Salesforce Testing Scenarios: Real-World Examples

A Practical Guide to Testing Critical Business Processes

How to Use This Guide

Salesforce testing theory only takes you so far. Real expertise comes from understanding how to test actual business processes with all their complexity, edge cases, and integration points.

This guide walks through six common but challenging Salesforce testing scenarios that we encounter across industries. Each scenario represents real business processes that generate revenue, serve customers, or manage critical operations.

What you'll find in each scenario:

- **Business process overview** - Understanding what the workflow actually does and why it matters
- **Testing challenges** - Specific complications that make this scenario difficult to test
- **Validation requirements** - What comprehensive testing must verify
- **Test case examples** - Concrete scenarios to validate
- **Automation approach** - How to balance automated and manual testing
- **Common pitfalls** - Mistakes to avoid based on real implementations

Who should use this guide:

- QA engineers designing test strategies for Salesforce implementations
- Salesforce developers building testable customizations
- Business analysts defining acceptance criteria for complex processes
- Project managers planning testing timelines and resources
- Anyone responsible for ensuring Salesforce quality and reliability

How these scenarios connect to comprehensive testing:

These examples demonstrate the testing types, tools, and best practices covered in our main Salesforce Testing Guide (Parts 1 and 2). If you haven't read those yet, they provide essential context on:

- Why Salesforce testing is uniquely complex
- Types of testing and when to use each
- Automation tools and selection criteria
- Best practices for maintainable test automation
- Roles and team structure for effective testing

Think of this scenarios guide as the practical application of those principles. Theory meets reality.

A note on customization:

Your Salesforce org is unique. These scenarios provide frameworks and approaches, not scripts to copy verbatim. Adapt the test cases, automation strategies, and validation points to your specific business processes, data model, and integration landscape.

The goal isn't to test exactly what we describe here. It's to learn the thinking process: how to break down complex workflows into testable components, identify critical validation points, and design automation that balances coverage with maintainability.

Let's dive into the scenarios.

Scenario 1: Lead-to-Opportunity Conversion Workflow

This is one of the most critical revenue-generating processes in any Salesforce implementation. When lead routing breaks, sales opportunities disappear. When scoring logic fails, high-value prospects get ignored. When conversion processes have issues, revenue pipeline reports become unreliable.

Business Process Overview

The lead-to-opportunity workflow typically spans multiple systems and teams:

1. **Lead generation** - Marketing automation platform (HubSpot, Marketo, Pardot) generates leads through campaigns
2. **Lead capture** - Leads flow into Salesforce via API integration or web-to-lead forms
3. **Lead scoring** - Salesforce assigns scores based on company characteristics, engagement, and firmographic data
4. **Lead routing** - Assignment rules distribute leads to appropriate sales teams based on territory, industry, company size, or product interest
5. **Sales qualification** - Sales reps review, update, and qualify leads through discovery conversations
6. **Lead conversion** - Qualified leads convert to accounts, contacts, and opportunities
7. **Integration updates** - Conversion triggers updates to ERP system for credit checking, marketing automation for nurture campaign removal, and data warehouse for analytics

At each step, business logic executes, data transformations occur, and integration points create potential failure modes.

Testing Challenges

Multi-system dependencies. The complete workflow touches marketing automation, Salesforce, ERP, and analytics systems. Testing requires data in the marketing system, properly configured Salesforce rules, and working integration endpoints.

Complex business logic. Lead scoring algorithms incorporate multiple data points: company size, industry, revenue, website engagement, email interactions, content downloads. Getting scoring logic wrong means misrouting valuable prospects.

Timing and asynchronicity. Leads don't process instantly. Marketing platform to Salesforce sync might take minutes. Lead scoring triggers can run asynchronously. Integration updates happen in queued jobs. Tests must handle these timing variations.

Data volume considerations. Marketing campaigns can generate thousands of leads overnight. Bulk import processes must respect governor limits while processing efficiently. Volume testing requires production-scale data.

Territory complexity. Assignment rules can be Byzantine: geographic territories with exceptions, industry-based routing with company size thresholds, round-robin distribution with workload balancing. Edge cases multiply quickly.

What Comprehensive Testing Validates

Lead scoring accuracy across scenarios:

Test that scoring algorithms produce correct results for different lead profiles:

- **High-value enterprise lead** - Technology industry, 500+ employees, $5M+ revenue, engaged with pricing content → Score: 85-90, Route to: Enterprise sales team
- **Mid-market qualified lead** - SaaS company, 100-500 employees, $1M-$5M revenue, multiple touchpoints → Score: 70-80, Route to: Regional account executive
- **Small business lead** - Local service company, <50 employees, <$500K revenue, single form fill → Score: 30-40, Route to: Inside sales or nurture campaign
- **Unqualified lead** - Personal email domain, missing company information, no engagement → Score: 10-20, Route to: Disqualified or long-term nurture

For each profile, validate that:

- Score calculation incorporates all relevant data points correctly
- Missing data fields don't break scoring logic
- Score thresholds trigger appropriate routing rules
- Scores update when lead data changes

Assignment rule validation

Test that leads route to correct teams and individuals:

- **Territory-based routing** - California leads → West Coast sales team, New York leads → East Coast team
- **Industry specialization** - Healthcare leads → Healthcare specialists, Financial services → FinServ team
- **Size-based routing** - Enterprise accounts (1000+ employees) → Strategic accounts team, SMB (<100 employees) → Inside sales
- **Round-robin distribution** - Leads distribute evenly among available reps within a team
- **Overflow handling** - When primary rep is at capacity or unavailable, leads route to backup
- **Weekend/off-hours** - Leads arriving outside business hours queue appropriately

Edge cases to test:

- Lead matches multiple territory rules simultaneously (conflict resolution)
- Assigned rep is on vacation or leaves company
- Territory definitions change mid-campaign
- Lead data updates trigger re-routing

Lead conversion process integrity

Validate that conversion maintains data relationships and triggers downstream processes:

- **Account creation or matching** - New accounts created for new companies, existing accounts matched correctly, duplicate prevention works
- **Contact creation** - Contact records preserve lead information, link to correct account, maintain custom field data
- **Opportunity creation** - Opportunities capture deal information, link to correct account and contact, populate with lead source data
- **Historical data preservation** - Lead history, activity records, and custom field data transfer to new records
- **Related record updates** - Campaign members, list memberships, and integration records update appropriately

Test conversion variations:

- Converting to existing account vs. creating new account
- Multiple contacts from same company
- Conversion with incomplete lead data
- Bulk conversion of multiple leads simultaneously

Integration data flow validation

Test that lead data synchronizes correctly with external systems:

Inbound integration (Marketing → Salesforce)

- Field mappings preserve data correctly (marketing platform fields → Salesforce lead fields)
- Required Salesforce fields populate from marketing data
- Custom field synchronization works
- Bulk lead imports handle thousands of records without timeout
- Failed imports retry appropriately with error logging

Outbound integration (Salesforce → other systems)

- Lead status changes trigger marketing automation updates (qualified → remove from nurture, disqualified → add to exclusion list)
- Conversion events notify ERP system for credit check initiation
- Analytics system receives lead lifecycle data for reporting
- API calls respect rate limits and handle failures gracefully

