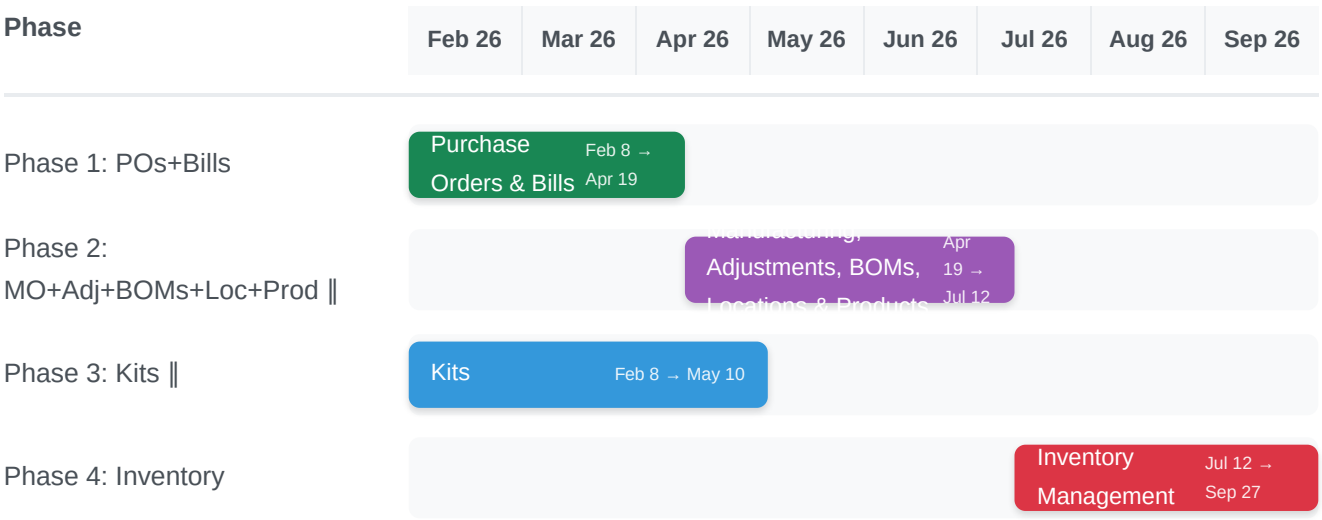
Odoo to SERP Migration Plan

Feb 8, 2026 → Sep 27, 2026 | Jack Kiefer

4	33	45	Sep 27
Phases	Weeks	Tasks	Target



‖ = Phases run in parallel

□ Data Migration Timeline

When Odoo data gets imported into SERP tables

Phase 1 — Week 4 — Mar 1

Import PO/Supplier Data

After schema design finalized, import PO and supplier data into SERP tables. Development continues against real data.

`purchase_order (1.8K)` `purchase_order_line (16K)` `res_partner (289 suppliers)` `product_supplierinfo (2.6K)` `account_move (vendor bills only)`

Phase 1 — Week 8 — Mar 29

Run Migration Scripts

Execute data cleanup, transformation, and validation scripts. Link SERP components to Odoo products.

`Data cleanup` `qty_received/qty_invoiced` `supplier-product links`

`bill line matching`

Phase 2 — Week 3 — May 3

Import MO/BOM/Product Data

Import manufacturing orders, BOMs, products, and locations after core schemas built.

`mrp_production (26K)` `mrp_bom (2K)` `mrp_bom_line (6.5K)` `product_product (7K)` `product_template (7K)` `mrp_unbuild (184)` `stock_location (2.7K)`

Phase 3 — Week 2 — Feb 15

Import Kit BOMs

Import phantom BOMs for kit explosion. Filter mrp_bom where type=phantom.

`mrp_bom (phantom)` `mrp_bom_line (kit components)`

Phase 4 — Week 4 — Aug 2

Test Inventory Import (Optional)

Practice run: export Odoo inventory snapshot to validate import scripts. Data will be stale - for testing only.

`stock_quant (14K) - TEST` `stock_valuation_layer (sample)` `Validate import scripts`

Phase 4 — Week 9 — Sep 6

□ INVENTORY CUTOVER

THE migration moment. Freeze Odoo, physical count, import current inventory. This is the only time inventory data is imported for real.

`stock_quant (14K) - LIVE` `stock_valuation_layer (current FIFO layers)` `stock_scrap (1.4K)` `stock_landed_cost (87)` `stock_landed_cost_lines (88)`

Phase 4 — Week 11 — Sep 20

Odoo Deprecated

Validate inventory accuracy, reconcile counts with system, train warehouse team. Odoo becomes read-only archive.

`Physical count validation` `Book vs actual reconciliation` `Odoo → read-only`

Key Milestones

- PO + Bill System Live — Phase 1 — **Apr 19, 2026**
- MO, Adjustments, BOMs, Locations & Products Live — Phase 2 — **Jul 12, 2026**
- Kit Migration Complete — Phase 3 — **May 10, 2026**
- Full Inventory Cutover

Phase 4

Sep 27, 2026

Phase Details

▼ Phase 1: Purchase Orders & Bills

Feb 8, 2026 → Apr 19, 2026 (10 weeks)

Move all PO creation, arrivals, and vendor bills to SERP. Bills are created MANUALLY after vendor invoice received (not auto-generated). On arrival, push FULL PO (header + lines + receipt) to Odoo for inventory updates.

