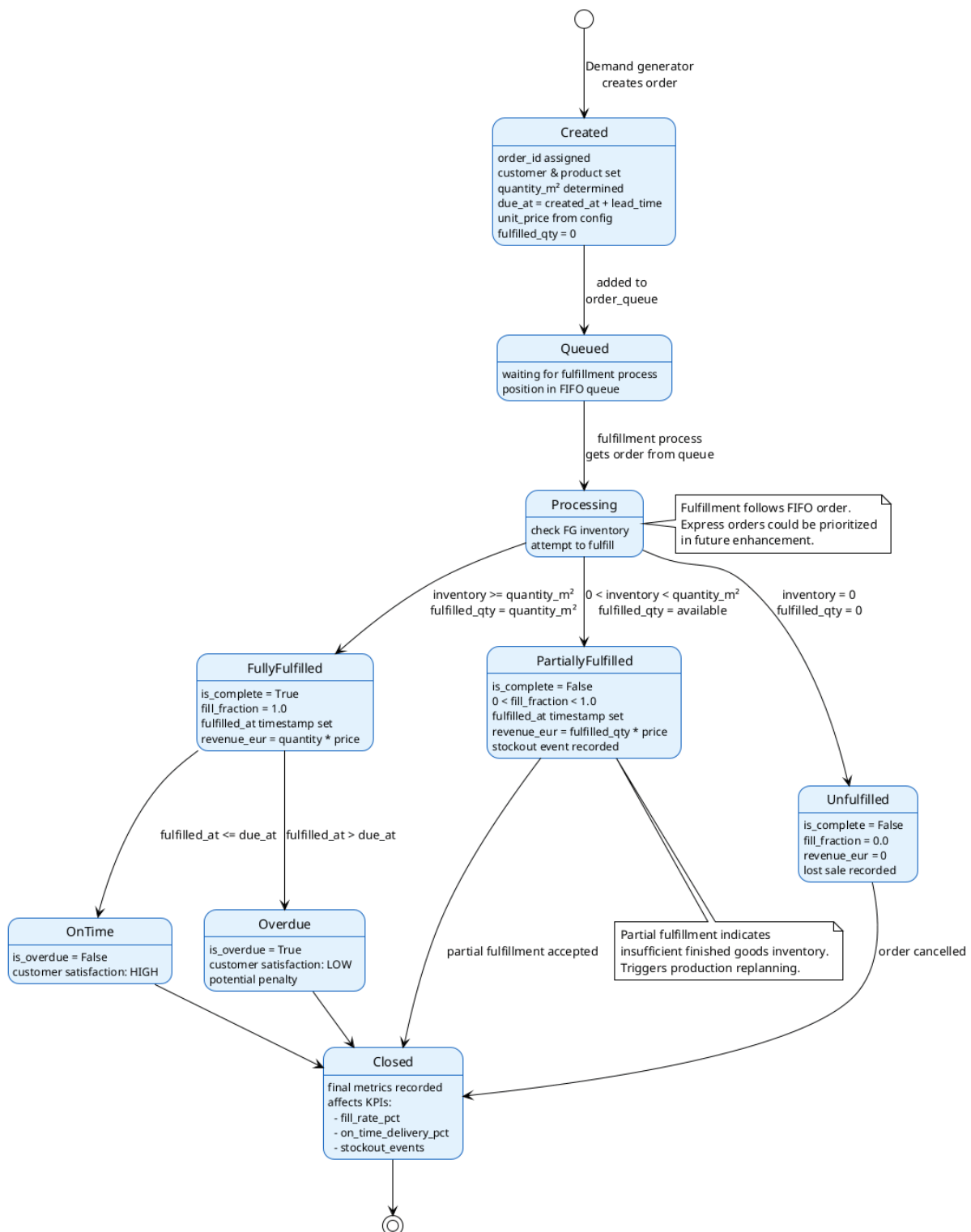
CustomerOrder State Machine Diagram

Figure 7: State Machine Diagram - CustomerOrder Lifecycle

3.4.3 Machine Resource State Machine

This diagram shows machine operational states:

- **States:** Idle, Requested, Processing, BreakdownDuringOperation, BreakdownIdle,

UnderRepair

- **Failure Transitions:** Random failures from Idle or Processing states based on MTBF
- **Repair Process:** UnderRepair state with repair duration governed by MTTR
- **Resource Release:** Return to Idle state after processing completion or repair