Error handling scenarios

- External system temporarily unavailable → Salesforce queues retry
- Malformed data from marketing system → Salesforce logs error without blocking process
- API authentication failure → Appropriate notifications sent
- Data validation failures → Clear error messages for troubleshooting

Performance under realistic volume

Validate that the process handles production-scale loads:

- **Bulk lead import** - 5,000 leads imported overnight process within acceptable timeframe
- **Scoring calculations** - Complex scoring logic doesn't hit CPU time limits with bulk processing
- **Assignment rule execution** - Territory evaluation completes without SOQL query limit violations
- **Integration API calls** - Batch processing respects API rate limits (15,000-100,000 calls per 24 hours)

Performance test scenarios:

- Normal daily volume (100-200 leads)
- Campaign spike (5,000 leads in 2 hours)
- Month-end bulk import (10,000+ leads)
- Concurrent manual lead creation by sales team during bulk import

Automation Approach

Unit testing for scoring logic (Apex Testing Framework)

Use the Apex Testing Framework to validate scoring rules and ensure reliability:

- **Create dedicated test classes** that isolate and verify lead scoring calculations.
- **Test high-value enterprise leads** to confirm expected scoring outcomes.
- **Validate bulk scoring performance** with large batches of leads to check governor limits and efficiency.
- **Include negative and edge cases** (e.g., missing industry, low revenue, no title) to ensure robustness.
- **Use TestDataFactory** for consistent, reusable test data creation.
- **Apply Test.startTest / Test.stopTest** to measure execution and isolate test context.
- **Assert meaningful results** rather than just non-null values, to guarantee business logic correctness.

Unit tests bypass UI complexity and directly validate the scoring engine's reliability.

Unit tests run in seconds, catch logic errors immediately, and validate governor limit compliance with bulk operations.

API testing for integration endpoints

Use tools like Postman, REST Assured, or custom test frameworks to validate integration APIs:

- Test lead creation via API with various data payloads
- Validate authentication and authorization
- Test error responses (invalid data, missing required fields, rate limiting)
- Verify data format transformations
- Test retry logic with service virtualization

API tests bypass UI complexity and test integration reliability directly.

UI automation for end-to-end workflow

Use Salesforce-specific automation tools (Provar, Leapwork) or adapted Selenium to validate complete user experience:

- Sales rep manually creates lead through Salesforce UI
- Lead scores and routes automatically
- Rep qualifies lead and converts to opportunity
- Verify that opportunity, account, contact created correctly
- Validate UI displays appropriate data throughout workflow

UI tests are slower and more fragile than API tests, so focus on critical user journeys only.

Manual exploratory testing

QA engineers and business users test scenarios that automation struggles with:

- Unusual data combinations that edge case logic
- User experience during error conditions
- Workflow variations that occur infrequently
- New features before automation is built

Common Pitfalls to Avoid

Pitfall #1: Testing only happy path. Most tests validate successful lead conversion with complete, clean data. Real leads have missing fields, invalid data, and unusual characteristics. Test error scenarios explicitly.

Pitfall #2: Ignoring timing issues. Tests that work in dev sandbox fail in production because asynchronous processes haven't completed. Build intelligent waiting into automation.

Pitfall #3: Hard-coded test data. Tests that depend on specific lead records break when sandbox refreshes. Generate test data programmatically within tests.

Pitfall #4: Not testing bulk operations. Logic that works with one lead might hit governor limits with 200 leads. Bulk testing is mandatory.

Pitfall #5: Skipping integration testing. Focus on Salesforce-only testing ignores the reality that most lead processes span multiple systems. Integration failures are common production issues.

Scenario 2: Complex Approval Workflow Testing

Approval processes in Salesforce often involve multiple decision points, parallel approvals, time-based escalations, and integration with external systems. Testing these workflows comprehensively requires validating all possible paths, timing scenarios, and edge cases.

Business Process Overview

Typical complex approval scenario: Discount approval for enterprise deals

1. **Initiation** - Sales rep requests discount >15% on opportunity
2. **Automatic approval tier** - Deals <$50K with discounts <20% auto-approve
3. **Manager approval tier** - Deals $50K-$250K require sales manager approval
4. **Director approval tier** - Deals $250K-$1M require sales director + finance approval (parallel)
5. **Executive approval tier** - Deals >$1M require VP Sales + CFO approval (parallel)
6. **Time-based escalation** - Approvals pending >48 hours escalate to next level

7. **Integration trigger** - Approved discounts sync to CPQ system for quote generation
8. **Notification chain** - Stakeholders notified at each stage via email and Chatter

This workflow has multiple decision points, conditional routing, parallel approval requirements, and integration dependencies.

Testing Challenges

Path explosion. With 5+ decision points and multiple approval tiers, the number of possible workflow paths grows exponentially. Testing every combination is impractical.

Parallel approval complexity. When multiple approvers must respond (Director + Finance), tests must validate:

- All required approvals collected before proceeding
- Any rejection stops the process immediately
- Approval order doesn't matter (unless it should)
- Partial approval state tracked correctly

Time-based triggers. Escalations that occur after 48 hours create testing challenges:

- Waiting 48 hours in real-time is impractical
- Date/time manipulation can have side effects
- Need to validate escalation logic without actual waiting

Dynamic routing changes. If deal size or discount percentage changes during approval, routing may need to restart or adjust. Tests must cover these scenarios.

Integration synchronization. Approved discounts must flow to quote system correctly, handling various success and failure scenarios.

What Comprehensive Testing Validates

Routing logic accuracy across deal tiers:

Test that opportunities route to correct approvers based on deal characteristics:

Tier 1: Auto-approval

- $45K deal, 18% discount → Auto-approved, no manual intervention
- Verify approval record created with "Auto-Approved" status
- Validate notification sent to rep
- Confirm integration trigger fires

Tier 2: Manager approval

- $100K deal, 22% discount → Routes to sales manager only
- Verify correct manager identified (territory-based, rep hierarchy)
- Test approval and rejection outcomes
- Validate rep notification of decision

Tier 3: Director approval

- $500K deal, 25% discount → Routes to director AND finance (parallel)
- Both approvers receive requests simultaneously
- Both must approve before proceeding
- Either can reject to stop process
- Verify partial approval state if one approves, one hasn't responded

Tier 4: Executive approval

- $2M deal, 30% discount → Routes to VP Sales AND CFO (parallel)
- Same parallel approval validation as Tier 3
- Additional validation of approval history tracking

Edge case routing:

- Deal exactly at tier boundary ($250,000) → Correct tier assignment
- Discount exactly at threshold (20%) → Correct approval path
- Multiple criteria conflicts → Highest tier wins

Approval action validation

Test all possible approver actions and their effects:

Approval path

- Approver clicks "Approve" → Opportunity progresses to next stage or final approval
- Approval record updates with timestamp and approver comments
- Notifications sent to next approver in chain
- Rep receives notification of approval progress

