Outcome Owl

See Clearly. Act Decisively.

Process Discovery Worksheet

A guided worksheet for mapping your business process into Outcome Owl.

Introduction

Use this worksheet to think through one business process or workflow you want to observe. You do not need all the answers right now — start with what you know, and Outcome Owl will help you discover what you did not expect.

Fill out one worksheet per process. If you have multiple processes, start with the one that causes the most pain or costs the most when it goes wrong.

Before You Start — Why This Process?

Before mapping steps and systems, take a few minutes to articulate *why* this process matters to your organization and *where it hurts today*. The answers here will guide every decision that follows — which steps to observe, what rules to set, how severe to rate them, and who needs to know when something goes wrong.

What Prompted This?

Something brought you here. Understanding the trigger helps focus the setup on what matters most.

Question	Your Answer
Why are you looking at this process now? (e.g., audit finding, customer complaints, cost overruns, compliance requirement, executive mandate, gut feeling that something is off)	

Question	Your Answer
Is there a specific incident or pattern that raised concerns? (e.g., “We missed three SLA deadlines last quarter,” “A claim was lost for 60 days,” “We have no visibility into how long onboarding actually takes”)	
What would success look like? (e.g., “We catch delays within 24 hours instead of discovering them a week later,” “We can prove to auditors that every claim is processed within the regulatory window”)	

Where Does It Break Today?

Every process has pain points — steps where things get stuck, exceptions that require manual intervention, or handoffs where work disappears. These are the areas Outcome Owl is designed to illuminate.

Question	Your Answer
Which steps have caused problems in the past? (delays, errors, rework, lost items)	

Question	Your Answer
Are there manual workarounds your team uses today? (spreadsheet tracking, email follow-ups, daily status meetings to find stuck items)	
Where do handoffs fail? (work moves from one team or system to another, and things fall through the cracks)	
Are there steps that sometimes do not happen at all? (a required approval is skipped, or a notification is never sent)	

How Critical Is This Process?

Not all processes carry the same weight. Understanding the business impact helps you assign meaningful severity levels to your deviation and threshold rules, and set the right Impact Areas on each process step.

Think about which of these areas are affected when this process goes wrong. Check all that apply — most processes touch more than one.

Impact Area	Affected?	How?
Compliance — regulatory requirements, audit obligations, legal exposure		

Impact Area	Affected?	How?
Customer Experience — service quality, responsiveness, satisfaction		
Finance — revenue, cost, margin, cash flow		
Operations — throughput, efficiency, resource utilization		
Reputation — brand perception, public trust, partner confidence		
Revenue — direct revenue impact, lost sales, delayed billing		
Security — data protection, access control, fraud exposure		

These seven areas correspond directly to the Impact Area flags you will set on each process step in Outcome Owl. The answers above will make that configuration step straightforward.

Who Needs to Know?

When this process goes wrong, who should hear about it — and how quickly?

Question	Your Answer
Who is the day-to-day owner? (the person or team who fixes problems when they arise)	

Question	Your Answer
Who needs to know if a problem persists or escalates? (manager, VP, compliance officer)	
How quickly does someone need to know? (real-time, same day, weekly summary)	
Are there external stakeholders who are affected? (customers, regulators, partners)	

These answers will inform how you configure severity levels (1–5) on your deviation and threshold rules, and will be valuable when your organization is ready to set up notifications and escalation paths.

1. Name Your Process

What do your people call this process today? Use the vocabulary your organization already uses — not a system name or a technical label.

Question	Your Answer
Process name (e.g., “Medical Claim,” “Order Fulfillment,” “Employee Onboarding,” “Equipment Lifecycle”)	
Who owns this process? (team, department, or role)	
What is at stake when this process goes wrong? (customer impact, financial exposure, compliance risk, operational cost)	
How many of these processes run per day / week / month? (rough order of magnitude is fine)	

2. The Starting Step

Every process begins somewhere. This is the event that kicks things off.

Question	Your Answer
What is the starting step called? (e.g., “Claim Received,” “Order Placed,” “Application Submitted,” “Asset Received into Inventory”)	
Which system produces this event? (e.g., ERP, CRM, claims platform, manual entry)	
Can that system export a file or send a message when this event happens? (CSV extract, API call, Kafka message, manual log)	
What unique identifier does this step carry? (e.g., Claim Number, Order ID, Application ID, Asset Tag). This is the thread that ties the whole process together.	
Does this step have a meaningful duration, or is it a point in time? A “Claim Received” might be instantaneous. An “Initial Review” takes hours or days.	