Goals

- Move purchase order management from Odoo to SERP
- Enable purchasing team to create, edit, and track POs in one place
- Automate inventory updates in Odoo when goods are received
- Preserve manual bill creation workflow (accountant creates bill after receiving vendor invoice)
- Implement 3-way matching: PO → Receipt → Vendor Invoice reconciliation

How It Works

SERP (Laravel)		Odoo
Create PO (draft or sent) — date_order captured		—
Set date_planned (expected arrival date)		—
Email supplier or download PO as PDF		—
Dock receipt (no inventory update yet)		—
Inspection → assign warehouse location → inventory receipt		—
Inventory receipt: effective_date set, sync triggered	→	Push: PO header + lines + stock_picking (receipt) → inventory updated
Mark backorders for future arrival (accept or cancel later)		—
3-WAY MATCHING WORKFLOW (bills in SERP)		
1. System creates DRAFT BILL from receipt (qty_received × price_unit)		—

2. Vendor sends invoice externally (email/mail)	—
3. Accountant compares draft bill vs vendor invoice	—
4a. If match: Accountant finalizes bill → state=posted	—
4b. If mismatch: Kick back to ops → reconcile with vendor	—
5. Finalized bill stored in SERP with line-item detail	—

> **POs + Bills in SERP. On arrival: push full PO + receipt to Odoo (for inventory). Bills created in SERP via 3-way matching. Use effective_date for actual arrival.**

Data Ownership

Data	SERP	Odoo
Purchase orders	**Owner**	Created on arrival (for billing)
PO line items	**Owner**	Created on arrival (for billing)
Backorders	**Owner**	—
Suppliers (156 vendors)	**Owner**	Mapping (partner_id)
Supplier-product links (which products from which supplier)	**Owner**	—
Receiving inspection records	**Owner**	—
Vendor bills	**Owner**	—
Inventory quantities	—	**Owner (stock_quant)**

Database Changes

Table	Purpose	Columns
`entity_message`	Activity log for POs, bills, receipts - tracks	`entity_type, entity_id, message_type, subtype, subject, body, author_id`

	comments, notes, and system events	
`entity_field_change`	Audit trail - tracks what changed, old value vs new value, linked to messages	`message_id, field_name, old_value, new_value`
`entity_attachment`	File attachments on POs/bills (vendor invoices, shipping docs, etc.)	`entity_type, entity_id, message_id, filename, storage_path`
`message_subtype`	Predefined message types (created, updated, RFQ sent, goods received, etc.)	`name, description, entity_type, is_default – seeded with PO/bill subtypes`
`suppliers`	Link SERP vendors to Odoo partners for bi-directional sync	`odoo_partner_id (sync link), supplier_rank (priority 1=preferred), vat (tax ID), address fields`
`locations`	Link SERP warehouse facilities to Odoo warehouse/location IDs	`odoo_warehouse_id, odoo_location_id, active`
`purchase_orders`	PO lifecycle tracking with Odoo-compatible workflow and THREE CRITICAL DATE FIELDS	`odoo_id, name, state (draft→sent→purchase→done), invoice_status, amount_*`
`purchase_orders (dates)`	⚠ THREE DATE FIELDS - understand the difference!	`date_order (CREATED), date_planned (EXPECTED arrival), effective_date (ACTUAL arrival - USE FOR VARIANCE)`
`purchase_order_line`	Line items with quantity tracking for partial receipts and billing	`product_qty (ordered), qty_received (arrived), qty_invoiced (billed), price_unit`
`purchase_order_line (product links)`	Dual-link products: SERP component_id + Odoo odoo_product_id for sync	`component_id (SERP FK), odoo_product_id (Odoo FK)`
`account_journal`	Define journal types that categorize transactions	`name, code (BILL, STJ), type (purchase, sale, general), default_account_id`

	(Vendor Bills, Inventory Valuation, etc.)	
`account_move`	Vendor bills - MANUAL creation after vendor invoice received (3-way matching)	`name (bill #), ref (VENDOR invoice #), state (draft→posted), payment_state, amount_*`
`account_move_line`	Bill line items linked to PO lines for line-level 3-way matching	`quantity, price_unit, price_subtotal, purchase_line_id (FK to PO line!)`
`account_move_purchase_order_rel`	Junction table linking bills to POs (one bill can reference multiple POs)	`account_move_id, purchase_order_id`
`account_payment_term`	Payment term definitions (Net 30, 2% 10 Net 30, etc.) - SEEDED with 8 terms	`name, active, note, sequence — seeded to match Odoo IDs`
`account_payment_term_line`	Payment schedule within terms (for split payments like 30% now, balance 60 days)	`payment_id, value (balance/percent/fixed), value_amount, days`
`procurement_group`	Groups related POs/pickings for traceability (links to origin sales orders)	`name, move_type (direct/one), partner_id, sale_id — created fresh, not migrated`
`odoo_sync_queue`	□ CORE INFRASTRUCTURE - async queue for all Odoo syncs with retry, DLQ, circuit breaker	`entity_type, entity_id, operation (create/update/delete), payload JSON, status (pending/processing/synced/failed/dlq), attempts, last_attempt_at, error_message`

