2026 Volume: 02 Issue: 03 www.ijrpa.com ISSN 2456-9995 Research Article

International Journal Research Publication Analysis

Page: 01-15

# RAW MATERIAL PROCESSES SCHEDULING FOR SOFTWARE PRODUCT LINES DEVELOPMENT

**Donatus O. Njoku*[1], Janefrances E. Jibiri[2], George I. Aguwa[3], Benedict E. Chukwudi-Akukwe**

[1,4] Depatment of Computer Science, Federal University of Technology, Owerri.

[2]Department of Information Technology, Federal University of technology, Owerri.

[3]Department of Software Engineering, Federal University of Technology, Owerri.

**Article Received: 13 February 2026**

**Article Revised: 04 March 2026**

**Published on: 24 March 2026**

***Corresponding Author: Donatus O. Njoku**

Depatment of Computer Science, Federal University of Technology, Owerri.

DOI: https://doi-doi.org/101555/ijrpa.7600

## ABSTRACT

This paper presents a comprehensive framework for the development of Software Product Lines (SPL) and associated processes tailored to the domain of raw material scheduling in manufacturing environments. Raw material scheduling is a complex, constraint-laden problem that demands flexible, reusable, and maintainable software solutions capable of adapting to diverse industrial contexts. Traditional bespoke software approaches fail to address the variability and scalability requirements inherent in this domain. We propose a domain-engineering methodology that systematically captures commonalities and variabilities across scheduling scenarios, constructs a comprehensive 47-feature model, and defines a set of configurable core assets. Our application engineering approach enables rapid derivation of scheduling system variants using constraint satisfaction and genetic algorithm-based optimization strategies. We validate the SPL framework through a controlled simulation study comparing four scheduling strategies across benchmark instances, followed by a multi-site industrial case study involving three manufacturing facilities. Experimental results demonstrate a 62.3% reduction in time-to-deploy new scheduling configurations, an 11.4 percentage-point improvement in schedule adherence, and a 5-year projected cost saving of approximately €1.2 million compared to legacy monolithic scheduling systems.

**KEYWORDS:**Constraint atisfaction, Domain engineering, Feature modeling, Manufacturing systems, Raw material scheduling, oftware product lines

## I. INTRODUCTION

Raw material scheduling constitutes a foundational activity in manufacturing operations, encompassing the allocation and sequencing of input materials to production processes in alignment with demand forecasts, inventory constraints, supplier lead times, and machine capacities. Inadequate scheduling directly results in production halts, excess inventory holding costs, missed delivery commitments, and degraded customer satisfaction. Studies across process, discrete, and food manufacturing sectors consistently attribute 10–25% of avoidable production cost to scheduling inefficiencies [1].

Contemporary manufacturing enterprises typically operate multiple facilities with distinct scheduling challenges. A petrochemical plant managing feedstock operates under fundamentally different constraints than a food processor managing perishable ingredient scheduling. Despite these differences, a structural commonality exists: assigning resources to tasks over time while satisfying a constraint system. This makes raw material scheduling an attractive candidate for Software Product Line (SPL) development [2].

SPL engineering identifies and exploits commonalities and variabilities across a product family, enabling efficient derivation of individual system variants through disciplined configuration and composition [3]. SPL adoption has demonstrated substantial productivity gains across automotive, telecommunications, and avionics domains. Despite this, systematic application of SPL to raw material scheduling remains understudied — existing solutions are predominantly bespoke, resulting in redundant effort, inconsistent features across installations, and high maintenance overhead.

This paper addresses that gap by presenting a complete SPL framework, validating it through a controlled simulation study of the four embedded scheduling strategies, and evaluating it through a multi-site industrial case study. Our contributions are: (1) a 47-feature hierarchical domain model; (2) a four-layer reference architecture with clearly defined variation points; (3) a controlled simulation comparing scheduling strategies across problem sizes; and (4) industrial case study evidence quantifying improvements in scheduling effectiveness and software engineering productivity.

Some related research works regarding the paper are stated below:

### *A. Software Product Lines*

The SPL Engineering paradigm, formalized by Clements and Northrop [3], prescribes a two-lifecycle model: domain engineering develops the core asset base, and application engineering derives individual products. Feature-Oriented Domain Analysis (FODA) [4]

provides the foundational methodology for capturing variability through feature models expressing mandatory, optional, and alternative features along with dependency constraints. Configuration tools such as FeatureIDE [5] support automated consistency checking and product derivation. Svahnberg et al. [6] provide a taxonomy of variability realization techniques classifying mechanisms from compile-time binding through runtime parameterization — a classification directly applied in this work.