Attributes worth capturing at this step:

List any data fields that travel with this event and might be useful for analysis later. Think about: who, what, where, how much, which type.

Attribute	Example Value	Why It Might Matter

Examples: customer_type = "Enterprise", region = "Southeast", submitted_amount = 4500.00, channel = "Web Portal", priority = "Expedited"

Common pitfalls with identifiers:

- **Do not use the entity ID alone when the entity passes through multiple steps.** A Claim Number identifies the claim, but you will have multiple observations for that claim (Received, Reviewed, Paid). Include the step in the ID to avoid collisions. See the Administration Guide for a worked example.
- **Do not use a timestamp as an identifier.** Timestamps collide when two events happen in the same second, carry no semantic value, and break when system clocks drift.
- **Do not use a display string that users can edit.** If a customer-facing label changes ("Case #4421" becomes "Escalated Case #4421"), the audit trail breaks — Outcome Owl matches the original ID, not the new one.

3. The Ending Step(s)

Most processes have at least one “good” ending and often a “bad” ending too.

Positive Ending (the happy path)

Question	Your Answer
What does the successful ending look like? (e.g., “Claim Paid,” “Order Delivered,” “Loan Funded,” “Asset Retired”)	
Which system knows when this happens?	
Does it carry the same unique identifier as the starting step, or a different one that can be linked back?	
Does this step have a meaningful duration?	
What does “good” look like in terms of time? (e.g., “The whole process should complete within 7 days,” “This final step should take less than 2 hours”)	

Negative Ending (the unhappy path)

Not every process ends well. Some are denied, cancelled, abandoned, or escalated. These endings are just as important to observe.

Question	Your Answer
What does the negative ending look like? (e.g., "Claim Denied," "Order Cancelled," "Application Withdrawn," "Asset Lost or Damaged")	
Which system knows when this happens?	
Is there more than one kind of negative ending? List them:	

Attributes worth capturing at ending steps:

Attribute	Example Value	Why It Might Matter

Examples: paid_amount = 3200.00, denial_reason = "Incomplete Documentation", delivery_carrier = "Regional Express", resolution_type = "Refund"

4. The Steps In Between

Most processes have intermediate steps — things that happen between the start and the end. You do not need to capture every micro-step. Focus on the ones where:

- **Handoffs happen** (work moves from one team or system to another)
- **Decisions are made** (approve/deny, route, escalate)
- **Time passes and you want to know how much**
- **Problems tend to occur**

List the steps you can think of. It is fine if this list is incomplete — you can add more later.

Step Name	Which System?	Can You Collect It?	Duration or Point-in-Time?	Links Back To (which prior step?)

Examples:

Step Name	Which System?	Can You Collect It?	Duration or Point-in-Time?	Links Back To
Eligibility Check	Claims Platform	Yes, API event	Duration (30 min avg)	Claim Received
Underwriting Review	Loan Origination System	Yes, nightly CSV	Duration (1–3 days)	Credit Check
Quality Inspection	Warehouse System	Yes, scan event	Point-in-time	Order Packed
Asset Installed	Field Service App	Yes, technician log	Point-in-time	Asset Shipped

Steps that happen sometimes but not always:

Some steps only occur under certain conditions (e.g., “Manager Approval” only for orders above $5,000, or “Appeals Process” only for denied claims). These are valuable to track — they reveal the exception paths.

Conditional Step	When Does It Happen?

Attributes worth capturing at intermediate steps:

Different steps may carry different attributes. A shipping step has carrier and tracking information; a review step has reviewer and decision.

Step Name	Attribute	Example Value

5. Linkage — How Steps Connect

Outcome Owl links steps into an end-to-end process chain using identifiers your systems already produce. The simplest approach: every step carries the same unique identifier (the "golden thread").

Question	Your Answer
Do all your systems use the same identifier for this process? (e.g., every system references the same Claim Number or Order ID)	
If not, how do they connect? (e.g., "The warehouse system uses a Pick Ticket ID that maps to the Order ID in the ERP")	
Is there a natural parent-child relationship? (e.g., "Each order line item is a child of the order header")	

Do not worry if the linkage is imperfect. Outcome Owl handles out-of-order arrival, delayed events, and even cases where a child event arrives before its parent. The platform links them retroactively.

