exalate

The Practical Guide to Reliable Integrations

Keep work moving across systems, teams and companies

A practical companion to the Exalate Field Report 2025–2026

SYSTEMS IN THIS GUIDE



An integration becomes part of how your business delivers work. When systems move, a supplier changes its process or the person who built the connection leaves, the important question is whether that work can continue.

This guide helps you decide what to check and what to do next. Start with your situation, follow the example and adapt the brief for your team.

CONTENTS

Start with the change you are facing

YOUR SITUATION	START HERE
Moving systems	1 **Move systems without losing live work**
Connecting companies	2 **Share progress while keeping control**
Serving multiple clients	3 **Run MSP integrations across multiple clients**
Connecting ITSM and delivery	4 **Connect Jira and ServiceNow around live work**
Supporting customers	5 **Close the loop between support and engineering**
Running an existing connection	6 **Make the connection work beyond one person**
Getting approval	7 **Build a business case that can be checked**
Using your business AI tool	8 **Use this guide with your business AI tool**
Mapping fields between tools	R **Reference: Field mapping by connection**

How each chapter works

Each chapter follows one team through a common situation. Adapt the rules, owners, and thresholds to your own workflow.

From the field → Example → Decisions to make → The agreed brief →
Checks before rollout → With Exalate → Discuss with your team → Customer story

The examples cover manufacturing, insurance, managed services, retail and enterprise software. Start with the one closest to yours. The same decisions apply in other industries.

1 Manufacturing | 2 Insurance | 3 Managed services | 4 Retail | 5 Enterprise software

CHAPTER 1 — MIGRATION

1 2 3 4 5 6 7 8

Move systems without losing live work

A migration plan has to cover more than the history. It also has to explain what happens to new work and live updates while records move. A successful bulk transfer doesn't prove your connected workflows are ready.

29%

of ~110 customer conversations

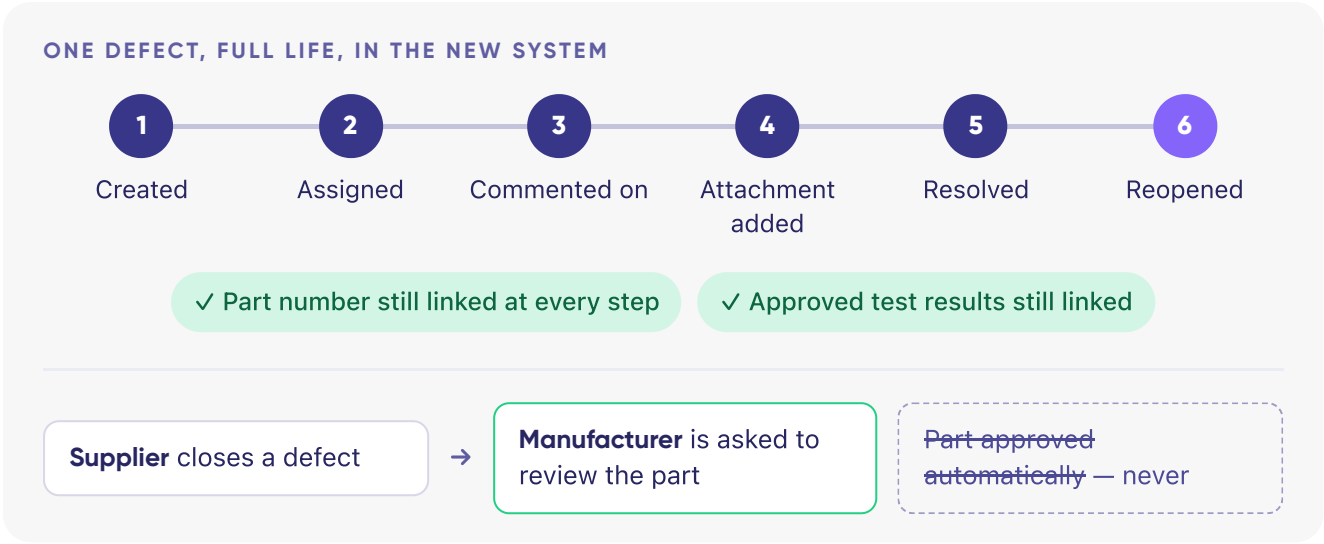
From the field: 29% of the conversations involved a migration or a period of running two systems side by side. Moving the history was rarely the hard part. Reconnecting records that were already synced, and rewriting scripts that depended on the old environment, took the most planning. Teams that reviewed their scripts early had the smoothest cutovers.

1.1 Example: manufacturing

A manufacturer is moving its engineering team's Jira to the cloud. One of its component suppliers tracks defects in its own tool, and the two sides sync those defects every day.

Each shared defect holds three things both teams depend on: the part number of the affected component, the supplier's replies, and the test results the manufacturer has approved. If any of these goes missing in the move, engineers can't tell which component failed or what the supplier already found.

So before switching over, the team takes one defect through its full life in the new system: created, assigned, commented on, given an attachment, resolved and reopened. At every step, they check that the part number and test results are still linked. They also make sure that when the supplier closes a defect, it only asks the manufacturer to review the part. It never approves the part automatically.



1.2 Decisions to make

Every migration with a live integration raises the same four questions:

IF THIS IS YOUR SITUATION	YOUR NEXT ACTION
Both systems will remain active	Agree which system owns each field during coexistence and how conflicting edits are handled.
Existing records are already paired	Test how those links will be preserved or re-established. Check for duplicates.
Rules depend on the old environment	Identify API, authentication and script dependencies. Test their replacements.
Volume or attachment size is uncertain	Rehearse a representative batch. Measure throughput, backlog and failure recovery.