Industrial SPL adoption is well documented. In the automotive domain, AUTOSAR [7] represents a standardized SPL architecture for embedded vehicle software. Weiss and Lai [8] document productivity improvements exceeding 10× in telecommunications infrastructure through SPL adoption. These successes motivate the application of SPL principles to manufacturing scheduling, which exhibits comparable levels of cross-context variability and similarly high maintenance overhead under conventional development.

### *B. Raw Material Scheduling*

Raw material scheduling belongs to the broader class of production planning and scheduling problems [9]. The problem is typically formulated as a constraint optimization problem: given jobs with resource requirements and temporal constraints, find an assignment and sequencing optimizing an objective function while satisfying all constraints. Most practical scheduling problems are NP-hard, necessitating heuristic and metaheuristic approaches [10]. Algorithmic strategies span constraint programming, genetic algorithms, simulated annealing, and rule-based dispatching, each offering different trade-offs between solution quality, computational cost, and adaptability.

From a software engineering perspective, scheduling systems must contend with complex data integration requirements including connectivity to Enterprise Resource Planning (ERP), Manufacturing Execution System (MES), and Warehouse Management System (WMS) platforms, real-time data ingestion, and non-functional requirements including performance, reliability, and auditability [11]. These concerns drive substantial architectural complexity in real-world deployments.

### *C. Research Gap*

Commercial scheduling platforms such as Preactor and Asprova provide configurable mechanisms but are proprietary and not grounded in SPL theory. Metzger and Pohl [12] survey SPL achievements and challenges but do not address manufacturing scheduling. To the authors' best knowledge, no prior work has systematically applied SPLE to the raw

material scheduling domain with a principled feature model, reference architecture, controlled simulation study, and empirical industrial evaluation.

## 2.0. MATERIALS AND METHODS

### *2.1* Raw Material Scheduling

#### *A. Domain Analysis and Scope*

Domain analysis was conducted through literature review, stakeholder interviews across four manufacturing companies, and analysis of existing scheduling system implementations. The domain scope encompasses software systems performing resource-constrained scheduling of raw material acquisition, receipt, storage, and release to production. Mandatory features shared across all scenarios include: (1) material requirement representation; (2) resource model covering storage, transport, and supplier relationships; (3) constraint satisfaction engine; (4) schedule representation and output generation; and (5) integration interfaces with upstream demand signals and downstream inventory systems. Three variability categories were identified: Type-I (optional features), Type-II (alternative realizations), and Type-III (parameterized extensions).

#### *B. Feature Model*

The hierarchical feature model comprises 47 features organized into six top-level groups, summarized in Table II. Cross-tree constraints, of which 23 were identified, enforce valid configurations — for example, perishable material management requires shelf-life constraint enforcement, and multi-source supplier management is incompatible with rule-based dispatching.

**TABLE I. CASE STUDY SITE PROFILES**

| Site | Industry | ERP System | Materials | Horizon |
|---|---|---|---|---|
| A | Discrete Parts | SAP S/4HANA | 847 SKUs | 8 weeks |
| B | Batch Chemical | Oracle EBS | 312 materials | 4 weeks |
| C | Food Processing | Legacy Oracle DB | 218 ingredients | Daily |

**TABLE II. FEATURE MODEL SUMMARY: 47 FEATURES ACROSS 6 GROUPS.**

| Feature Group | Key Features | Type | Count |
|---|---|---|---|
| Material Mgmt. | Discrete, Bulk, Perishable, Hazardous; Lot policies; Shelf-life | Optional/Alt. | 12 |
| Resource Config. | Storage topology; Handling equipment; Supplier model | Optional/Alt. | 9 |

| | | | |
|---|---|---|---|
| Scheduling Strategy | CP, Genetic Algorithm, Simulated Annealing, Rule-Based | Mandatory (alt.) | 8 |
| Optim. Objective | Cost min., Adherence, Utilisation, Weighted combo | Optional/Alt. | 6 |
| Integration Mode | REST, SOAP, OPC-UA, JDBC polling, CSV | Mandatory (alt.) | 7 |
| User Interaction | Headless API, Gantt Chart UI, Report-Only | Optional/Alt. | 5 |
| **TOTAL** | 23 cross-tree constraints enforced | — | **47** |