Which Identifier Pattern Fits Your Process?

Your answer to the first question above — whether all systems share the same identifier — is one of the most consequential decisions you will make. It determines how your observation IDs are designed, how strictly you can guard parent relationships, and how effectively Outcome Owl's analytics tools work for your organization over time.

Three common patterns:

Pattern A: Natural Golden Thread

All systems reference the same business identifier (e.g., every step carries the same Claim Number, Order ID, or Loan Application Number). The golden thread ties the entire process together natively.

- **Observation ID design:** Include the golden thread in every observation ID (e.g., `CLAIM-91823-RECEIVED`, `CLAIM-91823-REVIEWED`, `CLAIM-91823-PAID`). The business key is visible in every ID, making search and troubleshooting intuitive.
- **Linkage:** Strong. Every observation in the process chain shares a recognizable business key. Parent-child relationships are unambiguous.
- **Guardrails:** High value. Because linkage is clean, parent relationship guardrails catch genuine data errors rather than generating noise from ambiguous linkages.
- **Long-term benefit:** Search finds every step of a process from the business key alone. Process Explorer can navigate the full history of a business transaction by ID. Entity-level analytics — grouping all processes for a customer, asset, or member — can use the golden thread directly without additional configuration.

Pattern B: System-Scoped IDs with a Linking Key

Each system mints its own identifier, but a shared reference exists somewhere — maybe as a field in the record, maybe through a lookup table. For example, the warehouse uses Pick Ticket IDs, but each ticket references the ERP Order Number.

- **Observation ID design:** Prefix with the source system (e.g., `ERP-44210`, `WMS-PKT-8891`). The linking key (Order Number) travels as an attribute, not as part of the ID.
- **Linkage:** Requires mapping. The `parentid` on a WMS observation must reference the ERP observation's ID — someone (or some integration logic) needs to resolve the mapping.
- **Guardrails:** Useful — with caveats. Watch for false positives during the tuning period. If the mapping is occasionally wrong, guardrails will flag it — which is exactly what you want. But if the mapping is frequently approximate, you will generate noise.
- **Long-term benefit:** Attributes become the key to cross-process analysis. Attribute Insights and Attribute Drift reveal patterns across the linking key. Process Explorer navigation works through the parent chain; searching by the linking key requires it to be captured as a consistent attribute on every step.

Pattern C: No Shared Key

Systems are decoupled — there is no natural business identifier that appears everywhere. A customer complaint in the CRM, a return in the warehouse system, and an accounting adjustment in the ERP are related, but no single key ties them together until someone manually associates them.

- **Observation ID design:** Fully system-scoped (e.g., `CRM-CASE-4421`, `WMS-RET-8891`, `ERP-ADJ-11029`). Each system contributes its own namespace.
- **Linkage:** Relies on explicit `parentid` assignment at integration time. The team building the data pipeline decides which observation is the parent of which. This is the most error-prone pattern.
- **Guardrails:** Start permissive. Because linkage depends on integration logic rather than natural business keys, overly strict guardrails will generate false errors while you tune the pipeline. Add guardrails incrementally as the linkage stabilizes.
- **Long-term benefit:** Outcome Owl's own process chain becomes the linking mechanism your organization did not have before. Over time, Flow Explorer reveals the actual paths these disconnected systems create when linked together.

Question	Your Answer
Which pattern best describes your process? (A, B, or C)	
If Pattern B — what is the linking key, and which system is the authority?	
If Pattern C — who or what will assign the `parentid` when integrating?	

Why This Decision Matters Long-Term

The identifier strategy you choose today propagates through every analytics capability Outcome Owl provides — both current features and those on the roadmap:

- **Search** uses observation ID for exact and prefix matching. Pattern A customers can search by business key and instantly find every step. Pattern B and C customers rely on attributes or must know the system-scoped ID.
- **Process Explorer** navigates process trees by following parent-child chains. Clean linkage produces rich, navigable trees. Ambiguous linkage produces fragmented trees that require manual investigation.
- **Audit Trail Explorer** reconstructs the lifecycle of a long-lived entity (an asset, an employee, a contract) by grouping observations over months or years. A consistent identifier strategy — especially a golden thread — makes entity resolution straightforward.
- **Entity-level analytics** (grouping all processes for a customer, asset, or member across independent trees) depends on a shared identifier or consistent attribute to connect them. Pattern A customers get this for free. Pattern B customers get it through attribute-based grouping. Pattern C customers need explicit configuration.
- **Deep-linking from any page into the relevant Explorer** means a user can click an observation anywhere in the UI and see its full process context. This works best when process trees are cleanly formed — which starts with clean identifiers and guardrails.