Rejection path

- Approver clicks "Reject" → Opportunity returns to rep with rejection reason
- Entire approval process cancels (doesn't continue to other approvers)
- Rep receives notification with rejection comments
- Opportunity status updates appropriately

Reassignment

- Approver reassigns to different person → Request routes to new approver
- Original approver removed from approval chain

- History preserves reassignment action and reason

Recall

- Rep recalls approval request before approval → Request cancels
- Approvers notified of recall
- Opportunity returns to editable state

Timeout/no response

- Approval pending for 48 hours → Escalation triggers
- Escalation notification sent to higher-level manager
- Original approver still can approve
- Escalation history tracked

Escalation Logic Testing

Time-based escalations require special testing approaches to avoid waiting real hours or days:

- **Use date/time manipulation in Apex tests**: simulate passage of time with `Test.setCreatedDate` instead of waiting 48 hours.
- **Submit records for approval** and then trigger escalation jobs programmatically (e.g., via batch execution).
- **Verify escalation outcomes** by querying process instances and asserting expected statuses.
- **Test edge cases**: records that should not escalate, escalations with different thresholds, and multiple concurrent approvals.
- **Leverage Test.startTest / Test.stopTest** to isolate execution and ensure scheduled jobs run within the test context.
- **In UI automation**, adjust record timestamps directly via API before running escalation workflows to avoid long delays.
- **Combine functional and performance checks**: confirm escalations happen correctly and within governor limits.

Escalation tests ensure that time-driven workflows behave predictably without introducing impractical delays into your test suite.

Escalation scenarios to test

- Standard escalation after 48 hours
- Multiple pending approvals, some escalate, some don't (based on submission time)
- Escalated approval then approved by original approver
- Escalated approval approved by escalation approver
- Deal characteristics change after escalation (re-routing required)

Parallel Approval Validation

When multiple approvers must respond, validate complex state management:

Both approvers approve First approver approves → Process waits for second approver Second approver approves → Process completes Verify correct final approval state Validate notifications sent to all stakeholders	**Split decision (one approves, one rejects)** Approver A approves first Approver B rejects Entire process stops (rejection wins) Both approvers notified of final decision Rep receives rejection with both approvers' comments
Approval order variations: Test both possible approval orders (A then B, B then A) Results should be identical regardless of order Validate state consistency	**Reassignment during parallel approval:** Approver A approves Approver B reassigns to Approver C Approver C must still approve for completion Validate approval chain tracking

Integration Synchronization Testing

Approved discounts must flow to external quote/CPQ system:

Successful sync scenarios

- Approval completes → API call to CPQ system
- Discount percentage transfers correctly
- Opportunity ID links to quote record
- CPQ system acknowledges receipt
- Salesforce updates with sync confirmation

Error handling scenarios

- CPQ system unavailable → Retry logic executes

- API returns error → Salesforce logs error and notifies admin
- Partial data sync → Transaction rollback or compensation
- Authentication failure → Error handling and admin notification

Data validation scenarios

- Discount percentage exceeds CPQ system maximum → Validation error before sync
- Required fields missing → Sync blocked with clear error message
- Opportunity status conflicts with CPQ requirements → Business rule validation

Automation Approach

Approval Routing Logic Testing

Unit tests validate that the approval process selection and routing work correctly without actually submitting approvals:

- **Test tier boundaries**: create opportunities at different thresholds (auto-approval, manager, director, executive) to confirm routing logic.
- **Verify approver assignment**: ensure that records route to the correct user role (e.g., Sales Manager, Director).
- **Use helper methods or factories** to generate consistent test data for each tier scenario.
- **Leverage ApprovalHelper or equivalent utilities** to simulate submission without requiring full approval execution.
- **Assert routing outcomes** by checking assigned approvers in `ProcessInstanceWorkitem` rather than relying on UI.
- **Include edge cases**: discounts just below or above thresholds, missing approver roles, or overlapping conditions.
- **Isolate execution with Test.startTest / Test.stopTest** to ensure routing logic runs predictably in test context.

Approval routing tests confirm that business rules for tiered approvals are enforced consistently and without manual intervention.

UI automation for approval user experience	**API testing for CPQ integration**	**Manual testing for edge cases**

Test the actual approver experience through Salesforce UI: Manager receives approval notification Clicks email link to approval page Reviews opportunity details Adds approval comment Clicks "Approve" or "Reject" Sees confirmation message Verifies notification sent to next approver/rep	**Validate integration endpoint independently:** Create test approval scenario Trigger approval completion Monitor API call to CPQ system Verify request payload format Test response handling (success and error) Validate retry logic with service virtualization	**Complex scenarios benefit from human testing:** Multiple simultaneous deal approvals from same rep Approval during Salesforce platform maintenance Approver with multiple pending approvals across different processes Deals that change significantly during approval (product mix, customer change)

Common Pitfalls to Avoid

Pitfall #1: Only testing happy path approvals. Most testing validates successful approvals. Real value comes from testing rejections, reassignments, escalations, and error scenarios.

Pitfall #2: Ignoring parallel approval edge cases. Testing assumes sequential approval. Parallel approval scenarios create race conditions and state management challenges that require explicit testing.

Pitfall #3: Not testing approval process changes. When approval criteria or routing logic changes, existing in-flight approvals may behave unexpectedly. Test mid-process changes.

Pitfall #4: Overlooking notification testing. Email notifications are critical for approval workflow effectiveness. Verify that correct people receive correct notifications at correct times.

Pitfall #5: Assuming integration will work. Integration with CPQ/quote systems frequently has issues. Test integration explicitly, including all error scenarios.

Scenario 3: Data Migration Testing

Data migration to Salesforce — whether from legacy CRM, homegrown systems, or spreadsheets — represents one of the highest-risk phases of any Salesforce implementation. Poor migration testing leads to corrupted customer relationships, broken sales processes, and user adoption failure.

Business Process Overview

Typical enterprise migration scenario:

Source system: Legacy CRM with 10+ years of customer data
Data volume: 500,000 accounts, 2 million contacts, 5 million opportunities (historical)
Custom fields: 200+ custom fields requiring mapping and transformation
Integrations: 15 external systems that reference customer data
Timeline: Migration must complete over weekend to minimize business disruption

Migration process

1. **Data extraction** from legacy system
2. **Data cleansing** to fix quality issues
3. **Data transformation** to match Salesforce data model
4. **Data validation** before loading
5. **Initial load** to Salesforce sandbox
6. **Testing and validation** with sample users
7. **Production migration** during cutover window
8. **Post-migration validation** and issue resolution

Testing Challenges

Data volume complexity. Testing with sample data (1,000 records) doesn't reveal issues that only appear with production scale (500,000 records). Performance characteristics change dramatically.

Relationship preservation. Legacy systems organize data differently than Salesforce. Account hierarchies, contact relationships, opportunity associations must translate correctly.

Data quality issues. Legacy data contains duplicates, incomplete records, invalid values, and inconsistent formatting. Migration testing must validate that cleansing logic works and that poor-quality data doesn't break Salesforce.

Business continuity validation. Migration isn't just technical success (data loads without errors). It's business success (users can continue working, reports still run, processes still function).