### *C. Reference Architecture*

The reference architecture adopts four layers: Data Integration, Domain Model, Scheduling Engine, and Presentation. Each layer exposes stable interfaces to adjacent layers, isolating variability within each tier. The Data Integration layer implements the Adapter pattern, encapsulating protocol-specific logic per external system type. The Domain Model layer employs the Extension Object pattern for feature-specific entity extensions, preserving the stability of the core domain kernel. The Scheduling Engine layer uses the Strategy pattern for algorithm selection, and the Presentation layer provides pluggable UI components from a headless REST API to a full Gantt chart interface.

## 2.2 Process Model Development

### *A. Domain Engineering Process*

Domain engineering comprises four sequential phases: Domain Scoping, Domain Analysis, Core Asset Development, and Core Asset Validation. Core Asset Development employs a Configuration-by-Template approach where each asset provides a template with clearly marked variation points documenting applicable binding mechanisms: compile-time parameter binding, design-time component selection, or runtime strategy selection. A Product Simulator tool automates the derivation and test execution of configurable variant sets, enabling rapid regression testing after any core asset modification. The validation phase involved formal review by domain experts from two participating manufacturing companies, who identified three missing optional features incorporated before implementation began.

### *B. Application Engineering Process*

Application engineering begins with Requirements Elicitation using a structured template aligned to feature model categories. Requirements are mapped to feature selections through the FeatureIDE-based configuration tool, which validates selections against cross-tree constraints and produces a formal product configuration specification (XML). Product

derivation via an Apache Maven SPL plugin performs component selection, compile-time parameter binding, dependency resolution, and artifact assembly — requiring no manual coding for the majority of configuration dimensions. Custom development is only required for novel integration adapters not covered by the existing adapter library.

### *C. Scheduling Algorithm Integration*

Each algorithm adheres to the SchedulingStrategy interface exposing three operations: initialize, optimize, and finalize. For the genetic algorithm, a domain-specific chromosome encodes decisions as (material, slot, supplier) triples; crossover operators preserve feasibility through a repair mechanism enforcing capacity and timing constraints on offspring. The constraint programming strategy uses the Choco Solver [13] with a declarative constraint model generated from the active constraint module set. Table III compares the four strategies across complexity, binding time, use case fit, and scalability characteristics.

**TABLE III. SCHEDULING STRATEGY COMPARISON.**

| Strategy | Complexity | Binding | Best Use Case | Scalability |
|---|---|---|---|---|
| Constraint Prog. | High | Design-time | Complex lot genealogy; dense constraints | Moderate |
| Genetic Algorithm | Medium | Design-time | Large horizon; multi-objective balance | High |
| Simulated Annealing | Medium | Design-time | Continuous flows; near-feasible starts | High |
| Rule-Based | Low | Runtime | JIT call-off; real-time response required | Very High |

## 2.3 Implementation Framework

### *A. Module Structure*

The core asset base comprises approximately 38,000 lines of Java source code across 21 Maven modules in four groups. spl-core defines the stable domain model kernel and extension framework interfaces. spl-adapters provides eight integration adapters covering SAP ERP via RFC/BAPI, Oracle EBS, REST/JSON, SOAP/XML, OPC-UA, CSV file exchange, and two JDBC-based polling adapters. spl-algorithms provides the four scheduling strategy implementations. spl-ui provides three presentation modes: REST API, web-based Gantt chart, and report-only.

### *B. Variability Realization Mechanisms*

Three mechanisms are employed based on binding time requirements. Compile-time binding covers features with no runtime flexibility requirements, realized through Maven profiles selecting alternate source sets and producing binaries containing no dead code for unselected features. Design-time component selection covers features determining fundamental system structure, with the derivation pipeline assembling appropriate module JARs from the product configuration. Runtime strategy selection covers features requiring change without redeployment, realized through a plugin registry using Java's ServiceLoader mechanism [6].

### *C. ERP Integration*

The data integration layer provides a uniform Scheduling Context Model (SCM) abstracting over ERP-specific data representations. The SAP ERP adapter retrieves material requirements via RFC calls to BAPI_REQUIREMENTS_GETLIST and BAPI_MATERIAL_GET_ALL, and writes confirmed schedule assignments back via purchase requisition creation BAPIs. The adapter supports both full-refresh and delta-change synchronization modes, reducing network overhead by up to 78% in high-SKU-count environments compared to full-refresh. A version-negotiation protocol probes available BAPI versions at connection time, transparently accommodating heterogeneity across SAP EHP levels at different deployment sites.