The investment in getting identifiers and guardrails right during initial setup pays compounding returns across every analytics feature built on top of the process chain.

6. What Does Good Look Like?

Before setting up rules, think about what “normal” means for this process. These become your first deviation and threshold rules.

Time-Based Expectations (Deviation Rules)

“If step A happens, step B should follow within X days.”

After This Step…	…This Step Should Follow	Within How Long?	How Serious If It Doesn’t? (1–5)

Examples: - After “Claim Received,” “Eligibility Check” should follow within 2 days (severity 3) - After “Order Shipped,” “Order Delivered” should follow within 14 days (severity 3) - After “Application Submitted,” “Identity Verification” should follow within 1 day (severity 4)

Numeric Boundaries (Threshold Rules)

“Flag when a value exceeds or falls below a boundary.”

On This Step…	Check This Value…	Flag When…	How Serious? (1–5)

Examples: - On “Payment Processing,” flag when amount > $10,000 (severity 4) - On “Refund Issued,” flag when total refunds > $50,000 across all (severity 5) - On “Flight Completed,” flag when delay_minutes > 120 (severity 3)

Things That Should Not Happen

Some deviation rules detect unwanted events: “If step A happens, step B should NOT follow.”

After This Step…	…This Step Should NOT Happen	Within How Long?	How Serious? (1–5)

Examples: - After “Booking Confirmed,” “Flight Cancelled” should not follow within 30 days (severity 4) - After “Invoice Approved,” “Invoice Rejected” should not follow within 7 days (severity 4)

How your identifier pattern (Section 5) affects rules:

- **Pattern A (golden thread):** Rules are highly reliable because parent-child linkage is clean. Start with time-based expectations and trust the results immediately.
- **Pattern B (linking key):** Rules work well, but verify that parent mappings are correct before interpreting deviation results. A “missing step” deviation might mean the step did not happen, or it might mean the mapping broke.
- **Pattern C (no shared key):** Start with broad rules and loose thresholds. Tighten them as your integration matures and linkage errors decrease. Rules against strict sequential expectations will generate noise until the pipeline is stable.

7. Process or Audit Trail?

Most workflows are **Processes** — steps happen in a defined sequence with measurable duration between them.

Some are **Audit Trails** — standalone events tied to a long-lived entity (an asset, an employee, a contract) where the sequence is fluid, events span months or years, and duration between events is not the point.

Question	Your Answer
Is this a sequential workflow with a clear start and end? → Process	
Or is this a series of lifecycle events on a long-lived entity? → Audit Trail	
Could it be both? (e.g., Employee Onboarding is a Process; the employee's ongoing communications are an Audit Trail)	

8. Data Sources Summary

Summarize where the data will come from. This helps plan the technical integration.

Step(s)	Source System	Export Method	Frequency	Format
		CSV / API / Kafka / Manual		

9. Quick Sketch

Draw a simple left-to-right flow of your process. Boxes for steps, arrows for connections. Mark the starting step, the ending step(s), and any branches. This does not need to be perfect — it is a conversation starter.

(sketch your flow here)

What Happens Next

Once you have filled in what you can:

1. **You do not need every answer.** Start with the starting step, one or two intermediate steps, and an ending step. You can add more later without rebuilding anything.
2. **Lock in your identifier strategy early.** Your identifier pattern (Section 5) and guardrail decisions are the hardest to change later. Everything else — steps, rules, attributes — can evolve incrementally. The Administration Guide has detailed ID design guidance and worked examples.
3. **Configuration takes minutes.** Defining process steps in Outcome Owl is a configuration exercise — name, category, type. No code, no schema changes, no development project.
4. **Send us your first data file.** A CSV with just five columns (id, name, parentid, start, end) is enough to see your process in the platform.
5. **The platform discovers the rest.** Flow Explorer will map the actual paths your process follows. Attribute Insights will find correlations you were not looking for. Attribute Drift will alert you when your data changes shape.
6. **Rules evolve with your business.** Start with two or three deviation rules. Add more as you learn what matters. The What If? simulator lets you test rule changes against historical data before committing.

Outcome Owl is a product of Seattle Software Works, Inc.