1.3 The agreed brief

The manufacturer's migration lead and the supplier's integration owner wrote down what had to be true before go-live:

Scope: open supplier defects, their part numbers and approved test results. Closed history moves separately, with its own test.

Ready for go-live when: links and part numbers match in both systems, supplier replies arrive, test results open, private notes stay private, and any rejected update shows up for someone to fix.

Sign-off: the migration lead and the supplier's integration owner.

Stop if: updates go missing without explanation, records are duplicated, or data reaches someone it shouldn't.

Recovery: run the rollback or reconciliation steps you rehearsed before turning the flow back on.

1.4 Checks before rollout

Test the awkward cases too: a user who doesn't exist in the new system, a defect without a part number, and an attachment the new system won't accept. Then move a realistic batch and compare the results.

Rehearse stopping and recovering with both owners, and agree who handles new work during the cutover. Once the new setup is accepted, switch off old triggers, remove access nobody needs, and file the final mappings. Give each step an owner and a date.

With Exalate: Exalate keeps both systems live during the move and reconnects synced records after a bulk transfer with Atlassian's Jira Cloud Migration Assistant. Test Run checks a script change against real records before it reaches production.

DISCUSS WITH YOUR TEAM

Which part numbers, supplier updates, and test results must survive your migration?

CUSTOMER STORY · STELLANTIS

~150,000	25 days	~5,000	Zero
issues moved	from start to finish	issues synced a day	user downtime

Customer story: Stellantis inherited a Jira 7 Server instance too old for the Jira Cloud Migration Assistant. With partner ACA Group, it synced about 5,000 issues a day into a clean Jira Cloud, merging issue types and duplicate fields along the way. About 150,000 issues moved in 25 days, and go-live was a permission switch over one weekend with zero user downtime. Read the Stellantis story.

CHAPTER 2 — CROSS-COMPANY

1 2 3 4 5 6 7 8

Share progress while keeping control

When two companies share work, start by agreeing what each side sends, what it accepts, and who responds when something goes wrong.

39%

crossed a company boundary

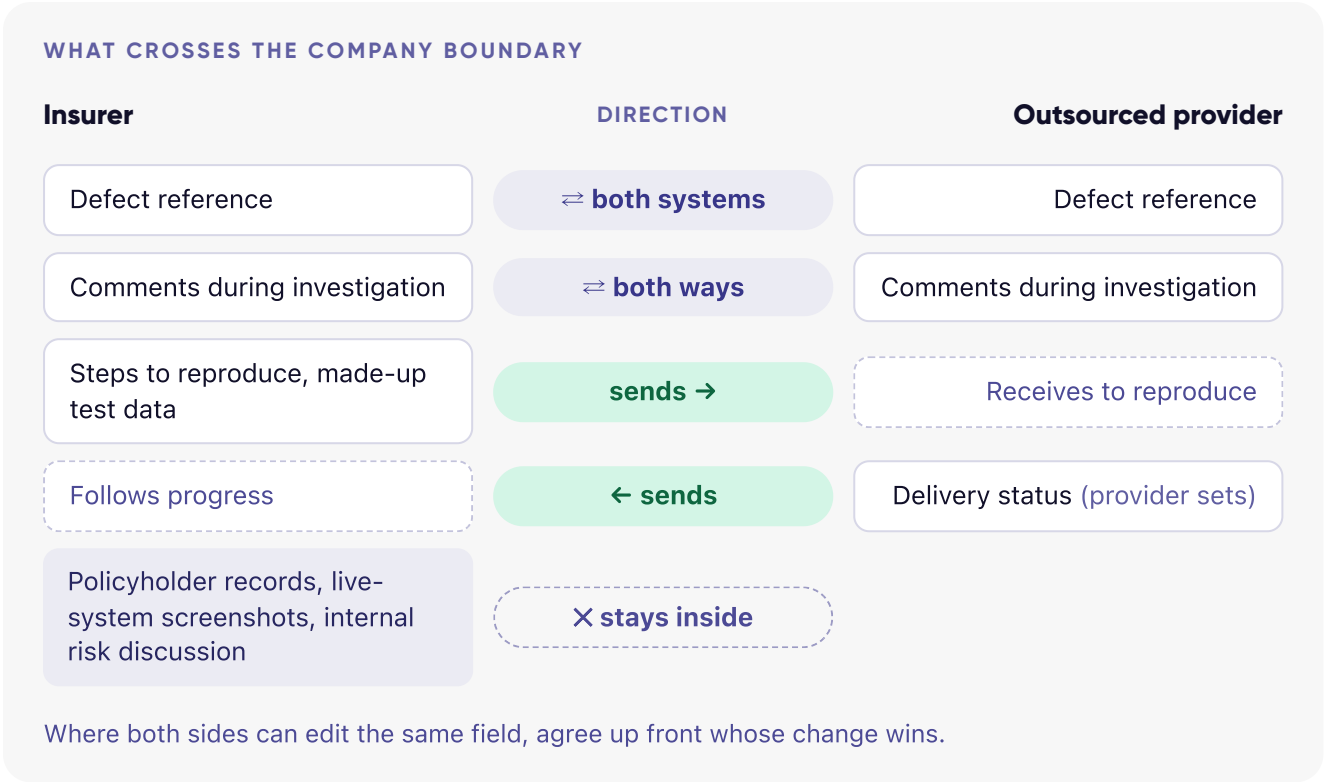
From the field: 39% of the conversations described an integration that crosses a company boundary, with vendors, clients, partners or outsourcers. The deciding condition was the same every time: each side wanted to control its own configuration, fields, and security. Integrations customers called business-critical almost always crossed a company boundary.