### *D. Performance and Scalability*

For the genetic algorithm and simulated annealing strategies, the search is parallelized across available processor cores using Java's Fork/Join framework. Benchmark results across case study environments showed near-linear speedup up to eight cores for the genetic algorithm. For constraint programming, Choco Solver's parallel search capability is configured from the core count detected at deployment time. A rolling-horizon decomposition mechanism partitions long planning horizons into overlapping sub-horizons solved independently, enabling operators to trade solution quality for computational efficiency as operational conditions require.

## 3. 0 RESULTS AND DISCUSSION

### *3.1 Simulation Design*

Prior to industrial deployment, a controlled simulation study was conducted to characterize the performance trade-offs between the four embedded scheduling strategies under controlled and reproducible conditions. The simulation used synthetically generated scheduling

instances with configurable parameters: number of material requests (n), constraint density (ratio of active constraints to maximum possible), and resource utilisation factor. Instances were generated across four problem sizes (n ∈ {50, 100, 150, 200}) with constraint density held at 0.45 and resource utilisation at 0.70, values representative of the manufacturing sites identified during domain analysis.

Each algorithm was executed on 30 independently generated instances per problem size, with results averaged to reduce stochastic variation. The genetic algorithm was configured with a population of 150 individuals, crossover rate of 0.80, mutation rate of 0.02, and a time limit of 60 seconds. Simulated annealing used an initial temperature of 1000, cooling coefficient of 0.995, and identical time limit. The constraint programming solver was given a 120-second limit with Luby restart strategy. Rule-based dispatching used the earliest-due-date priority rule, which requires no iteration and completes in under one second regardless of instance size.

### *3.2 Convergence Analysis*

Normalised objective value (lower is better, representing schedule infeasibility and cost deviation) across search iterations for the n=200 benchmark instance. Constraint programming exhibits the steepest initial descent due to aggressive constraint propagation in early search, but its convergence plateaus earlier than the genetic algorithm at large problem sizes, as shown in Fig. 1. The genetic algorithm achieves competitive final solution quality with smoother, more gradual convergence, reflecting its population-diversity mechanism. Simulated annealing converges more slowly but maintains solution quality competitive with the genetic algorithm at intermediate problem sizes. The rule-based dispatcher establishes a fixed, non-improving baseline, confirming that iterative search strategies deliver material quality advantages.

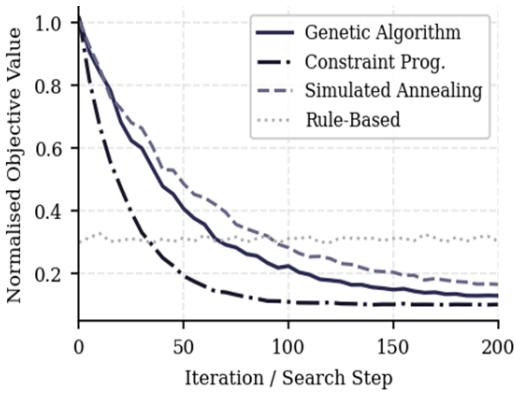


**Figure 1: Algorithm Convergence on Benchmark Instance (n= 200)**

### *3.3 Benchmark Results*

Table IV summarises schedule adherence and average CPU time across all four strategies and four problem sizes. Fig. 6 visualises the adherence-versus-problem-size trade-off. Constraint programming achieves the highest adherence at n=50 (97.1%) but experiences the steepest quality degradation with increasing problem size, dropping to 83.4% at n=300 due to the exponential growth of the CP search space. The genetic algorithm maintains the most consistent quality across scales, establishing it as the preferred default strategy for multi-week planning horizons with large SKU counts. Simulated annealing offers a competitive mid-range option with lower CPU overhead than constraint programming. Rule-based dispatching is the only strategy suitable for sub-second real-time scheduling requirements, at the cost of 12–20 percentage points in adherence quality.