Implementation Details

- Draft POs (save without sending to vendor)
- Two-stage arrival: dock receipt → inspection → inventory receipt
- On arrival: push FULL PO (header + lines) + stock_picking to Odoo — accountants can then create bills
- Location assignment on receipt (which warehouse bin receives goods)

- Supplier communication via email and activity log (similar to Odoo)
- Backorder tracking: mark future arrivals, accept or cancel later
- Partial arrival with delta sync tracking

ODOO SYNC QUEUE (core infrastructure for ALL phases)

- • Async queue pattern: SERP writes locally first, then queues Odoo sync job
- • Odoo sync MUST succeed: retry with exponential backoff (1s, 2s, 4s, 8s, max 5 min)
- • Dead letter queue: after N failures, move to DLQ for manual review
- • Sync status tracking: each record has sync_status (pending/synced/failed/retrying)
- • Admin dashboard: view failed syncs, retry individual items, bulk retry
- • Circuit breaker: if Odoo is down (5+ consecutive failures), pause queue and alert
- • Reconciliation job: daily job compares SERP vs Odoo, flags discrepancies
- • REUSED IN ALL PHASES: MO sync, adjustment sync, BOM sync, location sync, product sync

END SYNC QUEUE

THREE CRITICAL DATE FIELDS (understand the difference!)

- • date_order: When PO was CREATED (auto-populated, audit trail)
- • date_planned: EXPECTED arrival date (user-entered for planning)
- • effective_date: ACTUAL arrival (NULL until goods received!) — USE THIS FOR INVENTORY VARIANCE
- Migrates: 1,846 POs + 16,139 lines with qty_received/qty_invoiced status
- Data cleanup: audit over-receipted lines and cancelled POs with active moves
- Receipt validation with warehouse team before go-live

MANUAL BILL WORKFLOW (bills in SERP)

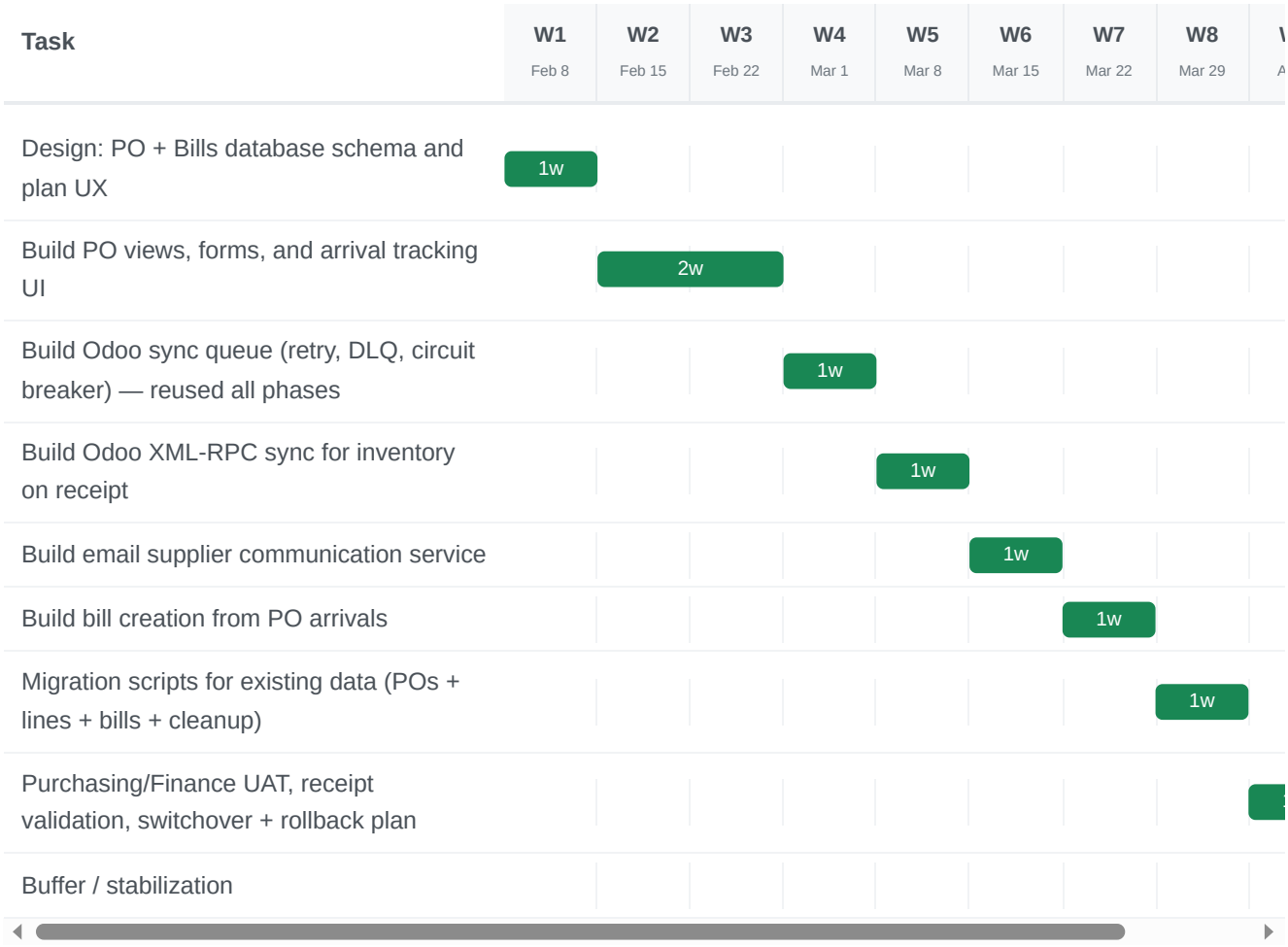
- Draft bill created from receipt (qty_received × price_unit)
- Accountant receives vendor invoice separately (email/mail)
- 3-way match: compare draft bill ↔ vendor invoice ↔ original PO
- If match: accountant finalizes bill
- If mismatch: ops reconciles with vendor before finalizing
- Bill tracks qty_invoiced vs qty_received for partial billing
- Migrates: 1,569 vendor bills from Odoo account_move