2.1 Example: insurance

An insurer outsources development of its policy administration software. When the insurer's testers find a defect, the outsourced team fixes it, and the insurer wants to follow progress without logging into the provider's tools.

The provider needs enough detail to reproduce the defect: a reference number, the steps that cause it, and test data. But the insurer can't share real policyholder records, screenshots from its live system, or its internal discussion about risk. So the provider gets made-up test data instead.

Some updates need to flow both ways, such as comments while the defect is investigated. Others flow one way, such as delivery status, which only the provider sets. Where both sides can edit the same field, they agree up front whose change wins.



2.2 Decisions to make

From this, the insurer and the provider agreed, field by field, what to share:

INFORMATION	WHAT THEY AGREED
Defect reference	Shared in both systems, so each side can find the same defect.
Description	Share the steps to reproduce it, using made-up test data.
Delivery status	The provider owns it, mapped to a status the insurer's testers understand.
Internal comments	Keep out policyholder records, live-system screenshots and internal risk discussion.
Attachments	Allow agreed file types and sizes. Flag anything held back so someone sees it.

2.3 The agreed brief

They wrote it down in a short brief both owners signed off on:

Purpose: let the insurer follow a defect the provider is fixing.

Ownership: the insurer decides urgency and accepts the fix. The provider owns delivery progress.

What's shared: the defect reference, an approved description, made-up test data, and progress updates. Policyholder records, live-system screenshots, and internal notes stay out.

Changes: both owners review any change to shared fields before it goes live.

If it fails: whoever spots the problem alerts the other owner through the agreed channel. The provider updates affected users.

Ending the connection: stop the triggers, remove access, and handle stored records as the contract says.

2.4 Checks before rollout

Test a normal progress update, the made-up test data, a restricted note, and an attachment of a type you don't allow. Check what arrives, what's held back, and who sees the rejection.

Then ask the provider's team to reproduce the defect using only what arrived. Review the boundary with both owners before changing shared rules.

With Exalate: Each company controls its own outgoing and incoming sync rules, so neither side needs access to the other's system.

DISCUSS WITH YOUR TEAM

Could the receiving team investigate the defect using only the information you have approved for sharing?

CUSTOMER STORY · VODAFONE GERMANY

20,000+	~10,000
synced entries	synced comments

Customer story: Vodafone Germany syncs a large Jira environment with external vendors for release management. Triggers send only vendor-specific issues, and each side controls its own sync. The connection has passed 20,000 synced entries and nearly 10,000 synced comments. Read the Vodafone story.

CHAPTER 3 — MULTI-CLIENT MSP

Run MSP integrations across multiple clients

A managed service provider (MSP) often does the same kind of work for several clients, each with its own tools, permissions, and service levels. The goal is to reuse one tested setup while keeping every client's records, access, and changes separate.

~90%

of fields covered by one standard mapping

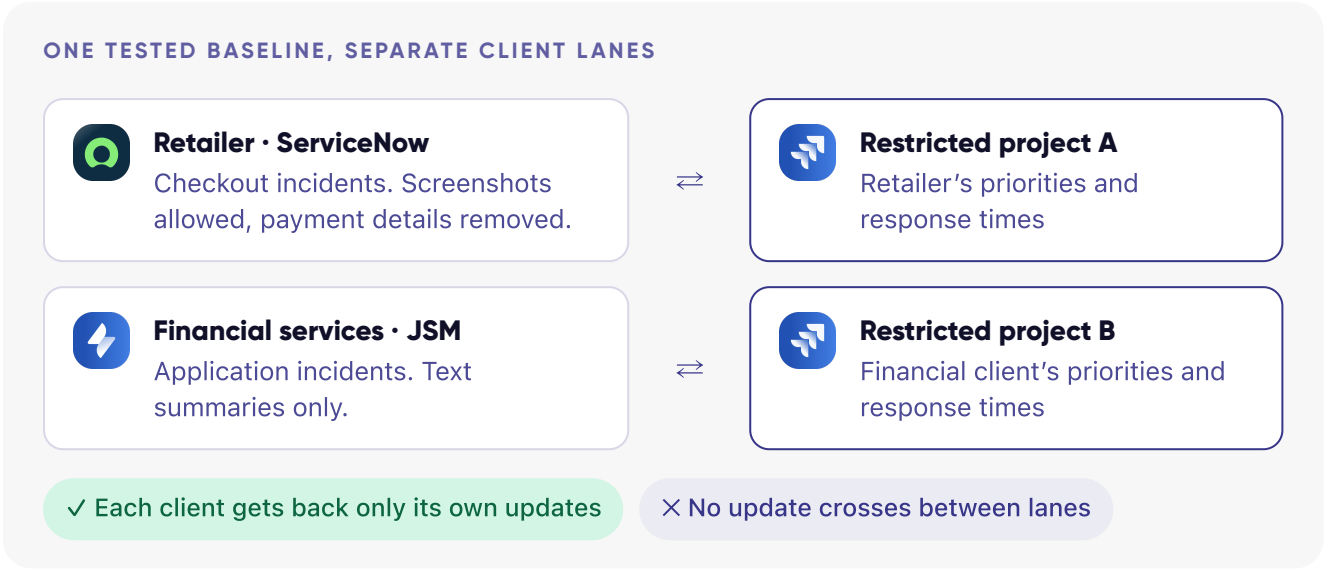
From the field: About half of the cross-company conversations came from service providers syncing their own ticketing with their clients. Software vendors with implementation teams described a standard mapping that covers about 90% of fields, plus 5 to 10 extra fields for each client. Each client's security review, not product fit, usually set the go-live date.