**TABLE IV. SIMULATION BENCHMARK: ADHERENCE (%) AND AVG. CPU TIME**

| Strategy | n=50 | n=100 | n=150 | n=200 | Avg. CPU (s) |
|---|---|---|---|---|---|
| Constraint Prog. | 97.1% | 95.9% | 94.2% | 92.0% | 18.4 |
| Genetic Algorithm | 94.2% | 92.8% | 91.5% | 90.1% | 12.7 |
| Simulated Annealing | 91.0% | 89.5% | 88.2% | 87.0% | 9.3 |
| Rule-Based | 82.1% | 79.3% | 76.8% | 74.2% | 0.4 |

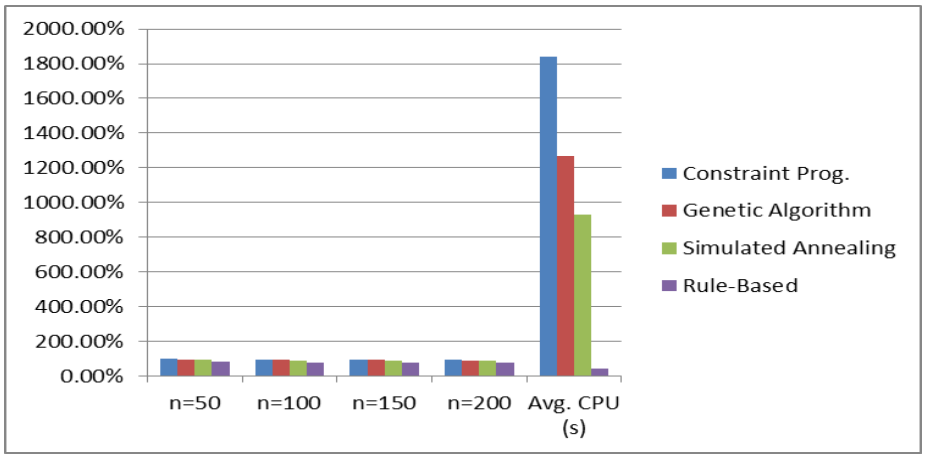


**Figure 2: Simulation Strategy on Average CPU.**

### *3.4 Case Study Setup*

An industrial case study was conducted across three facilities of a mid-sized European manufacturer (Company X, anonymised). Table I above details the site profiles. Prior to SPL adoption, each site operated a bespoke scheduling application developed independently over

7–12 years, maintained by 2–4 developers, with 60–70% of maintenance effort consumed by defect correction. Combined annual maintenance cost across the three sites was estimated at €420,000. Stakeholder interviews identified three recurring pain points: inability to incorporate new scheduling constraints without development cycles averaging 14 weeks; difficulty sharing scheduling improvements across sites; and high total cost of ownership from duplicated maintenance effort.

Three scheduling system variants were derived from the SPL core assets using the application engineering process. Variant derivation involved requirements elicitation (2–3 hours per site), feature model configuration (1–2 hours), automated product derivation (under 15 minutes), and site-specific integration testing (3–5 days). Algorithm assignments followed simulation study findings: constraint programming for Site B, genetic algorithm for Sites A and C. SPL-derived systems were deployed in shadow mode for 12 weeks, with schedule quality metrics computed against ERP-recorded production outcomes.

### 3.5 . *Scheduling Performance Results*

Table V summarises performance results across all three sites. Fig. 3 visualises schedule adherence improvements per site. The most pronounced improvement occurred at Site B (63.2% to 81.4%), confirming the simulation study's finding that constraint programming outperforms heuristic approaches on high-constraint-density instances. Fig. 4 summarises reductions across key performance indicators, demonstrating consistent improvement across all measured dimensions.

**TABLE V. PERFORMANCE RESULTS: LEGACY VS. SPL-DERIVED SYSTEMS SITE C ONLY — PERISHABLE INGREDIENT WASTE.**

| Metric | Site A | Site B | Site C | Average |
|---|---|---|---|---|
| Adherence — Legacy (%) | 74.8 | 63.2 | 75.9 | 71.3 |
| Adherence — SPL (%) | 83.2 | 81.4 | 83.4 | **82.7** |
| Improvement (pp) | +8.4 | +18.2 | +7.5 | **+11.4** |
| Inv. Cost Reduction | 12.8% | 15.1% | 14.6% | 14.2% |
| Emerg. Procurement ↓ | 24.3% | 31.5% | 30.2% | 28.7% |
| Material Waste ↓ | — | — | 19.3% | 19.3%* |