Integration impact. External systems reference customer data. After migration, these integrations must still work despite different IDs, different data structures, and potentially different API endpoints.

Rollback complexity. If migration fails, returning to the legacy system must be possible. Testing rollback procedures before production migration is critical.

What Comprehensive Testing Validates

Data integrity and completeness

Validate that every record migrates accurately and completely

Record count reconciliation	**Field-level data validation**
Source system: 500,000 accounts → Salesforce: 500,000 accounts (or documented exceptions) Contact counts match per account Opportunity counts match per account Related records (tasks, notes, attachments) migrate completely	For sample records, validate field-by-field accuracy: Account Name, Industry, Phone, Address fields match source Contact names, emails, titles, relationships preserved Opportunity amounts, stages, close dates accurate Custom fields populated correctly after transformation Picklist values are mapped appropriately Date/time fields converted to correct timezone
Relationship preservation	**Data quality validation**
Test that record relationships maintain integrity:	Verify that cleansing rules applied correctly:

Parent/child account hierarchies preserved Contacts linked to correct accounts Opportunities associated with correct accounts and contacts Contact roles on opportunities maintained Task and event associations preserved	Duplicate accounts merged according to rules Incomplete records handled appropriately (rejected, completed, flagged) Invalid data transformed or defaulted correctly Email addresses validated and standardized Phone numbers formatted consistently Addresses standardized to correct format

Business logic compatibility

Migrated data must work with Salesforce automation:

Workflow and validation rules	**Reporting and dashboards**	**User permissions and sharing**
Historical opportunities don't trigger "new opportunity" workflows Migrated data passes validation rules (or documented exceptions exist) Process Builder and Flow automations handle migrated data correctly Email alerts don't spam customers about historical records	Standard sales reports work with migrated data Custom reports produce expected results Dashboards render correctly with production data volume Historical trend analysis shows correct patterns	Record ownership assignments correct Territory assignments appropriate Sharing rules apply correctly to migrated data Users can access records they should access

Performance Validation with Production Volumes

Testing with realistic data volumes reveals performance issues:

Page load performance	**Report generation performance**
Account pages load acceptably with 200+ related contacts Opportunity pages render with 50+ related opportunities List views perform adequately with hundreds of thousands of records Search results return within acceptable timeframes	Standard reports complete within timeout limits Complex custom reports with large datasets don't fail Dashboard refreshes complete successfully Scheduled reports generate and deliver correctly
Bulk operation performance	**Governor limit validation**
Mass updates to thousands of records complete successfully Bulk delete operations work without hitting limits Data loader operations handle large files appropriately	Triggers process bulk data updates without hitting SOQL limits Workflows and Process Builder handle volume appropriately Batch jobs complete within execution limits

Integration Continuity Testing

External systems must continue working after migration:

API authentication and connectivity	**Data synchronization validation**

External systems can authenticate to Salesforce with new credentials API versions compatible between Salesforce and external systems Network connectivity and firewall rules configured correctly	ERP system still syncs customer data correctly Marketing automation platform maintains contact sync Support system accesses case and account information Analytics platform pulls Salesforce data successfully
ID mapping for integrations	**Real-time integration testing**
External systems that stored old CRM IDs can map to new Salesforce IDs Integration middleware updated with new ID mappings Historical transactions maintain referential integrity	New transactions trigger integrations correctly Historical data doesn't trigger duplicate integration events Error handling works appropriately

User Acceptance Testing for Migration

Business users validate that migrated data supports their work:

Sales rep UAT scenarios	**Sales manager UAT scenarios**	**Customer service UAT scenarios**
Find my accounts and contacts Access opportunity history for customer conversations	Review team's opportunities and pipeline Generate forecast reports with historical trends	Look up customer accounts to log cases Access customer interaction history

Generate quotes using migrated customer data Update account information and verify changes persist Run territory reports and verify accuracy	Analyze won/lost opportunity patterns Validate data for executive reporting	Update contact information Review case history across old and new systems

Each UAT scenario should use actual migrated data in realistic business workflows.

Rollback Testing

Before production migration, validate rollback procedures:

Rollback scenario testing	**Go/no-go decision criteria**
Document step-by-step rollback process Execute rollback in sandbox environment Measure rollback time requirements Verify legacy system restores to pre-migration state Validate that users can resume work in legacy system Test data integrity after rollback	<5% data discrepancy in record counts (or other threshold) All critical business processes functional in UAT Key integrations tested successfully Performance meets defined SLAs Rollback procedures documented and tested Executive approval obtained

Automation Approach for Migration Testing

ETL Testing for Data Validation

Tools like QuerySurge, iCEDQ, or custom scripts validate data transformations during migrations:

- **Row count validation**: compare record counts between source and destination systems to detect mismatches.
- **Field-level checks**: validate that individual fields are transformed and mapped correctly according to business rules.
- **Sample record verification**: select random samples from source data and compare them against destination records.
- **Relationship integrity**: ensure parent-child relationships and lookups are preserved after migration.
- **Error logging**: capture mismatches or anomalies with clear reporting for troubleshooting.
- **Automation scripts**: use SQL queries or custom frameworks to automate repetitive validation tasks.
- **Incremental testing**: validate data in batches to identify issues early and reduce risk in large migrations.

ETL validation ensures that migrated data is complete, accurate, and consistent with business requirements.

Automated validation scripts compare source and destination data programmatically, checking thousands of records faster and more accurately than manual validation.

API testing for integration validation	**Salesforce test automation for business processes**	**Manual validation for data quality**
Test integration endpoints with migrated data: Query Salesforce via API to verify data accessibility Test CRUD operations against migrated records Validate that external system queries return expected data	**UI automation verifies that users can do key tasks:** Login and navigate to accounts Search for specific customers Create opportunities from migrated accounts Generate reports and dashboards	**QA and business users manually review:** Random sample of migrated records for accuracy Edge cases and unusual data patterns User experience with actual workflows

Verify ID mapping functions correctly	Update records and verify persistence	Report outputs and dashboard displays

Common Pitfalls to Avoid

Pitfall #1: Testing only with sample data. Small data sets hide performance issues, governor limit problems, and UI rendering issues that only appear at production scale. Always test with production-volume data.

Pitfall #2: Focusing on technical success over business continuity. Data that loads successfully but doesn't support business processes is a failed migration. Test actual business workflows, not just data presence.

Pitfall #3: Inadequate UAT time. Business users need sufficient time to validate migrated data in realistic scenarios. Rushing UAT leads to missed issues that surface post-migration.

Pitfall #4: Not testing rollback. Assuming migration will succeed without tested rollback procedures is risky. Practice rollback before production migration.

Pitfall #5: Ignoring data quality issues. Legacy data quality problems don't disappear during migration. Test cleansing logic explicitly and have clear rules for handling poor-quality data.

Pitfall #6: Underestimating integration testing. External systems often have hard dependencies on customer data structure and IDs. Integration testing can't be an afterthought.

Scenario 4: Integration Testing Across Multiple Systems

Modern Salesforce implementations rarely operate in isolation. The most complex testing scenarios involve validating processes that span Salesforce and multiple external systems — each with different update schedules, API versions, and ownership teams.