3.1 Example: managed services

An MSP runs its delivery team in Jira and serves two clients. A retailer sends checkout incidents from ServiceNow. A financial services company sends application incidents from Jira Service Management.

The two clients have different rules. The retailer allows screenshots of errors, as long as payment details are removed. The financial services client allows text summaries only. Each client also has its own priority levels and response times in its contract.

So the MSP keeps each client's incidents in a separate, restricted Jira project, and sends each client back only its own updates.



3.2 Decisions to make

Running connections for several clients comes down to five decisions:

DECISION	RECOMMENDED STARTING POINT
Client routing and identity	Use a verified client identifier and approved destination. Never route on a free-text client name alone.
Access and data boundaries	Scope access and shared fields to each client. Check permissions in the MSP tool as well as the connection.
Shared configuration	Version a common baseline; record and approve client-specific mappings and exclusions.
Service commitments	Keep contractual response and resolution clocks distinct from synced priority and status.
Failure and client exit	Define how to pause one client flow, reconcile updates and revoke its access without disrupting others.

3.3 The agreed brief

For the retailer, the MSP's brief looked like this:

Scope: checkout incidents sent from the retailer's ServiceNow to its restricted project in the MSP's Jira.

Ownership: the retailer's service desk owns business impact. The MSP's delivery lead owns progress.

What's shared: approved incident details, error screenshots with payment data removed, and progress updates. Payment data, internal notes, and other clients' records stay out.

Setup: the MSP's tested baseline, plus the retailer's routing, priority mapping and attachment rules. Both owners review the retailer-specific settings before switching on.

If it fails or the contract ends: pause the retailer's flow, tell both owners, catch up on missed updates, and remove access as the contract says.

3.4 Checks before rollout

Test similar ticket numbers from both clients, an incident with no client ID, and an update sent to the wrong project. Make sure updates can't cross between clients, and that any attempt is flagged. Check that a screenshot allowed for the retailer is blocked in the financial services client's text-only flow.

Then rehearse changing the shared baseline and pausing one client. Track failed updates and manual follow-up per client.

With Exalate: Providers who don't want to run every client connection themselves can use Exalate's managed service. A dedicated team scopes, builds, monitors, and maintains each connection; reusable templates speed up each new client, and costs are itemized per connection so you can pass them on as a line item. See how managed services work.

DISCUSS WITH YOUR TEAM

Could you change or pause one client's connection without exposing its records or interrupting another client?

CUSTOMER STORY · QUORUM CYBER

A couple → ~100
external syncs a month

+60%
more customers ask for a service desk integration

Customer story: Quorum Cyber's clients wanted security tickets in their own service desk instead of another vendor portal. Exalate's managed service connects them. External syncs grew from a couple to about 100 a month, and 60% more customers now ask for a service desk integration. Read the Quorum Cyber story.

3 Run MSP integrations across multiple clients

Next: 4 Connect Jira and ServiceNow around live work →

CHAPTER 4 — ITSM AND DELIVERY

1 2 3 4 5 6 7 8

Connect Jira and ServiceNow around live work

When the service desk works in ServiceNow and engineering works in Jira, agree what each record stands for before you map a single field. A ServiceNow incident and a Jira bug can have different owners, priorities, and rules for closing.

25%

Jira to ServiceNow, second only to Jira to Jira

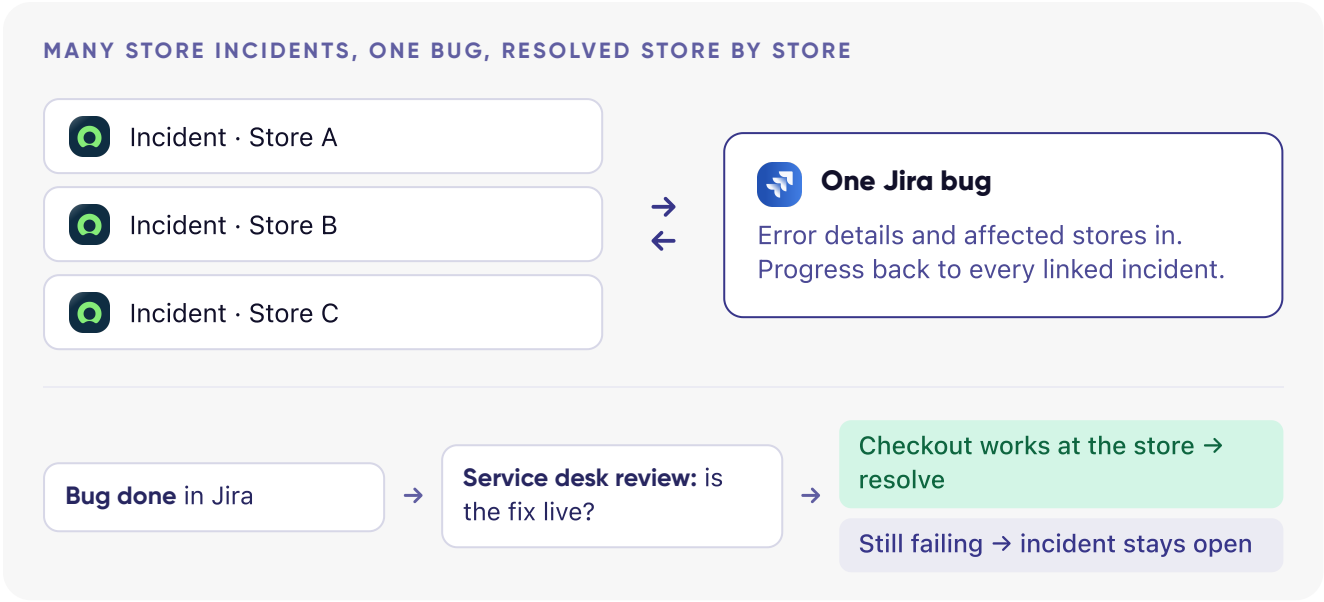
From the field: Jira to ServiceNow came up in 25% of the conversations, second only to Jira to Jira. The friction was rarely the connection itself. It was mapping users between Jira account IDs and ServiceNow user records, and handling fields that arrive empty.