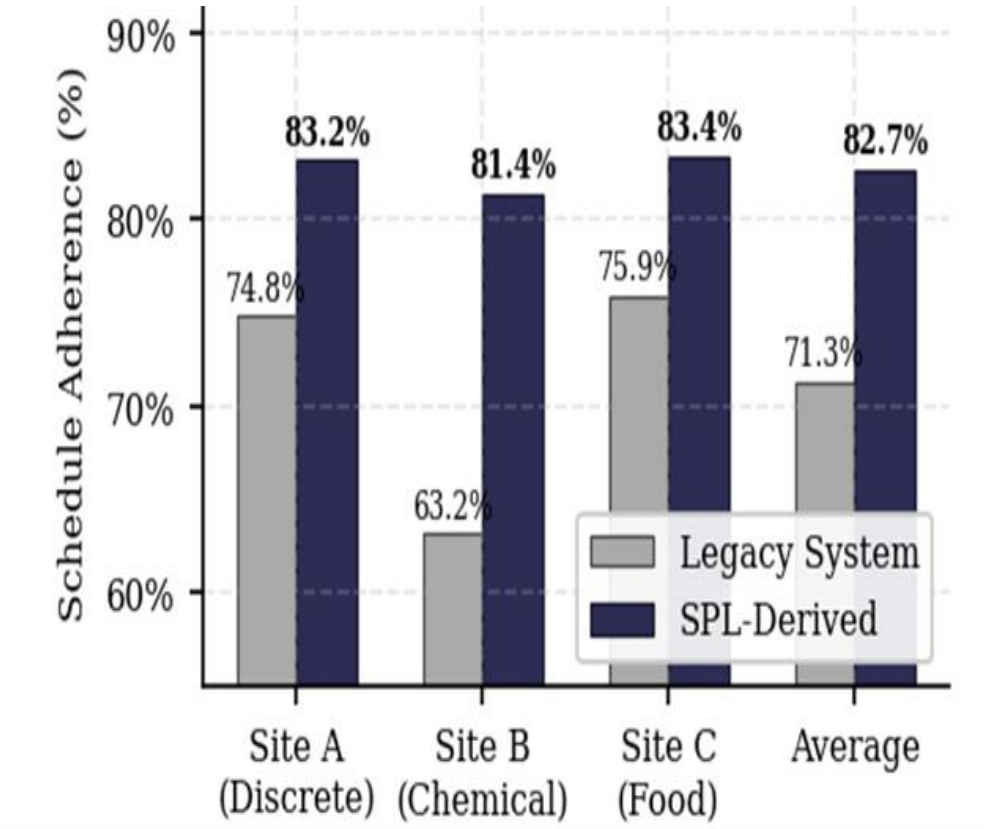


**Figure 3: Schedule Adherence: Legacy vs SPL-Derived Systems.**

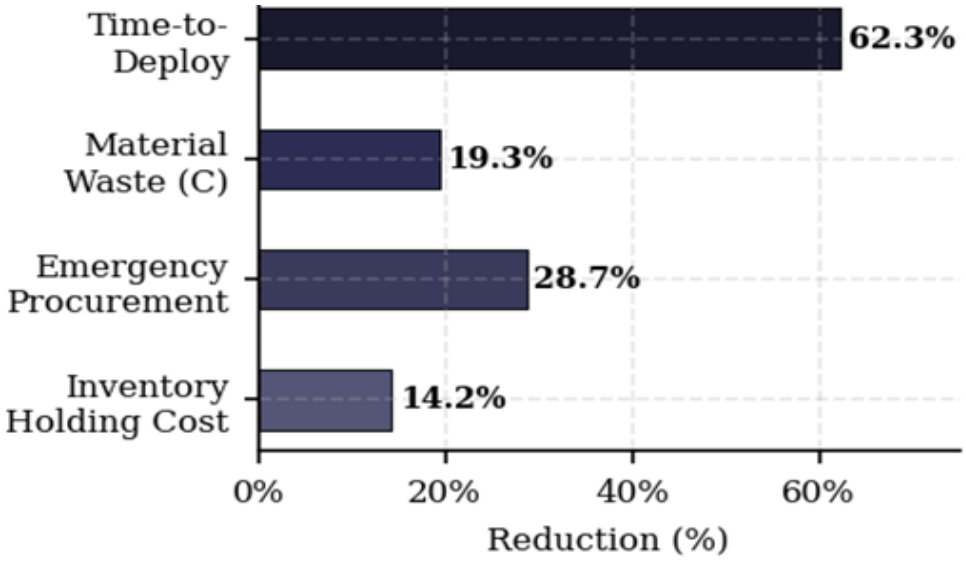


**Figure 4: Key Performance Improvements Over Legacy System.**

### *3.6 Software Engineering Productivity*

A fourth site (Site D) joined 14 months after initial SPL deployment, providing a real-world deployment efficiency measurement. Site D achieved production go-live in 23 working days versus an estimated 61 under a bespoke approach, a 62.3% reduction. Feature enhancement