Team Confirmations Before Launch

- Purchasing team - PO creation and editing workflow

- Warehouse team - receipt and inspection process
- Finance team - 3-way matching workflow (draft bill vs vendor invoice)
- Finance team - bill approval and finalization process

Timeline



▼ Phase 2: Manufacturing, Adjustments, BOMs, Locations & Products

Apr 19, 2026 → Jul 12, 2026 (12 weeks) • Runs in parallel

Build Odoo features in SERP. Users switch to SERP UI, which dual-writes to both Odoo (XML-RPC) and SERP tables (stock_quants). Incremental by feature.

Goals

- Build features in SERP that replicate Odoo functionality
- Users switch to SERP UI for each feature (stop using Odoo UI)
- Dual-write: SERP sends XML-RPC to Odoo AND updates SERP shadow tables
- stock_quants table is independent (no linkage to component/RM inventory yet)
- Incremental rollout: Manufacturing → Adjustments → BOMs → Stock Locations → Product Info
- Track COGS correctly throughout transition

How It Works

SERP (Laravel)		Odoo
User performs action in SERP UI (stops using Odoo UI)		Odoo UI disabled for this feature
SERP dual-writes: XML-RPC to Odoo + update stock_quants	→	Receives sync, stays in sync
stock_quants updated independently (not linked to component inventory)		—
MO created/completed/scrapped in SERP	→	XML-RPC sync
Inventory adjustment in SERP + COGS tracking	→	XML-RPC sync
BOM create/edit in SERP	→	XML-RPC sync
Location management in SERP	→	XML-RPC sync
Product management in SERP	→	XML-RPC sync

> **SERP becomes the UI, dual-writes to Odoo. stock_quants is independent (not linked to component inventory yet). Feature-by-feature migration.**

Data Ownership

Data	SERP	Odoo
Manufacturing orders	**UI + Owner**	Synced via XML-RPC
Inventory adjustments	**UI + Owner**	Synced via XML-RPC
BOMs (2,078 total)	**UI + Owner**	Synced via XML-RPC
Stock locations	**UI + Owner**	Synced via XML-RPC
Products (5,809)	**UI + Owner**	Synced via XML-RPC
stock_quants (shadow)	**Owner (independent)**	Has its own copy
component_inventory	Unchanged	—
Valuation layers (COGS)	**Owner**	—

Database Changes

Table	Purpose	Columns
`manufacture_orders`	Track production orders with Odoo workflow (draft→confirmed→progress→done)	`name, odoo_id, receiver_product_id (OUTPUT), state, bom_id, product_qty, qty_produced, location_src/dest_id`
`mrp_bom`	BOM headers - define "recipes" (what components make a product)	`code, product_id (OUTPUT), type (normal/phantom), consumption, active, odoo_bom_id`
`mrp_bom_line`	BOM components - qty of each component needed per 1 unit of output	`bom_id, product_id (component), product_qty, sequence, odoo_bom_line_id`
`uom_category`	Unit of measure categories (Unit, Weight, Volume, etc.) - SEEDED with 7 categories	`name – seeded to match Odoo IDs`
`uom_uom`	Units of measure with conversion factors - SEEDED with 27 units	`name, category_id, factor, rounding, uom_type (reference/bigger/smaller)`

`stock_location`	Hierarchical warehouse locations (bins, zones, areas) - different from SERP "locations" table	`name, complete_name (path), parent_id, usage (internal/supplier/customer), scrap_location`
`stock_picking_type`	Define operation types - how receipts, deliveries, transfers behave	`name, code (incoming/outgoing/internal/mrp), default_location_src/dest_id, reservation_method`
`stock_quant`	□ INVENTORY TRUTH - current on-hand qty per product/location. Available = quantity - reserved	`component_id/receiver_product_id, location_id, quantity, reserved_quantity, in_date, unit_cost`
`stock_picking`	Warehouse operations (receipts, deliveries, transfers) - groups stock_moves	`name (WH/IN/00123), origin (PO#), location_id→location_dest_id, state (draft→assigned→done)`
`stock_move`	Individual inventory movements - move X units from A to B. Only state=done updates inventory	`product_uom_qty (requested), quantity_done (actual), state, picking_id, production_id`
`stock_move_line`	Detailed lines within stock_move - for lot/serial tracking (one move can have multiple lots)	`move_id, qty_done, location_id→location_dest_id`
`stock_valuation_layer`	□ FIFO COSTING - each receipt creates a layer; consumption decrements oldest first	`quantity, unit_cost (from PO), remaining_qty (unconsumed), remaining_value (for GL)`
`product_supplierinfo`	Supplier-specific pricing/lead times per product - critical for PO cost calculation	`supplier_id, component_id, price, min_qty (MOQ), delay (lead time days), case_qty`
`stock_scrap`	Track scrapped/damaged inventory - moves stock to scrap location	`scrap_qty, location_id (from), scrap_location_id (to), production_id (if during MO)`
`stock_landed_cost`	Add freight/duties/customs to inventory value after receipt	`amount_total, state, vendor_bill_id, split_method (equal, by_qty, by_weight)`
`stock_landed_cost_lines`	Individual cost items within a landed cost (freight line, customs line, etc.)	`cost_id, name, product_id, price_unit, split_method, account_id`