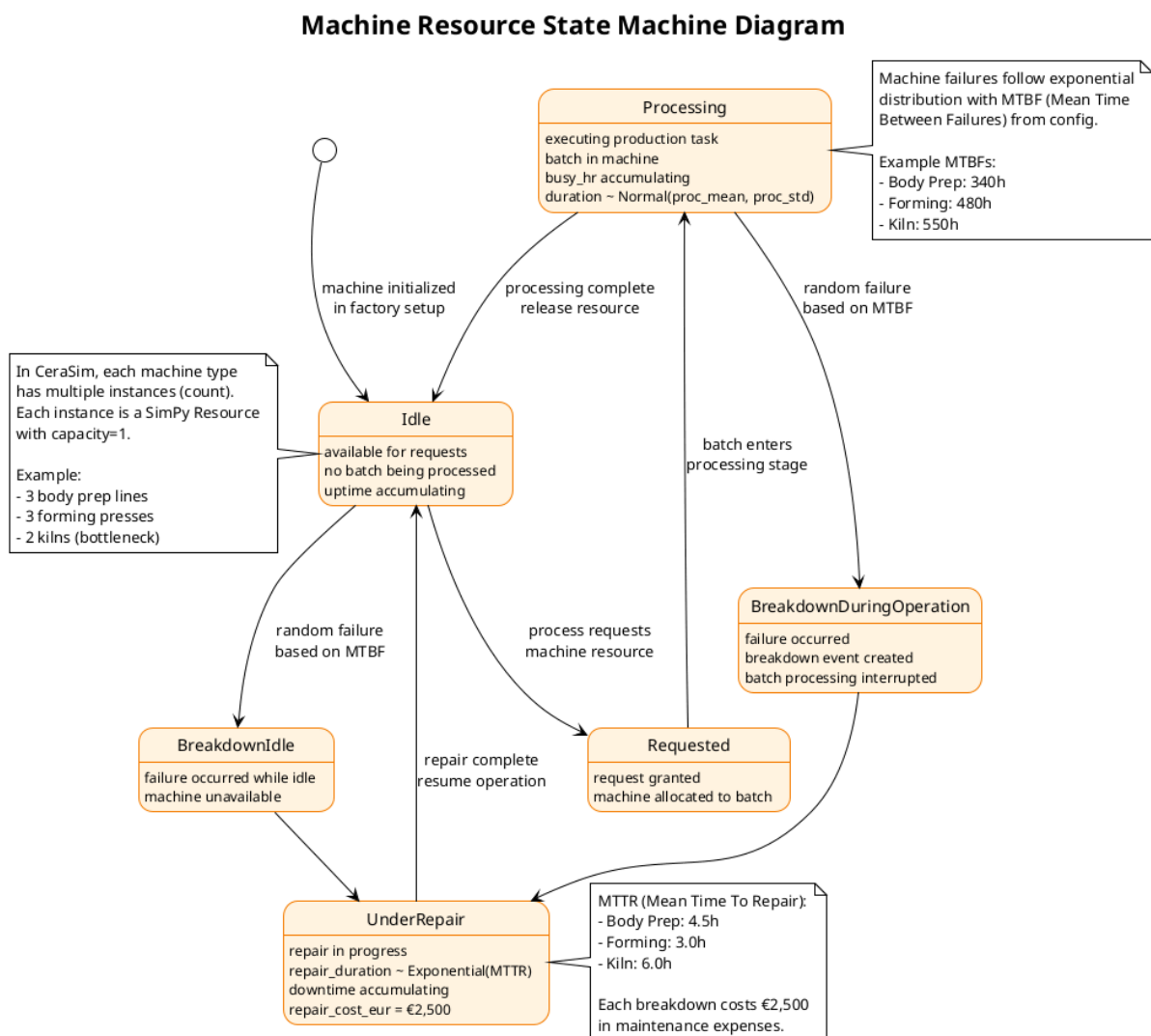


Figure 8: State Machine Diagram - Machine Resource States

3.5 Object-Oriented Design Principles Demonstrated

3.5.1 Encapsulation

Each class encapsulates related data and behavior:

- **ProductionBatch** hides internal state, exposes properties like `cycle_time_hr`, `saleable_units`, `leak_test_pass`, `flush_test_pass`
- **CustomerOrder** computes `is_complete`, `is_overdue`, `revenue_eur` from internal fields

3.5.2 Abstraction

High-level abstractions hide complexity:

- Users interact with scenarios ("baseline", "supply_disruption") without knowing SimPy internals
- `CeramicFactory.register_processes()` hides intricate process registration logic

3.5.3 Modularity

Clear separation of concerns:

- `config.py` - Parameters
- `models.py` - Data structures
- `factory.py` - Simulation logic
- `metrics.py` - Analysis
- `reports.py` - Presentation

3.5.4 Composition

`CeramicFactory` composes multiple resources:

- Machines (SimPy Resources)
- Buffers (SimPy Containers)
- Queues (SimPy Stores)
- MetricsCollector instance

3.5.5 Polymorphism

- All production stages follow the same pattern: request resource → process → release
- SimPy processes are uniform generator functions, allowing uniform registration

4 References

4.1 Textbooks and Academic Resources

1. Grady Booch, Robert A. Maksimchuk, Michael W. Engle, et al. (2007). *Object-Oriented Analysis and Design with Applications*, 3rd Edition. Addison-Wesley Professional.
2. Martin Fowler (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd Edition. Addison-Wesley Professional.
3. Averill M. Law (2014). *Simulation Modeling and Analysis*, 5th Edition. McGraw-Hill Education.
4. Banks, J., Carson, J.S., Nelson, B.L., Nicol, D.M. (2009). *Discrete-Event System Simulation*, 5th Edition. Pearson.

4.2 Software Documentation

1. SimPy Documentation. *Discrete event simulation for Python*.
<https://simpy.readthedocs.io/>
2. Python Software Foundation. *Python 3 Documentation*.
<https://docs.python.org/3/>
3. Matplotlib Development Team. *Matplotlib: Visualization with Python*.
<https://matplotlib.org/>
4. Rich Library Documentation. *Rich - Python library for rich text in the terminal*.
<https://rich.readthedocs.io/>

4.3 UML and Design Resources

1. Object Management Group (OMG). *Unified Modeling Language Specification*, Version 2.5.1.
<https://www.omg.org/spec/UML/>
2. PlantUML. *Open-source tool for creating UML diagrams from plain text*.
<https://plantuml.com/>
3. Lucidchart. *UML Diagram Tutorial*.
<https://www.lucidchart.com/pages/uml>
4. Visual Paradigm. *UML Unified Modeling Language Guide*.
<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/>

4.4 Ceramic Industry Domain Knowledge

1. Ceramic Industry Magazine. *Tile Manufacturing Process Overview*.
<https://www.ceramicindustry.com/>
2. European Ceramic Tile Manufacturers Federation. *Production Process Documentation*.
<https://www.cerame-unie.eu/>

4.5 Supply Chain and Operations Management

1. Chopra, S., Meindl, P. (2015). *Supply Chain Management: Strategy, Planning, and Operation*, 6th Edition. Pearson.
2. Hopp, W.J., Spearman, M.L. (2011). *Factory Physics*, 3rd Edition. Waveland Press.

4.6 Online Learning Resources

1. GeeksforGeeks. *Object Oriented Programming in Python*.
<https://www.geeksforgeeks.org/python-oops-concepts/>
2. Real Python. *Object-Oriented Programming in Python 3*.
<https://realpython.com/python3-object-oriented-programming/>

3. Coursera. *Object-Oriented Design Course by University of Alberta*.
<https://www.coursera.org/learn/object-oriented-design>

4.7 Tools Used

Tool	Version	Purpose
PlantUML	Latest	UML diagram generation from text specifications
LaTeX	TeXLive 2023+	Professional document typesetting
Python	3.11	Application development and simulation implementation
SimPy	4.1.0	Discrete-event simulation framework
Matplotlib	3.10.8	Data visualization and charting
Git	2.x	Version control system

Table 10: Software Tools Used in Project Development and Documentation

Appendix A: Class Diagram

While not required by the assignment, we include a class diagram showing the static structure of the system for completeness.

CeraSim Class Structure

ProductionBatch

- + batch_id: str
- + product: str
- + quantity_m2: float
- + created_at: float
- + forming_done: Optional[float]
- + glazing_done: Optional[float]
- + firing_done: Optional[float]
- + finished_at: Optional[float]
- + grade_a_m2: float
- + grade_b_m2: float
- + reject_m2: float
- + cycle_time_hr: Optional[float]
- + saleable_m2: float

CustomerOrder

- + order_id: str
- + customer: str
- + product: str
- + quantity_m2: float
- + is_express: bool
- + created_at: float
- + due_at: float
- + fulfilled_qty: float
- + is_complete: bool
- + is_overdue: bool
- + revenue_eur: float

CeramicFactory

- + env: simpy.Environment
- + scenario_id: str
- + machines: Dict[str, simpy.Resource]
- + raw_materials: Dict[str, simpy.Container]