Business Process Overview

Order-to-cash process spanning five systems:

1. **Salesforce** - Opportunity and quote management
2. **CPQ System** (Salesforce CPQ or external) - Product configuration, pricing, quote generation
3. **ERP System** - Order processing, inventory management, fulfillment
4. **Payment Gateway** - Payment processing and authorization

5. **Customer Data Platform** - Unified customer view across all touchpoints

Complete workflow	
Sales rep creates opportunity in Salesforce Rep configures product bundle in CPQ system CPQ generates quote with pricing and terms Customer approves quote Quote converts to order in ERP system	ERP checks inventory and initiates fulfillment Payment gateway processes payment Order fulfillment completes Customer data platform updates with purchase information Salesforce opportunity closes as won

At each handoff between systems, data transformations occur, API calls execute, and potential failures can disrupt the entire process.

Testing Challenges

Cross-system orchestration. Testing requires coordinating environments across multiple teams. Each system has its own sandbox strategy, refresh schedule, and availability windows.

Asynchronous processing. Many integration points use queued jobs, batch processes, or event-driven architectures. Tests must handle varying completion times and out-of-order processing.

API version compatibility. Salesforce updates quarterly. Your ERP might update annually. CPQ system updates monthly. API versions across systems may not align, creating compatibility issues.

Error propagation complexity. When an order fails in the ERP system, how does that error flow back to Salesforce? How does it affect the CPQ system? How do users get notified? Error scenarios multiply across system boundaries.

Environment availability. External systems aren't always available for testing. Scheduled maintenance, team priorities, and conflicting test schedules create availability gaps.

Data consistency across systems. The same customer exists in Salesforce, ERP, payment gateway, and CDP. Keeping test data consistent across all systems is challenging.

What Comprehensive Testing Validates

API contract compliance

Validate that each integration point adheres to expected API contracts:

Salesforce to CPQ integration	**CPQ to ERP integration**
Request payload structure matches CPQ expectations Required fields populated correctly Data types match (strings, numbers, dates formatted appropriately) Authentication headers correct API version specified and supported	Order data transforms to ERP format correctly Product SKUs map between systems appropriately Pricing and tax calculations preserve accuracy Customer ID mapping works correctly
ERP to payment gateway	**Cross-system to CDP**
Payment requests contain required billing information Amount and currency match order total Transaction IDs link back to orders correctly	Customer activity events flow to CDP from all systems Event timing preserves sequence Customer identifiers consistent across sources

Data synchronization accuracy

Test that data updates propagate correctly across systems:

Customer data sync scenarios	**Product data sync scenarios**

Customer updates address in Salesforce → Address updates in ERP and CDP ERP updates credit limit → Salesforce reflects new limit Payment gateway updates payment method → Salesforce and ERP reflect change CDP enriches customer data → Enrichment flows back to Salesforce	CPQ adds new product → Salesforce product catalog updates ERP updates product availability → CPQ pricing engine reflects stock status Salesforce updates product description → Marketing systems receive updates
Transaction data sync scenarios	**Conflict resolution testing**
Order created in ERP → Salesforce opportunity updates to "Closed Won" Payment processed in gateway → ERP and Salesforce reflect payment Fulfillment completes in ERP → Customer notification triggered via Salesforce	Simultaneous customer updates in multiple systems → Last write wins? Merge logic? Conflict notification? Race conditions between systems → Test timing variations Partial sync failures → Compensation logic executes

End-to-End Workflow Validation

Test complete business processes across all systems:

Happy path scenario

Complete order-to-cash process with no errors:

1. Create opportunity in Salesforce with complete customer data	5. Customer accepts quote (simulated) 6. Quote converts to order in ERP	10. Order fulfillment initiated in ERP 11. Fulfillment completes

2. Configure product bundle in CPQ (3 products, volume discount applied) 3. Generate quote with approval workflow 4. Quote approved by sales manager	7. ERP validates inventory availability 8. Payment authorization requested from gateway 9. Payment approved (test credit card)	12. Salesforce opportunity updates to "Closed Won" 13. CDP records complete customer journey

Validate at each step:

- Data accuracy maintained
- Timing expectations met
- Notifications sent appropriately
- Records linked correctly across systems

Error scenario testing

Test each potential failure point and error handling:

CPQ system unavailable	**ERP rejects order**
Salesforce queues quote creation request Retry logic executes after timeout User receives notification of delay Quote eventually creates when CPQ returns	Invalid product configuration detected Error message returns to Salesforce Order creation stops gracefully Sales rep notified with specific error details Opportunity remains in "Quote Generated" stage
Payment authorization fails	**System timeout scenarios**
Gateway returns "Insufficient funds" error ERP pauses order fulfillment	API call exceeds timeout threshold Calling system logs timeout error

Salesforce updates opportunity with payment issue Customer service notified for follow-up Retry mechanism available for re-authorization	Retry logic executes appropriate number of times Fallback procedure activates after max retries Human intervention triggered when needed
Partial fulfillment scenario	
• ERP ships partial order (2 of 3 products) • Salesforce reflects partial fulfillment status • Customer notified of split shipment • Remaining items tracked for future fulfillment	

Performance and Scalability Testing

Integration performance affects user experience and business throughput:

Synchronous call performance:

- CPQ quote generation completes within user-acceptable timeframe (<10 seconds)
- Payment authorization responds quickly (<5 seconds)
- Real-time inventory check doesn't block Salesforce UI

Asynchronous processing performance:

- Order creation in ERP completes within SLA (e.g., 15 minutes)
- Batch data synchronization completes overnight
- Event processing handles peak volumes without backlog

Bulk operation handling:

- Mass quote generation for 100 opportunities
- Bulk order import from external system
- End-of-month batch processing completing successfully

Rate limit compliance:

- Integration respects Salesforce API limits (15,000-100,000 per 24 hours)
- External system rate limits not exceeded
- Retry logic with exponential backoff when limits approached

Service Virtualization for Independent Testing

External systems aren't always available. Service virtualization enables testing without dependencies:

Mock external systems:

Create simulated versions of CPQ, ERP, and payment gateway that:

- Respond to API calls with expected data formats
- Simulate various response scenarios (success, errors, timeouts)
- Don't require actual external system availability
- Support faster test execution (no network latency)

Tools for service virtualization:

- WireMock (open source, Java-based)
- Postman Mock Server (part of Postman platform)
- Mockoon (open source, easy setup)
- Parasoft Virtualize (enterprise-grade)

Service virtualization lets you test Salesforce integration logic independently while external systems are unavailable or undergoing maintenance.

Automation Approach

API testing as foundation:

Most integration testing happens at API level:

- Faster than UI testing
- More stable (no UI changes breaking tests)
- Direct validation of integration contracts
- Easier to simulate error conditions

Use REST Assured, Postman, or similar tools to build comprehensive API test suites.

UI testing for critical user workflows:

Reserve UI automation for scenarios where user experience matters:

- Sales rep creating quote through Salesforce UI

- Viewing order status that syncs from ERP
- Error message display when integration fails

UI tests validate that users see correct information and appropriate error messages, not just that APIs function correctly.