`stock_landed_cost_stock_picking_rel`	Junction: which receipts (pickings) a landed cost applies to	`stock_landed_cost_id, stock_picking_id`
`mrp_unbuild`	Disassembly orders - reverse a BOM to recover components from finished product	`product_id (to unbuild), bom_id, product_qty, production_id (optional - unbuild specific MO), state`

Implementation Details

- Incremental rollout: MO → Adjustments → BOMs → Locations → Products
- Pattern: build feature in SERP → users switch to SERP UI → disable Odoo UI
- Dual-write on every action: XML-RPC to Odoo + update SERP tables

USES SYNC QUEUE FROM PHASE 1

- • Same odoo_sync_queue table and infrastructure built in Phase 1
- • Extend with new entity_types: mrp_production, stock_move, mrp_bom, stock_location, product
- • Same retry logic, DLQ, circuit breaker patterns

END SYNC QUEUE

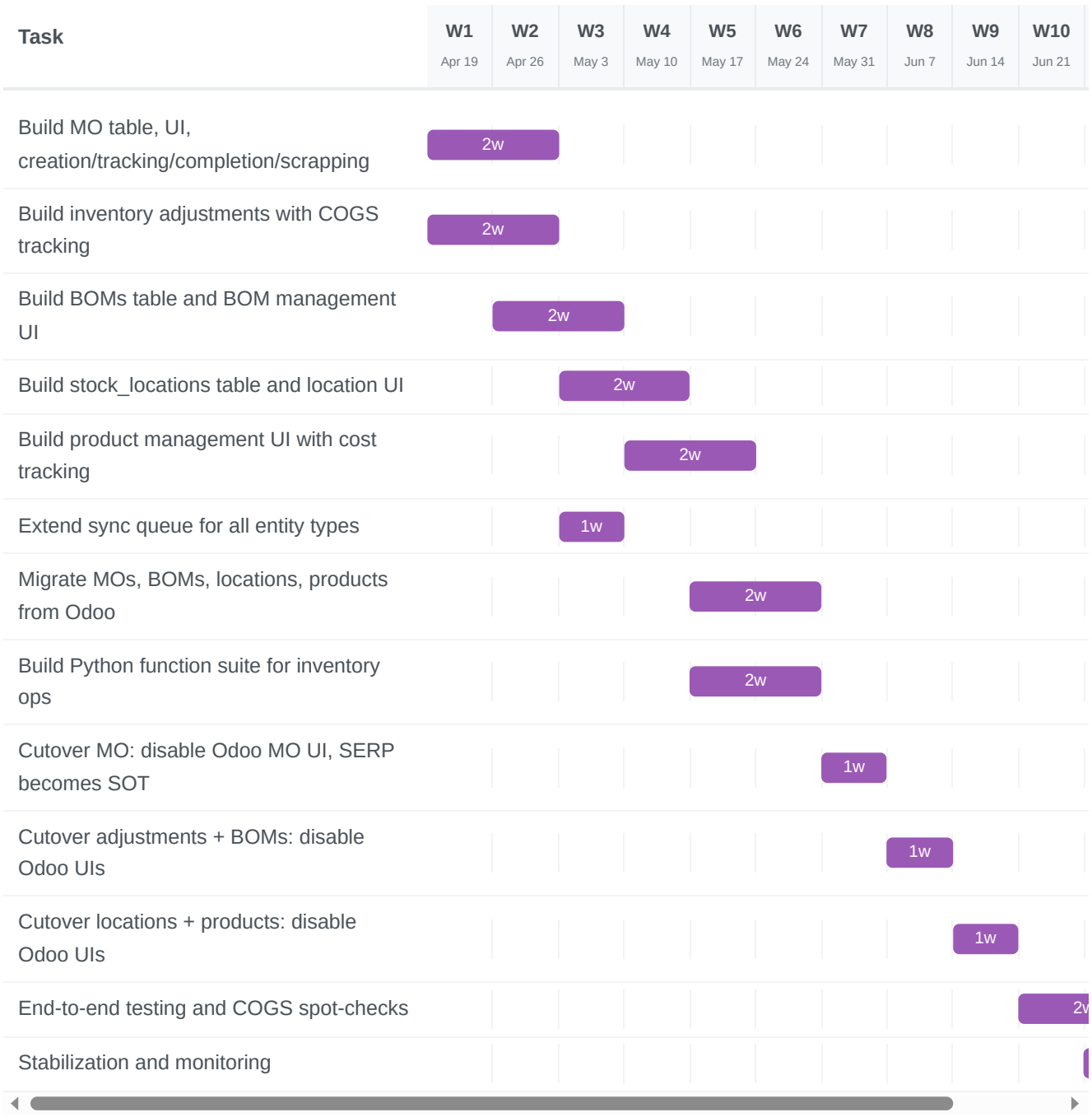
- stock_quants is independent shadow table (NOT linked to component/RM inventory)
- Existing component_inventory unchanged during this phase
- Manufacturing: MO in SERP, dual-writes to Odoo mrp_production
- MO scrapping/cancellation: cancel defective MOs, reverse any partial consumption
- Adjustments: adjustment in SERP updates stock_quants + XML-RPC to Odoo stock_move
- BOMs: BOM management in SERP, syncs to Odoo mrp_bom
- Stock locations: location hierarchy in SERP, syncs to Odoo stock_location
- Products: product info in SERP, syncs to Odoo product_product
- COGS tracking: valuation layers updated on MO completion and adjustments
- Odoo stays in sync but users stop using Odoo UI for migrated features
- Python function suite: reusable functions for inventory ops that maintain DB integrity
- Functions: make_adjustment(), receive_arrival(), complete_mo(), scrap_mo()
- All functions enforce COGS structure, update valuation layers, log transactions