4.1 Example: retail

A retailer's stores start reporting that checkout is failing. Each store's report becomes an incident in the service desk's ServiceNow.

The service desk sees that the incidents share one cause, so it links them all to a single bug in engineering's Jira, with the error details and the stores affected. Engineering's progress then flows back to every linked incident.

When engineering marks the bug as done, the service desk doesn't close the incidents straight away. It first checks that the fix is live and that checkout works again at each affected store. Only then does it resolve that store's incident.



4.2 Decisions to make

This example raises five decisions every Jira and ServiceNow connection needs:

DECISION	RECOMMENDED STARTING POINT
Records and escalation trigger	Specify the ServiceNow record type, Jira issue type and condition that creates or links the delivery record.
Record relationships	Choose one-to-one or many-to-one linking explicitly. Preserve references and prevent duplicate creation.
Priority and status	Map business impact to an agreed delivery signal. Keep incident urgency distinct from engineering priority.
Comments and attachments	Map only explicitly shareable content. Treat ServiceNow work notes and Jira comment restrictions separately.
Resolution and reopening	Return a resolution for service-team review. Agree how reopening affects the linked record.

4.3 The agreed brief

The service desk and engineering agreed on this brief:

Scope: checkout incidents in ServiceNow linked to one Jira bug for the shared cause. Unrelated checkout issues stay separate.

Ownership: the service desk owns incident priority, communication with stores, and checking that service is back. Engineering owns delivery progress.

What's shared: store references, error details, and progress updates. Payment data, internal work notes, and restricted comments stay out.

Resolution: a done bug in Jira asks the service desk to review. Each incident is resolved only after checkout works again at its stores.

Ready when: escalating the same problem twice doesn't create a second bug, every linked incident gets progress updates, and a failed update reaches the integration owner.

4.4 Checks before rollout

Test a status with no mapping, a missing required field in Jira, a restricted comment, and an attachment that's too large. Then test a fix that works at some stores but not others: only the stores where checkout works again should resolve.

Reopen one incident and check what happens in Jira. Make sure retries keep the links intact and that incident SLAs behave as agreed. Measure time spent chasing status before and after rollout.

With Exalate: Exalate's Groovy-based sync rules map statuses value by value, look up users between systems, set defaults for empty fields, and keep internal work notes private. See how to turn a ServiceNow case into a Jira epic.

DISCUSS WITH YOUR TEAM

Who confirms that service is restored, and what happens when the same fix works for only some affected locations?

CUSTOMER STORY · UBER FREIGHT

~130,000	15 months	Zero
tickets moved	of history	downtime

Customer story: Uber Freight moved about 130,000 tickets and 15 months of history from ServiceNow to Jira Service Management. A live two-way sync kept the service desk working throughout, with zero downtime. Read the Uber Freight story.

CHAPTER 5 SUPPORT AND ENGINEERING

1 2 3 4 5 6 7 8

Close the loop between support and engineering

A good escalation gives engineers enough context to act, and gives support enough visibility to keep the customer informed. It also keeps internal discussion separate from what customers see.