Integration monitoring in production:

Testing can't catch everything. Production monitoring complements testing:

- API call success rates
- Response time trends
- Error frequency and types
- Data sync lag times

Monitoring alerts teams to integration issues that testing didn't anticipate.

Common Pitfalls to Avoid

Pitfall #1: Testing only in ideal conditions. Most integration tests assume all systems are available and responsive. Real value comes from testing failures, timeouts, and degraded performance.

Pitfall #2: Ignoring timing and sequencing. Asynchronous integrations create race conditions and sequencing issues. Test explicitly for timing variations.

Pitfall #3: Not coordinating test data across systems. When customer "ABC Corp" has different IDs or attributes in different systems, integration tests fail for wrong reasons.

Pitfall #4: Overlooking API version compatibility. Systems update independently. Test that integration continues working across API version changes.

Pitfall #5: Insufficient error scenario coverage. Happy path testing misses most integration issues. Error handling is where integrations commonly fail.

Pitfall #6: Dependency on external team schedules. Relying on other teams to make test environments available creates bottlenecks. Service virtualization reduces these dependencies.

Scenario 5: Salesforce Release Testing

Salesforce platform updates happen three times per year. Each release brings new features, bug fixes, and potential compatibility issues with your customizations. Pre-release testing in preview environments catches problems before they affect production users.

Business Process Overview

Salesforce release testing process:

1. **Release notes review** (4-6 weeks before release) - Identify features affecting your org
2. **Preview sandbox access** (3-4 weeks before release) - Salesforce provides preview instances with new release
3. **Impact assessment** (2-3 weeks before release) - Test customizations against new features
4. **Regression testing** (1-2 weeks before release) - Validate that existing functionality still works
5. **Fix implementation** (if needed) - Address compatibility issues discovered in testing
6. **Production release** (scheduled date) - Platform updates automatically
7. **Post-release validation** - Verify production operates correctly

The testing window is compressed: 3-4 weeks to validate an entire year's worth of customizations against platform changes.

Testing Challenges

Unknown changes. Salesforce makes thousands of updates per release. Release notes document major features, but subtle changes can affect customizations in unexpected ways.

Limited time window. Preview environments are only available for 3-4 weeks. Complete regression testing must fit this window.

Platform changes you didn't request. Unlike your customizations where you control timing, platform updates happen whether you're ready or not. Testing is defensive, not proactive.

Combination effects. New platform features might conflict with old customizations in ways that aren't obvious. Multi-year-old triggers might interact badly with new Process Builder capabilities.

Mobile app updates. Platform releases affect Salesforce mobile app. Mobile testing often gets overlooked but field users depend on mobile functionality.

What Comprehensive Testing Validates

Existing customization compatibility:

Test that your custom code and configurations still work:

Apex trigger validation:

- All triggers execute without errors
- Business logic produces expected results

- Governor limits aren't affected by platform changes
- Error handling still functions correctly

Workflow and Process Builder validation:

- Automations still fire on expected conditions
- Email alerts and field updates work correctly
- Time-based workflows execute on schedule
- New platform features don't conflict with existing automation

Lightning component validation:

- Custom components render correctly
- Event handling still works
- Data binding maintains functionality
- Shadow DOM changes don't break components

Visualforce page validation:

- Pages render without errors
- Controller logic executes correctly
- JavaScript functions properly
- Mobile rendering works (if applicable)

Integration validation:

- API versions remain compatible
- Callouts execute successfully
- Authentication mechanisms work
- Data formats haven't changed

New Feature Evaluation

Each release introduces capabilities you might want to adopt:

Feature identification:

- Review release notes for features relevant to your business
- Identify enhancements to existing functionality you use
- Note deprecated features that might affect you

Adoption testing:

- Test new features in preview sandbox with representative data
- Evaluate compatibility with existing customizations
- Assess performance impact

- Document configuration requirements

Business value assessment:

- Does this feature solve existing pain points?
- What effort is required to implement?
- What user training is needed?
- What ROI can we expect?

Example: Testing new Flow feature

If Salesforce introduces new Flow capabilities:

- Build test flows using new features
- Validate interaction with existing Process Builder automations
- Test performance with production data volumes
- Determine if new features can replace existing customizations
- Assess migration effort from old to new approach

Regression Test Suite Execution

Run comprehensive regression tests against preview environment:

Critical business process testing:

- Lead-to-opportunity conversion
- Quote and approval workflows
- Case escalation and routing
- Order processing integrations
- Standard and custom reports

Test each process end-to-end to verify no platform changes broke functionality.

Data operation testing:

- CRUD operations on standard and custom objects
- Bulk data imports and updates
- Data loader functionality
- List view and search performance

User interface testing:

- Page layouts render correctly
- Lightning pages display properly
- Mobile app functionality works
- Record detail pages load acceptably

Reporting and analytics:

- Standard reports generate correctly
- Custom reports produce expected results
- Dashboards render without errors
- Scheduled reports deliver successfully

Governor Limit Validation

Platform updates can affect resource consumption:

CPU time monitoring:

- Complex triggers might consume more CPU time with platform changes
- Validate that critical processes stay within limits
- Test bulk operations explicitly

SOQL query usage:

- Platform features might use additional queries
- Ensure triggers and classes don't exceed limits
- Test with production-scale bulk operations

API call consumption:

- New platform features might consume API calls
- Verify integration processes don't hit limits
- Test batch processes and scheduled jobs

Heap size validation:

- Large object collections might behave differently
- Test memory-intensive operations
- Validate batch processing with large datasets

Mobile Experience Testing

Platform releases affect Salesforce mobile app:

Mobile functionality validation:

- Field service technician workflows
- Sales rep opportunity management
- Manager approval processes
- Offline data access and sync

Mobile UI validation:

- Page layouts render on mobile devices
- Lightning pages adapt to screen sizes
- Custom components work on mobile
- Performance acceptable on cellular networks

Mobile-specific features:

- Location services functionality
- Camera integration for attachments
- Barcode scanning (if used)
- Mobile-only actions work correctly

Automation Approach for Release Testing

Automated regression suite:

Comprehensive automated tests enable rapid validation:

```
# Release testing automation approach (pseudocode)

def run_release_regression_suite():
    results = []

    # Unit tests
    results.append(run_apex_tests())

    # Integration tests
    results.append(test_external_integrations())

    # UI tests - critical paths
    results.append(test_lead_conversion_ui())
    results.append(test_opportunity_close_ui())
    results.append(test_case_escalation_ui())

    # Report generation
    results.append(test_standard_reports())
    results.append(test_dashboard_rendering())

    # Performance tests
    results.append(test_bulk_operations())

    generate_release_test_report(results)
```

```
identify_failures_for_investigation(results)
```

Automated regression suites that take 2-3 weeks manually can execute in hours with automation.