Team Confirmations Before Launch

- Production team - MO creation and completion workflow

- Warehouse team - inventory adjustments and location management
- Finance team - COGS calculations and valuation accuracy
- Operations team - BOM management and product info

Timeline



▼ Phase 3: Kits

Feb 8, 2026 → May 10, 2026 (13 weeks) • Runs in parallel

Start planning immediately, migrate phantom BOMs from Odoo to Laravel. ADDITIVE logic: use BOTH existing kits AND Laravel BOMs together.

Goals

- Begin discovery and planning immediately (parallel with Phase 1)
- Meet with Carolyn to understand kit/BOM workflows and requirements
- Migrate phantom BOMs from Odoo mrp_bom to Laravel mrp_bom table
- Modify updateInsertKit() to query BOTH existing kits AND Laravel mrp_bom (additive logic)
- Match buyer_products.sku → product_template.default_code → mrp_bom
- Create ComponentOrders from BOTH sources - union of kit components + BOM components
- Build kit editing UI in SERP (create, edit, delete kits)
- Deprecate Odoo phantom BOM lookups

How It Works

SERP (Laravel)		Odoo
DATA MIGRATION (Odoo → SERP, one-time)		
Export mrp_bom WHERE type=phantom AND active=true (265 BOMs)		Source
Export mrp_bom_line for those BOMs (~800 component records)		Source
Export product_template for SKU matching (default_code)		Source
Load into SERP mrp_bom + mrp_bom_line + product_template tables		—
Validate: verify component_id mappings, count records		—
ADDITIVE LOGIC (do BOTH kits + BOMs)		
1. Get existing kit components (if buyer_product has kit)		—
2. ALSO query mrp_bom WHERE type=phantom AND active=true		—

3. Match: buyer_products.sku = product_template.default_code	—
4. Get mrp_bom_line → map product_id to components.odoo_id	—
5. UNION both sets of components (kit + BOM)	—
6. Create ComponentOrders for ALL components from both sources	Deprecated

> **ADDITIVE: updateInsertKit() uses BOTH existing kits AND Laravel mrp_bom (not fallback). Components from both sources are unioned. SKU matching: buyer_products.sku = product_template.default_code.**

Data Ownership

Data	SERP	Odoo
mrp_bom (phantom type)	**Owner**	Deprecated
mrp_bom_line (components)	**Owner**	Deprecated
product_template (SKU matching)	**Owner**	Deprecated
Existing kits table	**Owner (unchanged)**	—

Database Changes

Table	Purpose	Columns
`mrp_bom (CREATE in SERP)`	Phantom BOMs migrated from Odoo. 265 active records. Same schema as Odoo for compatibility.	`id, code, product_tmpl_id, type (phantom), active, product_qty, product_uom_id, odoo_bom_id (link back to Odoo)`
`mrp_bom_line (CREATE in SERP)`	BOM components migrated from Odoo. ~800 records for 265 phantom BOMs.	`id, bom_id (FK), product_id (→ components.odoo_id), product_qty, sequence, odoo_bom_line_id`
`product_template (CREATE in SERP)`	SKU mapping table. Links buyer_products.sku to mrp_bom via default_code field.	`id, name, default_code (SKU), odoo_product_tmpl_id`
`kits (EXISTS - unchanged)`	Existing Laravel kits table. 1,091 active kits. NOT replaced by mrp_bom.	`Unchanged - both kits AND mrp_bom used together (additive)`

`component_kits (EXISTS - unchanged)`	Existing kit-component mappings. Used alongside mrp_bom_line.	`Unchanged - additive logic unions both sources`

Implementation Details

Data Migration (Odoo → SERP)

WHAT GETS MIGRATED

- mrp_bom: 265 active phantom BOMs (type=phantom, active=true)
- mrp_bom_line: ~800 component records linked to those BOMs
- product_template: SKU mapping table (default_code = buyer_products.sku)