47%

support-to-engineering, the most common reason

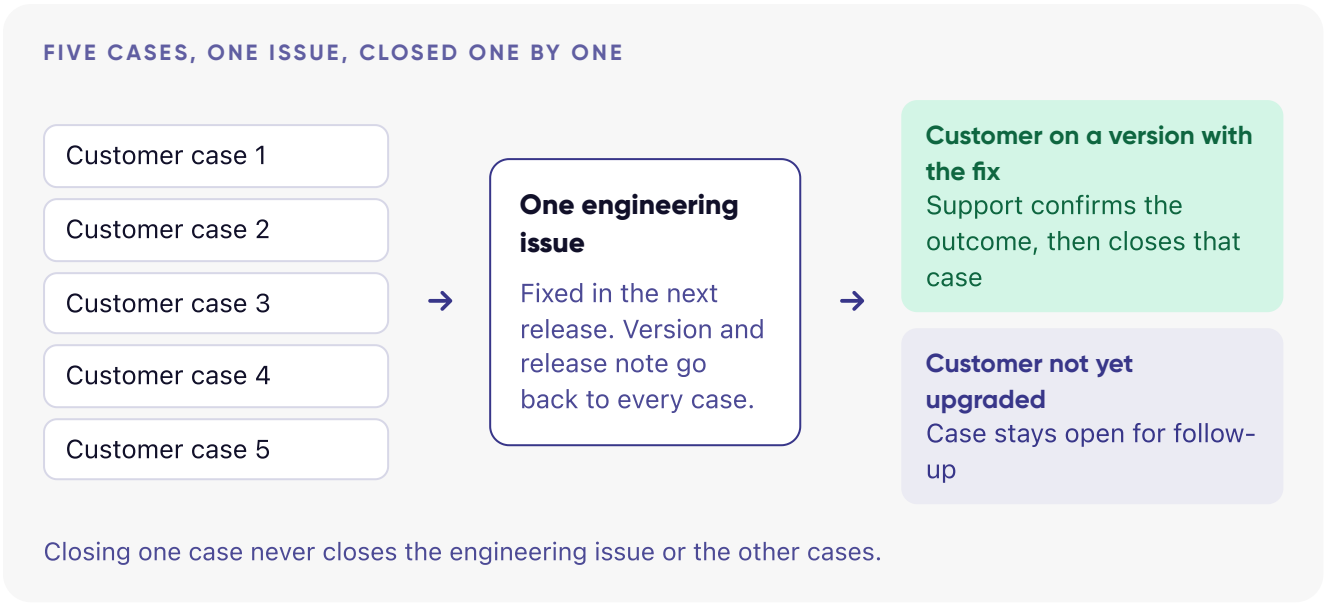
From the field: 47% of the conversations described a support-to-engineering flow, still the most common reason teams integrate. The workaround they wanted to escape was always the same: agents copying tickets by hand and updates relayed by email. One team's email bridge created a duplicate ticket for every update.

5.1 Example: enterprise software

Five business customers report the same bug to a software company's support team. Instead of opening five engineering tickets, support links all five cases to one engineering issue.

Engineering fixes the bug in the next release. But the five customers run different versions of the software and upgrade on different schedules, so the fix doesn't reach them all at once.

Support needs to know which version contains the fix and when it ships. Only then can it tell each customer when the fix applies to them, and close each case when it actually does.



5.2 Decisions to make

To make that work, the team settled five questions:

DECISION	RECOMMENDED STARTING POINT
When to create an engineering issue	Use an explicit escalation condition and check for an existing linked defect.
What engineering receives	Send the problem, reproducible steps, impact, and approved attachments.
What support receives	Return an agreed progress status and explicitly shareable updates.
What happens at resolution	Notify linked tickets for agent review; do not assume every customer case is resolved.
What stays private	Exclude internal notes unless the workflow explicitly permits sharing.

5.3 The agreed brief

Support and engineering agreed on this brief:

Scope: five customer cases linked to one engineering issue, with the affected versions and approved release details.

Ownership: engineering owns delivery progress. Support owns customer communication and closing cases.

Ready when: every linked case shows the delivery status, the version that contains the fix and an approved release note. Internal engineering discussion stays private.

Closing cases: a release prompts support to review each case. Support confirms the fix applies and checks the customer's outcome before closing it. Closing one case doesn't close the engineering issue or the other cases.

If it fails: a failed update shows up for the named integration owner.

5.4 Checks before rollout

Test a missing required field, an unknown user, a status with no mapping, and a restricted comment. Then test a released fix against cases on different versions. A customer on a version without the fix should stay open for follow-up, and closing one customer's case shouldn't close the others.

Track time spent chasing status, duplicate engineering issues, and escalations missing context, before and after the change. Faster updates only matter if the right information reaches the right people.

With Exalate: Exalate creates the engineering issue when a ticket meets your escalation condition, links related cases to it, and sends progress back to the agent's tool. Screenshots and formatting in the customer's ticket arrive intact, so engineers see the problem the way the customer reported it. Comments can sync one way only, so internal notes never reach customers. See how to link many Zendesk tickets to one Jira work item.

DISCUSS WITH YOUR TEAM

What would you need to know before telling each affected customer that a released fix resolves their case?

CUSTOMER STORY · DPG MEDIA

2,000 → 6,000–8,000
employees through acquisitions

Manual → synced
Zendesk to Jira, freeing a role for product work

Customer story: DPG Media grew from 2,000 to between 6,000 and 8,000 employees through acquisitions. An IT employee was copying Zendesk tickets into Jira by hand, and the free connector couldn't handle more than one instance. With Exalate syncing the tickets, that employee moved into a product owner role. Read the DPG Media story.

5 Close the loop between support and engineering

Next: 6 Make the connection work beyond one person →

CHAPTER 6 — OWNERSHIP AND HANDOVER

1 2 3 4 5 6 7 8

Make the connection work beyond one person

Integrations tend to depend on whoever set them up. Plan for that person to be away, change roles, or leave, so the work keeps moving without them.

32%

of check-in conversations

From the field: 32% of check-in conversations involved a team that had lost the person who set up the integration, or relied on one person to run it.

6.1 Example: service delivery

A service provider's integration admin is moving to a new role next month. She set up the connections that sync client incidents with the provider's delivery team, and she's the only one who knows them well.

Before she moves, a service operations engineer becomes the backup owner. He gets his own access, reads the documented mapping, and learns how long updates normally take to arrive.

To prove the handover works, he makes a real change while she's still around: a client wants one extra field synced. He adds it, tests it, and publishes it on his own. When she moves on, nothing changes for the clients.

HANDOVER, PROVEN BEFORE THE PRIMARY OWNER MOVES ON

1. Backup gets their own access
2. Reads the documented mapping
3. Learns normal update times
4. Adds, tests and publishes one real change
5. ✓ Primary moves on. Nothing changes for clients.