Parallel test execution:

Run tests simultaneously to fit compressed timeline:

- Unit tests in one thread
- Integration tests in another
- UI tests in multiple parallel browsers
- Performance tests in dedicated environment

Continuous monitoring during preview:

Set up monitoring in preview sandbox:

- Error logs and exceptions
- Performance degradation
- API call patterns
- User feedback from UAT participants

Issue Remediation Strategy

When testing reveals compatibility issues:

Issue categorization:

- **Critical** - Breaks business-critical processes, blocks production use
- **High** - Significant functionality impaired, workaround possible
- **Medium** - Minor functionality affected, limited user impact
- **Low** - Cosmetic issues, nice-to-have fixes

Remediation approaches:

For critical issues:

- Immediate fix required before production release
- Engage Salesforce support if platform bug
- Implement workaround if fix not possible
- Consider release delay request (rare but possible)

For high/medium issues:

- Fix if time permits during preview window

- Document workarounds for users
- Schedule fix in post-release sprint
- Communicate impact to stakeholders

For low issues:

- Log for future resolution
- No immediate action required
- Include in routine maintenance backlog

Production Release Day Validation

After platform updates to production:

Smoke testing:

- Critical workflows function
- Users can log in and access data
- Integrations operating
- No widespread errors in logs

Monitoring heightened:

- Watch error logs closely
- Monitor user feedback channels
- Track performance metrics
- Respond quickly to issues

Rollback planning:

Unlike custom deployments, platform releases can't be rolled back. Instead:

- Document issues discovered
- Implement quick fixes or workarounds
- Engage Salesforce support for platform bugs
- Communicate with users about known issues

Common Pitfalls to Avoid

Pitfall #1: Skipping preview testing. Assuming Salesforce releases won't affect your org is risky. Even minor platform changes can break customizations.

Pitfall #2: Testing only happy paths. Focus regression testing on edge cases and error scenarios where platform changes create unexpected interactions.

Pitfall #3: Ignoring mobile testing. Field users depend on mobile functionality. Mobile issues discovered post-release disrupt business operations.

Pitfall #4: Not involving business users. UAT during preview window catches usability issues that technical testing misses.

Pitfall #5: Insufficient time allocation. Trying to complete regression testing in 1 week when it requires 2-3 weeks leads to missed issues.

Pitfall #6: No post-release validation. Platform changes sometimes behave differently in production than preview. Post-release monitoring is essential.

Scenario 6: Performance Testing Under Load

Performance testing validates that your Salesforce implementation handles production-scale loads. Issues that don't appear in development sandboxes with sample data emerge when hundreds of users access millions of records simultaneously.

Business Process Overview

Performance testing scenario: Enterprise sales organization

User base: 500 Salesforce users (300 sales reps, 100 sales managers, 100 support staff)
Data volume: 2 million accounts, 8 million contacts, 10 million opportunities (5 years historical)
Peak usage: Monday mornings (9-11am), month-end closing (last 3 days of month), quarterly business reviews
Critical workflows: Opportunity pipeline reviews, quote generation, forecast submissions, executive dashboards

Performance testing validates system responsiveness during these peak periods and with production data volumes.

Testing Challenges

Realistic load simulation. Users don't just click buttons repeatedly. They navigate, search, read, update, generate reports. Simulating realistic user behavior is complex.

Governor limit constraints. Salesforce platform limits create unique performance challenges. Tests must validate both response time and resource consumption.

Data volume effects. A Lightning page that loads in 2 seconds with 100 records might take 30 seconds with 500,000 records. Volume testing reveals these issues.

Concurrent user patterns. 500 users logging in simultaneously (Monday morning) creates different load than 500 users distributed throughout the day.

Third-party app impact. AppExchange packages and managed packages consume resources. Performance testing must account for their overhead.

What Comprehensive Performance Testing Validates

Page load performance under load:

Measure Lightning page response times with concurrent users:

Normal load (50 concurrent users):

- Account detail page: <3 seconds
- Opportunity list view: <4 seconds
- Custom dashboard: <5 seconds
- Report generation: <8 seconds

Peak load (300 concurrent users):

- Account detail page: <5 seconds
- Opportunity list view: <7 seconds
- Custom dashboard: <10 seconds
- Report generation: <15 seconds

Stress load (500 concurrent users - theoretical maximum):

- All pages should still load, though slower
- No system errors or crashes
- Graceful degradation of performance

Critical workflows under load:

Test business processes with realistic concurrency:

Opportunity pipeline review (typical Monday morning):

- 100 sales managers simultaneously access opportunity reports
- Each manager has 50-200 opportunities in their view
- Opportunity stages and amounts update in real-time
- Page refreshes complete within acceptable timeframes

Month-end forecast submission:

- 300 sales reps submit forecasts in 3-day window

- System handles bulk forecast updates
- Approvals process without delay
- Reports reflect real-time submission status

Executive dashboard access:

- 20 executives simultaneously view company-wide dashboards
- Dashboards aggregate data from millions of records
- Real-time calculations complete successfully
- Drill-down functionality performs adequately

Report and Dashboard Performance

Reporting performance often degrades with data volume:

Standard report performance:

- Opportunity pipeline report (all opportunities, current year): <10 seconds
- Account list with custom filters: <8 seconds
- Contact relationship reports: <12 seconds

Complex custom report performance:

- Multi-object joins (Account + Opportunity + Product): <20 seconds
- Historical trend analysis (5 years data): <30 seconds
- Territory-based aggregations: <15 seconds

Dashboard rendering:

- 5-component dashboard: <10 seconds initial load
- Dashboard refresh: <15 seconds
- Scheduled dashboard email: Completes within timeout limits

Report builder performance:

- Creating new reports responsive
- Filter changes update preview quickly
- Large result sets paginate appropriately

Bulk Operation Performance

Large data operations reveal scalability issues:

Data loader operations:

- 50,000 record update: Completes in <10 minutes

- 100,000 record insert: Completes successfully without partial failures
- Bulk delete of 25,000 records: Executes within platform limits

Mass update performance:

- List view mass update (200 records): <30 seconds
- Territory reassignment (1,000 accounts): <5 minutes
- Bulk email sends (500 contacts): Completes without timeout

Batch job performance:

- Nightly data sync jobs complete before business hours
- Scheduled maintenance jobs don't impact daytime performance
- Large batch sizes process without hitting governor limits

Governor Limit Validation at Scale

Performance testing identifies governor limit risks:

SOQL query limits:

- Complex triggers stay within 100 query limit during bulk operations
- Nested loops don't create excessive queries
- Inefficient queries identified and optimized

CPU time limits:

- Complex business logic stays within 10,000ms synchronous limit
- Batch processes stay within 60,000ms asynchronous limit
- Trigger optimization prevents timeout errors

Heap size limits:

- Large object collections don't exceed 6MB synchronous limit
- Bulk processing manages memory efficiently
- No out-of-memory errors during peak loads

API call limits:

- Integration processes stay within daily limits (15,000-100,000)
- Peak integration periods don't exhaust allocation
- Rate limiting logic prevents limit violations

Search Performance

Salesforce search with large datasets requires specific testing:

Global search performance:

- Keyword searches return results in <3 seconds
- Recent items populate quickly
- Search suggestions responsive

Lookup field performance:

- Lookup dialogs on records with thousands of related items load acceptably
- Search-within-lookup responsive
- Recently viewed items appear quickly