MIGRATION SCRIPT

- Export from Odoo: mrp_bom JOIN mrp_bom_line JOIN product_template
- Transform: map product_id → components.odoo_id for each line
- Load into Laravel: mrp_bom + mrp_bom_line tables (same schema as Odoo)
- Validate: count records, verify component mappings exist

EDGE CASES

- Zero-qty consumables (tape, labels): preserved with qty=0
- Inactive BOMs (511 records): NOT migrated - only active=true
- Normal BOMs (977 records): migrated in Phase 2, not this phase

Code Changes (updateInsertKit)

- ADDITIVE LOGIC: Query BOTH existing kits AND mrp_bom (not fallback)
- Step 1: Get kit components if buyer_product has assigned kit
- Step 2: Match buyer_products.sku = product_template.default_code
- Step 3: Query mrp_bom WHERE product_tmpl_id=X AND type=phantom AND active=true
- Step 4: Loop mrp_bom_line → map product_id to components.odoo_id
- Step 5: UNION kit components + BOM components (dedupe by component_id if needed)
- Step 6: Create ComponentOrder for EACH component from combined set
- Deduct inventory from component location (same as existing)

Odoo Sync (Already Handled)

- ComponentOrders already sync to Odoo via existing /api/odoo/component/prepick endpoint
- Odoo polls Laravel, fetches orders where component_imported=0
- ComponentResource sends: odoo_id, sku, name, quantity from pivot table

- No additional sync code needed - phantom BOM ComponentOrders work automatically

Validation

- Diff report: compare ComponentOrders created by Laravel vs Odoo
- Test orders with kits (existing path unchanged)
- Test orders without kits (new phantom BOM path)
- Verify component mapping: mrp_bom_line.product_id → components.odoo_id
- Verify ComponentOrders from phantom BOMs sync to Odoo correctly

Team Confirmations Before Launch

- Ops - phantom BOM components match expected packaging
- Tech - ComponentOrders created correctly for both kit and BOM paths

Timeline

Test order expansion: verify kit + phantom BOM components combined	1w
Parallel run: compare ComponentOrders between systems	1w
Cutover: disable Odoo phantom BOM lookups	
Monitor and stabilize	

▼ Phase 4: Inventory Management

Jul 12, 2026 → Sep 27, 2026 (11 weeks)

Make SERP the source of truth for inventory. Create views to expose stock_quant quantities. Build order processing queue. Build financial reports for Ric. Deprecate Odoo.

Goals

- Migrate all inventory from Odoo to SERP stock_quant (quantities + FIFO cost layers)
- Make stock_quant the single source of truth for inventory
- Create inventory views (receiver_product_inventory, component_inventory) that aggregate stock_quant
- Update all code to JOIN views instead of reading inventory_qty columns
- Build order processing queue (reserve → ship → COGS)
- Build financial reports for Ric (Valuation, Roll Forward, FIFO COGS, Cost Tracking)
- Complete cutover: activate SERP queues, disable Odoo sync

How It Works

Inventory Views

SERP (Laravel)	Odoo
stock_quant is source of truth for inventory quantities	—
receiver_product_inventory view aggregates stock_quant by receiver_product_id	—
component_inventory view aggregates stock_quant by component_id	—
Code JOINs views instead of reading inventory_qty columns	—
Physical count reconciliation before cutover	Final sync

Order Flow

SERP (Laravel)	Odoo
Order created → stock_move_id = NULL	—

SERP queue finds unprocessed (WHERE stock_move_id IS NULL)	—
SERP reserves inventory, writes stock_move_id back	—
Order ships → SERP consumes FIFO, writes cost_unit back	—
Order cancelled before ship → release reservation	—

SERP becomes source of truth. Inventory views aggregate stock_quant (no sync needed). Order queue handles reservations and COGS.

Data Ownership

Data	SERP	Odoo
stock_quant (source of truth)	**Owner**	Deprecated
receiver_product_inventory (view)	Aggregates stock_quant	—
component_inventory (view)	Aggregates stock_quant	—
Order inventory deductions	**Owner**	Deprecated
COGS (cost_unit on items/component_orders)	**Owner**	—

Database Changes

Table	Purpose	Columns
`receiver_product_inventory (VIEW)`	Aggregates stock_quant for finished goods - replaces receiver_products.inventory_qty column	`receiver_product_id, total_qty, reserved_qty, available_qty (total - reserved)`
`component_inventory (VIEW)`	Aggregates stock_quant for components - replaces components.inventory_qty column	`component_id, total_qty, reserved_qty, available_qty (total - reserved)`
`items`	Link order items to inventory moves + track FIFO cost at fulfillment	`stock_move_id (FK to stock_move), cost_unit (FIFO cost per unit)`
`component_orders`	Link component orders to inventory moves + track FIFO cost at	`stock_move_id (FK to stock_move), cost_unit (FIFO cost per unit)`

	fulfillment	
`ec_order`	Track order state and link to warehouse picking operations	`serp_state (pending→confirmed→fulfilled)`, `serp_picking_id (FK to stock_picking)`
`× DROP legacy tables`	Remove tables replaced by Odoo-style inventory system	`serp_audit_logs, component_shelves, supplier_product, raw_materials, raw_material_inventory, rm_inventory_transactions, component_inventory, component_inventory_transactions`