6.2 Decisions to make

Ask these four questions about every connection:

CHECK	IF THE ANSWER IS NO
Can a backup owner access the tools?	Give them their own access now, before it's needed.
Can someone else tell what normal looks like?	Write down how long updates usually take, and agree when a delay needs attention.
Is the current configuration documented?	Record the mapping, the last version that worked, and why each rule exists.
Has the backup made a real change?	Have them make, test, and publish one change while the primary owner is still available.

6.3 The agreed brief

The provider's handover brief looked like this:

Workflow: client incidents escalated to the provider's delivery team.

Owners: the integration admin as primary, a service operations engineer as backup with their own access, and the client's service desk admin on the other side.

Configuration: the approved mapping and version history live in the connection documentation. Credentials stay in the company's password vault.

Normal behavior: how long updates usually take, and when a delay should alert the operations queue.

Making changes: test every change before publishing it, and record who changed what and why.

Communication: the service delivery lead tells clients about any planned change that affects them.

6.4 Checks before rollout

Ask the backup owner to explain the connection back to you: what it syncs, what normal looks like, and where the documentation lives. Then have them make one small change, end to end.

Record any missing access, instructions, or information as follow-up tasks with an owner and a date.

With Exalate: Exalate keeps every connection in one console, with version control that records each script change, who made it, and when. Aida, Exalate's AI assistant, helps a new admin write and edit sync rules from plain-language prompts.

DISCUSS WITH YOUR TEAM

Could your backup owner make a change to the integration tomorrow, without help?

CHAPTER 7 — BUSINESS CASE

1 2 3 4 5 6 7 8

Build a business case that can be checked

Start with one workflow you can measure and a realistic assumption about how much it will improve. Count freed-up staff time as capacity, unless the change produces a cash saving you can point to.

24% vs native connector

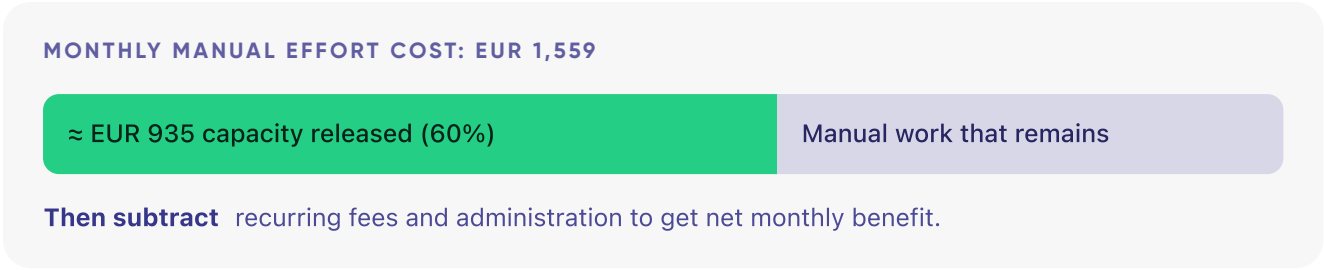
20% vs in-house build

From the field: In 24% of the conversations, teams chose Exalate over a native connector, and in 20% over an in-house build. In-house builds tended to work until attachments, API changes, or maintenance hit. The person running the integration was rarely the person who approved the budget.

7.1 Example: an escalation workflow

A support team spends about eight hours a week copying escalations between tools. Here's how its business case adds up:

INPUT OR CALCULATION	VALUE
Manual work each week	8 hours
Loaded hourly cost	EUR 45
Monthly manual effort cost	8 × 4.33 × EUR 45 = **EUR 1,559**
Expected effort reduction	60 percent
Monthly capacity released	**Approximately EUR 935**
Net monthly benefit	Capacity value less recurring fees and administration
Simple payback	Setup and transition cost ÷ positive net monthly benefit



Use your own figures. Include the manual work that remains and the effort to support the integration, and don't count the same time saving twice. Payback based on freed-up time is an economic estimate, not necessarily cash in the bank.

7.2 Decisions to make

Each approver needs different evidence:

APPROVER	EVIDENCE TO AGREE
Operations	Workflow, baseline manual effort, owners and recovery plan.
Security	Approved data flow, access model and hosting requirements.
Budget owner	Setup and transition costs, recurring fees, administration and improvement assumptions.
Client or partner	Responsibilities, service commitments and who pays.

7.3 The agreed brief

The team's brief for approval:

Scope: one measured escalation workflow.

Baseline: eight hours of manual work per week, checked with the team.

Assumption: 60 percent less manual effort, confirmed during the pilot.

Costs: actual setup, transition, license, and administration estimates.

Decision: the budget owner reviews net benefit and risk before approving a wider rollout.

7.4 Checks before rollout

Check a sample of the work with the people who do it. Measure the same way after the pilot, and revise the assumption if the savings are smaller than expected. If costs exceed the benefit, narrow the scope.

With Exalate: Exalate is priced per integration on the number of active items in sync, with no per-user fees. Base your cost estimate on the volume you'll sync, not your total user count. To compare against building in-house, use the build or buy calculator.

DISCUSS WITH YOUR TEAM

Which measured improvement would justify the cost, and who will check it after the pilot?

CUSTOMER STORY · QUALCO

100+
entities synced a day on one connection

Days → minutes
to produce reports

Customer story: Qualco runs support in Jira Service Management and development in Azure DevOps, syncing over 100 entities a day on one connection. Reports that took days now take minutes. Read the Qualco story.