List view search and filtering:

- Filter application on large lists (<5 seconds)
- Column sorting on large datasets works
- Pagination through thousands of records functions

Mobile Performance Testing

Mobile app performance with limited bandwidth:

3G network performance:

- Page loads complete within acceptable mobile timeframes
- Offline data sync works reliably
- App remains responsive despite slow connection

Offline functionality:

- Critical records cache locally
- Offline edits sync when connection restored
- Conflict resolution handles network interruptions

Mobile data usage:

- App doesn't consume excessive cellular data
- Image and attachment handling optimized
- Background sync respectful of data limits

Automation Approach

Load testing tools:

JMeter, LoadRunner, or Salesforce-specific load testing tools:

```
# JMeter test plan structure (conceptual)

Thread Group: Simulate 300 concurrent users
  - Login to Salesforce
  - Navigate to opportunity list view
  - Search for specific opportunities
  - Open opportunity detail page
  - Update opportunity stage
  - Save changes
  - Generate opportunity report
  - Logout

Duration: 30 minutes sustained load
Ramp-up: Gradually increase from 0 to 300 users over 5 minutes

Measure:
  - Response times for each step
  - Error rates
  - Throughput (transactions per second)
  - Resource consumption on Salesforce side
```

Realistic user behavior simulation:

Users don't click as fast as scripts can. Include think time:

- Page viewing: 5-10 seconds
- Data entry: 15-30 seconds
- Decision making: 10-20 seconds

Realistic load testing provides meaningful performance data.

Production monitoring for baseline:

Before load testing, measure production performance:

- Current page load times
- Report generation duration
- Peak usage patterns
- Resource consumption trends

Load testing validates that performance meets or exceeds production baseline.

Performance Metrics and SLAs

Define acceptable performance levels:

Page load SLAs:

- Critical pages (opportunity, account): <5 seconds @ peak load
- Standard pages (contacts, tasks): <8 seconds @ peak load
- Reports: <15 seconds @ peak load
- Dashboards: <10 seconds initial load @ peak load

Transaction SLAs:

- Record creation: <3 seconds
- Record updates: <2 seconds
- Record deletion: <2 seconds
- Search results: <4 seconds

Bulk operation SLAs:

- 1,000 records: <2 minutes
- 10,000 records: <10 minutes
- 50,000 records: <30 minutes

When performance testing reveals SLA violations, optimization is required.

Common Pitfalls to Avoid

Pitfall #1: Testing with sample data. Performance characteristics change dramatically between 1,000 and 1,000,000 records. Always test with production-scale data.

Pitfall #2: Unrealistic load simulation. Tests that just hammer login endpoints don't represent real user behavior. Simulate realistic workflows.

Pitfall #3: Ignoring third-party apps. AppExchange packages and managed packages affect performance. Test with actual org configuration, not clean sandbox.

Pitfall #4: Not testing mobile performance. Field users experience different performance characteristics on cellular networks with mobile app.

Pitfall #5: Testing only average load. Peak load testing reveals scalability limits. Test Black Friday equivalents, not average Tuesday.

Pitfall #6: No baseline metrics. Without knowing current production performance, you can't determine if test results are acceptable.

Conclusion: Applying These Scenarios to Your Testing

These six scenarios represent common but complex testing challenges in Salesforce implementations. Each demonstrates principles that apply broadly:

Start with business impact. Test processes that affect revenue, customer relationships, and business operations first. Perfect testing of critical workflows beats mediocre testing of everything.

Layer your testing approach. Unit tests catch code-level issues. Integration tests validate cross-system flows. UI tests verify user experience. Performance tests reveal scalability limits. Each layer serves different purposes.

Design for maintenance. Tests you write today must work next quarter after platform updates. Modular, well-structured tests survive changes better than brittle, tightly-coupled tests.

Balance automation and manual testing. Automation handles repetitive validation efficiently. Manual testing brings human judgment to complex scenarios. Both are necessary.

Test in realistic conditions. Sample data, perfect network conditions, and isolated testing environments don't reveal production issues. Test with production-scale data, simulate failures, and validate error handling.

Adapting Scenarios to Your Org

Your Salesforce implementation is unique. These scenarios provide frameworks, not prescriptions:

Identify your critical processes. What workflows generate revenue? What processes serve customers? What integrations enable business operations? Focus testing there.

Understand your complexity. Complex customizations require more testing. Heavy integrations demand more validation. Large data volumes need performance testing. Assess your specific challenges.

Match automation to team skills. Tools and approaches that work for organizations with dedicated automation engineers won't work for small teams. Choose automation that your team can maintain.

Prioritize based on risk. Not everything can be tested comprehensively. Assess business impact of failures and test high-risk areas thoroughly.

Next Steps

Use these scenarios as templates:

Step #1. Select 1-2 scenarios most relevant to your business

Step #2. Adapt test cases to your specific customizations and workflows

Step #3. Build automation for repetitive validation in those scenarios

Step #4. Execute tests before major releases and deployments

Step #5. Expand coverage incrementally to additional scenarios

Testing is not one-time effort. It's ongoing investment in system reliability, business continuity, and user confidence.

The scenarios in this guide demonstrate that effective Salesforce testing requires understanding business processes, technical architecture, integration patterns, and user behavior. No single tool or approach solves all testing challenges. Success comes from strategic thinking, appropriate automation, and continuous improvement.

About TestFort

TestFort specializes in QA and testing services for complex Salesforce implementations. Our team has built testing frameworks, automated regression suites, and validated migrations across industries including fintech, healthcare, SaaS, and enterprise technology.

We understand that Salesforce testing is complex, not just difficult. It requires specialized expertise, purpose-built approaches, and deep understanding of the platform's unique characteristics.

Our Salesforce testing services include:

- **Test strategy development** - Identifying high-value testing opportunities specific to your org
- **Test automation implementation** - Building maintainable frameworks that survive platform updates
- **Migration testing** - Validating data integrity, performance, and business continuity
- **Integration testing** - Verifying cross-system workflows and API reliability
- **Performance testing** - Ensuring scalability with production data volumes
- **Release testing** - Pre-validating platform updates before they affect your business

Why organizations choose TestFort:

✓ **Salesforce-specific expertise** - We understand Lightning components, Shadow DOM, governor limits, and platform update cycles

✓ **Proven testing frameworks** - Our approaches have been refined across dozens of implementations

✓ **Balanced automation strategy** - We automate what provides ROI and keep manual testing where it adds value

✓ **Transparent communication** - You'll always know what we're testing, what we've found, and what it means for your business

Ready to strengthen your Salesforce testing?

Whether you need comprehensive testing strategy, automation implementation, or specialized testing for migrations or integrations, we can help.

[Schedule a Testing Strategy Consultation →]

Or download our complete Salesforce Testing Guide (Parts 1 & 2) for comprehensive coverage of testing types, tools, best practices, and implementation approaches.

This guide is part of TestFort's Salesforce Testing series. For foundational concepts, testing types, and platform challenges, see Salesforce Testing Complete Guide Part 1. For automation strategies, tool selection, and best practices, see Salesforce Testing Complete Guide Part 2.