Implementation Details

Inventory Views

- Create receiver_product_inventory view: SELECT receiver_product_id, SUM(quantity), SUM(reserved_quantity) FROM stock_quant GROUP BY receiver_product_id
- Create component_inventory view: SELECT component_id, SUM(quantity), SUM(reserved_quantity) FROM stock_quant GROUP BY component_id
- Index stock_quant on (receiver_product_id) and (component_id) for view performance

Code Migration

- Find all code reading receiver_products.inventory_qty → change to JOIN receiver_product_inventory
- Find all code reading components.inventory_qty → change to JOIN component_inventory
- Update queries, reports, dashboards, APIs that reference inventory columns
- Test all inventory-related features with new views

Database Changes

- items: ADD stock_move_id (BIGINT), cost_unit (DECIMAL 12,4)
- component_orders: ADD stock_move_id (BIGINT), cost_unit (DECIMAL 12,4)

Order Processing Queue

- Queue runs every 5 minutes
- Process new orders: find items/component_orders WHERE stock_move_id IS NULL
- Fulfill shipped orders: when ec_order.ship_date is set, consume FIFO, write cost_unit
- Cancel orders: if cancelled before ship, release reservation

New Order Processing

- Find ec_order where line items have no stock_move_id
- Map items.product_sku → receiver_products
- Create stock_picking + stock_move, reserve inventory
- Write stock_move_id back to items/component_orders

Ship Order Processing

- Find orders where ship_date is set but cost_unit is empty
- Consume oldest FIFO layers, decrease stock_quant.quantity
- Write cost_unit to items/component_orders

Cancel Order Processing

- Before ship: release reservation, cancel moves
- After ship: no action - COGS already recorded

Data Migration

- Export all receiver_product inventory from Odoo stock_quant to SERP stock_quant
- Export all component inventory from Odoo stock_quant to SERP stock_quant
- Export FIFO cost layers from Odoo stock_valuation_layer to SERP
- Physical count reconciliation to verify migrated quantities

Cutover Checklist

- ☐ All inventory data migrated from Odoo to SERP stock_quant
- ☐ All code updated to use inventory views (no direct inventory_qty reads)
- ☐ Inventory views created and tested (receiver_product_inventory, component_inventory)
- ☐ SERP order processing queue built and tested
- ☐ Parallel run completed (both Odoo and SERP process same orders)
- ☐ COGS accuracy validated between Odoo and SERP
- ☐ Physical count reconciliation completed
- ☐ Legacy tables dropped (serp_audit_logs, component_shelves, etc.)
- ☐ Odoo order sync disabled
- ☐ SERP order queue activated as primary
- ☐ Monitoring/alerting configured for queue failures
- ☐ Rollback plan documented and tested

Team Confirmations Before Launch

- Ops - inventory counts match stock_quant
- Finance - COGS calculations and valuation accuracy
- Tech - all syncs work

Timeline

▼ Future Improvements

Features not currently in Odoo — to be built as needed

These are **new capabilities** that don't exist in Odoo today. They can be built in SERP independently of the core migration phases, prioritized by business need.

Why These Are Separate

These features don't exist in Odoo. The main migration phases focus on replacing existing Odoo functionality. These improvements are net-new capabilities that can be built on-demand.

Available Improvements

Blanket PO Tracking

Track long-term vendor agreements with multiple releases. SERP manages the agreement, Odoo receives individual release POs.

- **Create Agreement:** Blanket PO in SERP with total qty, price terms, validity period
- **Release Orders:** Each release creates a real purchase.order in Odoo
- **Track Usage:** SERP tracks total released vs. agreement amount

Why not in Odoo: No native blanket/framework PO support exists.

PO Approval Workflows

Route POs for approval based on amount thresholds. Notify via Slack, update Odoo when approved.

- **Threshold Rules:** POs over $X need manager, over $Y need director
- **Multi-Step:** Ops approval first, then Finance for larger amounts
- **Slack Integration:** Approver receives Slack message with approve/reject buttons

Why not in Odoo: studio_approval_rule exists but has 0 active rules configured.

Supplier Forecast → Odoo Integration

Turn SERP supplier forecasts into draft POs in Odoo with one click.

- **Generate:** Supplier Forecast calculates what to order based on demand
- **Review:** User reviews and adjusts suggested quantities
- **Push:** Click "Send to Odoo" to create draft purchase.order records

Why not in Odoo: Odoo has no forecasting integration. Tool exists in SERP already.

Inventory Adjustment Approvals

Require approval for manual inventory adjustments above a threshold.

- **Request:** User submits adjustment with reason code
- **Threshold Check:** Large adjustments require approval
- **Audit Trail:** Log who requested, who approved, and why

Why not in Odoo: Odoo allows direct adjustments with no approval gate.

Other Potential Improvements

Additional features discussed but not yet prioritized:

- **Bulk BOM Import:** Import BOMs from spreadsheet with validation
- **Product/BOM Archiving:** Soft-delete with archive flag instead of hard delete
- **Warehouse Location Tracking:** Track inventory by pallet/location within warehouse
- **Bulk Inventory Adjustments:** Adjust multiple items at once from spreadsheet