CHAPTER 8 — WORKING WITH AI

1 2 3 4 5 6 7 8

Use this guide with your business AI tool

Use this guide as reference material in a company-approved AI tool that can read documents or pasted text. Combine the relevant chapter with approved context about your industry, systems, and workflow. Ask for a draft connection brief, acceptance tests, or workshop questions, then review the output with the people responsible.

01 **Prepare the material**

02 **Run the starting prompt**

03 **Review with owners**

04 **Turn it into a team decision**

8.1 Prepare the material

Upload the guide if your approved tool supports it, or paste one chapter and its copyable brief. Ask the tool to identify the chapter and summarize its main decisions first. If it misses the table or cannot read the attachment, paste that section as text.

Use only information permitted under your company's AI policy and client agreements. A company-approved tool is not automatically approved for every client dataset. Replace ticket content with synthetic examples where possible; keep credentials, tokens, and unnecessary personal or confidential information out of the prompt.

8.2 Copy this starting prompt

Replace the Our situation paragraph with your own approved context. Include your industry, systems, workflow, known constraints, and decisions still open. Copy the following five paragraphs together.

STARTING PROMPT **5 PARAGRAPHS · COPY TOGETHER**

Use the attached Practical Guide to Reliable Integrations as planning guidance. Its examples describe other teams, not our business. First identify the relevant chapter and summarize its main decisions so I can check that you have read it.

REPLACE WITH YOUR OWN APPROVED CONTEXT

Our situation: an MSP uses Jira to handle checkout incidents escalated by a retail client in ServiceNow. We need approved progress updates in both tools. We have not yet agreed on field ownership, sharing boundaries, or closure rules.

Use only the context I supply. Separate confirmed facts, proposed rules, and unknowns. Adapt the questions and tests to our industry and workflow. Ask up to five questions that materially affect the plan, including any missing operational context. Do not invent regulatory obligations, service commitments, or approved business rules.

Then draft a one-page connection brief covering scope, record relationships, routing, field ownership, data boundaries, failure handling, and client exit. Include acceptance tests with expected results, plus actions with a proposed owner role and required evidence. Mark all unconfirmed owners and dates as to be agreed.

Refer each recommendation to the relevant chapter heading. Do not assume that any vendor supports a requirement. List the capabilities we must verify in current documentation or a pilot. Produce a draft for human review; do not change any system.

8.3 Review before using the output

Ask each owner to check their part of the draft:

REVIEWER	WHAT THEY CHECK
Workflow owner	Business rules
Integration owner	Technical feasibility
Data owner	Sharing boundaries

Verify product claims against current documentation and test critical behavior. An AI-generated plan does not prove that the connection works or that a data flow is authorized.

8.4 Turn the AI draft into a team decision

Keep the discussion in your existing project or service-management tool. The AI output becomes useful when someone resolves an open question, tests a proposed rule, or accepts responsibility for a next step.

8.5 Worked example

For the MSP scenario, the AI may propose copying Jira Done to ServiceNow Resolved. The service desk could instead require restoration checks before resolving the incident. Record the agreed rule and ask the AI to update the draft and its tests. This keeps a plausible suggestion from becoming an unexamined business decision.

AI MAY PROPOSE

Done → ~~Resolved~~

TEAM AGREES

Done → Service-desk review → Restoration check → Resolved

8.6 Copy this follow-up prompt

FOLLOW-UP PROMPT

Revise the brief with this confirmed rule: Jira Done requests a service-desk review and does not resolve the ServiceNow incident. The service desk must verify that the fix is deployed and checkout works at the affected stores. Add tests for successful validation, partial restoration, failed validation, and reopening. For each test, state the starting records, action, and expected result in both systems. Keep all other unknowns visible. End with the decisions the team still needs to make.

REFERENCE

Field mapping by connection

These are the fields teams spent the most time on for each connection, and the issue that tripped them up most often.

CONNECTION	MAP THESE CAREFULLY	WATCH OUT FOR
Jira and ServiceNow	Assignee and reporter, assignment groups, state and status, comments marked internal or external	Jira uses account IDs, and ServiceNow uses its own user records, so build a user lookup before go-live.
Jira and Azure DevOps	Issue types to work item types, status to state, parent and child links	Hierarchies flatten if parent links aren't mapped explicitly.
Jira and Zendesk	Status, priority, custom list fields, links between several tickets and one work item	Zendesk has fewer statuses than most Jira workflows, so decide which Jira statuses share one Zendesk status.
Jira and Salesforce	Cases with their contact and account details; status back to Salesforce	Decide which side owns each field. One team pushes about 40 custom fields one way only.
Jira and GitHub	Issues and comments from GitHub into Jira	Jira labels can't contain spaces, so GitHub labels with spaces need converting. Keep fix versions and internal status away from external reporters.

For every connection, set a default for fields that can arrive empty. Empty values were one of the most common reasons a script stopped.

Choose one useful next step this week

IF THIS IS YOUR SITUATION	YOUR NEXT STEP
You are migrating	Rehearse one complete connected workflow. Chapter 1
You are sharing work externally	Agree the field boundary with the other owner. Chapter 2
The connection already runs	Ask a backup administrator to complete a recovery exercise. Chapter 6

Record the outcome where the team already works.

ABOUT THIS GUIDE

Based on themes in the Exalate Field Report 2025–2026, covering approximately 110 customer conversations from July 2025 to September 2026.

exalate

A question about your own workflow

If a question remains, you are welcome to contact Exalate. Share the systems involved, the workflow you need to support, and the decision you are working through. That gives us a useful place to start.

Discuss your use case with an integration expert → or try Exalate for free → .

info@exalate.com exalate.com/contact