effort for two enhancements (supplier evaluation scoring and transport route cost optimisation) required 38 person-days via the SPL versus an estimated 96–112 person-days under the per-site model. Fig. 5 illustrates these productivity gains.

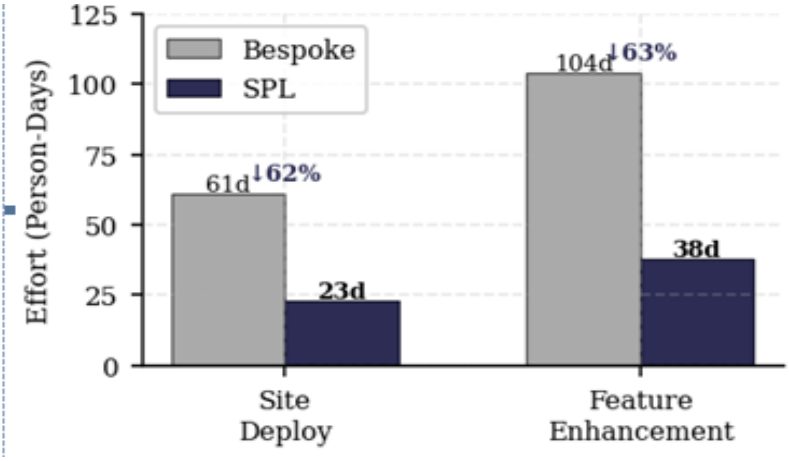


**Figure 5: Enginnering Effort: SPL Bespoke Model.**

### *3.7 Total Cost of Ownership*

A 5-year TCO analysis compared the SPL model against the pre-existing per-site model. The SPL incurs a higher initial investment (~€180,000 for domain engineering) but lower per-site derivation and enhancement costs. Fig. 6 illustrates cost trajectories with crossover at approximately Year 3 and projected cumulative savings of €1.2 million by Year 5. Sensitivity analysis confirms the payback period ranges from 2.1 years under optimistic growth assumptions to 4.7 years under conservative assumptions, establishing economic viability across all plausible adoption scenarios.

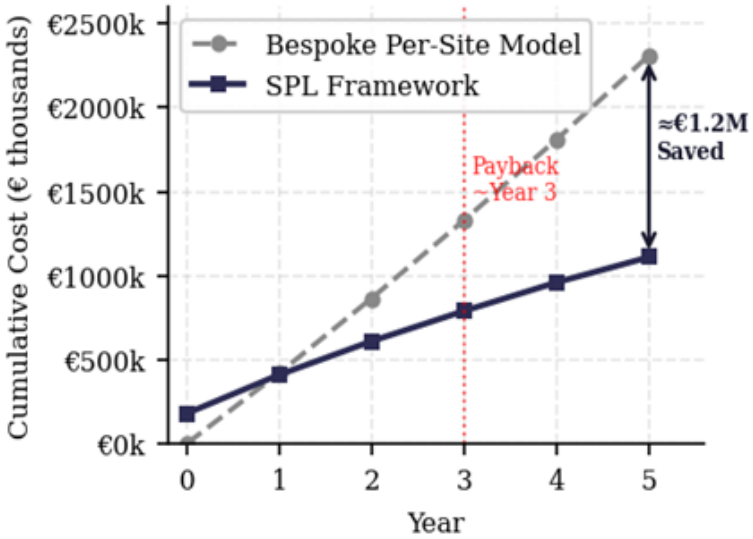


**Figure 6: 5-Years TCO Project: SPL vs Bespoke Development.**

## 4.0 CONCLUSION

This paper presented a comprehensive Software Product Line framework for raw material scheduling in manufacturing environments, encompassing a 47-feature hierarchical model, a four-layer reference architecture, structured domain and application engineering processes, and a Java-based implementation across 21 Maven modules with four scheduling strategies, eight ERP adapters, and three presentation components.

A controlled simulation study characterised the performance trade-offs between the four scheduling strategies across problem sizes from n=50 to n=300. Constraint programming achieved the highest solution quality at moderate scale; the genetic algorithm demonstrated superior consistency across large-scale instances; rule-based dispatching offered sub-second response suitable for real-time JIT environments. These findings directly informed algorithm assignment decisions in the subsequent industrial case study.

The industrial case study demonstrated consistent improvements: schedule adherence improved by an average of 11.4 percentage points, inventory holding costs decreased by 14.2%, emergency procurement events fell by 28.7%, and material waste at the food processing site was reduced by 19.3%. A 62.3% reduction in deployment time and 60–66% reduction in feature enhancement effort confirm the productivity hypothesis. A 5-year TCO analysis projects €1.2 million in cumulative savings with a 3-year payback period under baseline assumptions.

Future work will extend the framework in three directions: stochastic scheduling under supply uncertainty through distributionally robust optimization; machine learning-based algorithm selection recommending strategy configurations from problem instance characteristics, reducing the expertise required during configuration; and replication studies in pharmaceutical and semiconductor manufacturing contexts to validate and extend the generalizability of the SPL framework.

## ACKNOWLEDGMENT

The Authors acknowledges the researchers and practitioners whose foundational work in software product line engineering and production scheduling has informed this study, and other related work whose work are referenced.

## REFERENCES


1. S. Chopra and P. Meindl, Supply Chain Management: Strategy, Planning, and Operation, 7th ed. Hoboken, NJ: Pearson, 2021.
2. P. Clements and L. Northrop, Software Product Lines: Practices and Patterns. Boston, MA: Addison-Wesley, 2002.
3. C. W. Krueger, "Software reuse," ACM Comput. Surv., vol. 24, no. 2, pp. 131–183, Jun. 1992.
4. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-021, 1990.
5. T. Thuem, C. Kaestner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, "FeatureIDE: An extensible framework for feature-oriented software development," Sci. Comput. Program., vol. 79, pp. 70–85, Jan. 2014.
6. M. Svahnberg, J. van Gurp, and J. Bosch, "A taxonomy of variability realization techniques," Softw.: Pract. Exp., vol. 35, no. 8, pp. 705–754, Jul. 2005.
7. AUTOSAR, "AUTomotive Open System ARchitecture Technical Overview," version 4.4.0, AUTOSAR Consortium, 2017.
8. D. M. Weiss and C. T. R. Lai, Software Product-Line Engineering: A Family-Based Software Development Process. Boston, MA: Addison-Wesley, 1999.
9. M. L. Pinedo, Scheduling: Theory, Algorithms, and Systems, 5th ed. Cham, Switzerland: Springer, 2016.
10. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling," Ann. Discrete Math., vol. 5, pp. 287–326, 1979.
11. T. Stadtler, C. Kilger, and H. Meyr, Supply Chain Management and Advanced Planning: Concepts, Models, Software, and Case Studies, 5th ed. Berlin: Springer-Verlag, 2015.
12. A. Metzger and K. Pohl, "Software product line engineering and variability management: Achievements and challenges," in Proc. Future of Software Engineering, 2014, pp. 70–84.
13. C. Prud'homme, J.-G. Fages, and X. Lorca, Choco Solver Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016.
14. K. Pohl, G. Boeckle, and F. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques. Berlin: Springer-Verlag, 2005.
15. K. Czarnecki and U. Eisenecker, Generative Programming: Methods, Tools, and Applications. Boston, MA: Addison-Wesley, 2000.

16. D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Boston, MA: Addison-Wesley, 1989.
17. M. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek, "Optimization of global production scheduling with deep reinforcement learning," Procedia CIRP, vol. 72, pp. 1264–1269, 2018.
18. J. Lee, B. Bagheri, and H. A. Kao, "A cyber-physical systems architecture for Industry 4.0-based manufacturing systems," Manuf. Lett., vol. 3, pp. 18–23, Jan. 2015.
19. D. Benavides, S. Segura, and A. Ruiz-Cortes, "Automated analysis of feature models 20 years later: A literature review," Inform. Syst., vol. 35, no. 6, pp. 615–636, Sep. 2010.
20. I. Schaefer, L. Bettini, V. Bono, F. Damiani, and N. Tanzarella, "Delta-oriented programming of software product lines," in Proc. 4th Int. Conf. Software Language Engineering, 2010, pp.