

# Modeling Application Security Processes with *Work the System*, *Traction*, and *This Is Service Design Doing*

Jordan Suber

November 18, 2025

## At a Glance

- Section 1 – Why this document exists and what you can do with it.
- Section 2 – How each book contributes to modeling AppSec processes.
- Section 3 – A practical reading order tailored to AppSec work.
- Section 4 – A sketch of your current AppSec pipeline as blueprints and SOPs.
- Section 5 – How to combine these ideas into an iterative improvement loop.

---

## Contents

<b>1</b>	<b>Purpose of This Document</b>	<b>2</b>
<b>2</b>	<b>How Each Book Helps with AppSec Process Modeling</b>	<b>3</b>
2.1	<i>This Is Service Design Doing</i> . . . . .	3
2.2	<i>Work the System</i> . . . . .	4
2.3	<i>Traction</i> . . . . .	5
<b>3</b>	<b>Recommended Reading Order for AppSec Modeling</b>	<b>7</b>
<b>4</b>	<b>Sketch: Modeling the Current AppSec Pipeline</b>	<b>8</b>
4.1	High-Level View of the AppSec Pipeline . . . . .	8
4.2	Service-Blueprint Perspective . . . . .	9
4.3	<i>Work the System</i> : SOP for Secure Pull Request Flow . . . . .	11

## 1 Purpose of This Document

This document has two goals:

1. To summarize how three books *Work the System* (Sam Carpenter), *Traction* (Gino Wickman), and *This Is Service Design Doing* can help with modeling Application Security (AppSec) processes.
2. To sketch how an existing AppSec pipeline —with CI/CD gates, GitHub Advanced Security (GHAS), secrets scanning, and related checks—can be modeled using:
  - Service blueprints (from *This Is Service Design Doing*),
  - Standard operating procedures (SOPs) in the style of *Work the System*.

The intent is to give you an actionable way to move from:

“We have tools and gates”

*to*

“We have a coherent, documented AppSec **system** that can be improved over time.”

**Navigation tip.** If you already know these books, you can skim Section 2 and jump straight to the AppSec-specific material in Sections 4 and 5.

## 2 How Each Book Helps with AppSec Process Modeling

We will briefly summarize how each of the three books contributes to AppSec process modeling:

- *This Is Service Design Doing* – Service blueprinting and journeys, especially from the developer’s point of view.
- *Work the System* – Thinking in terms of systems and SOPs.
- *Traction* – Organizational structure, accountability, and execution.

### 2.1 *This Is Service Design Doing*

#### Core Idea

*This Is Service Design Doing* is a practical guide to service design. It emphasizes:

- Understanding users and stakeholders through research,
- Mapping **customer journeys**,
- Designing services using **service blueprints**,
- Prototyping and iterating based on feedback.

A **service blueprint** typically includes:

- Customer actions (what the user does),
- Frontstage actions (what the user sees),
- Backstage actions (what the organization does behind the scenes),
- Supporting processes and systems,
- Evidence (artifacts, communications, logs).

#### For AppSec

This book is most helpful if your goal is:

**Design AppSec as a developer-centered service**—with clear journeys, touchpoints, and support systems.

**Navigation tip.** Use Section 3 as a guide if you want to see everything applied directly to your AppSec pipeline.

## 2.2 *Work the System*

### Core Idea

*Work the System* argues that every organization is a collection of **systems** and **subsystems**. To improve results, you should:

- Step outside of the “whirlwind” of daily operations,
- Identify key systems,
- Document them as simple written procedures,
- Improve those procedures incrementally.

The book emphasizes three core documents:

1. A **Strategic Objective** that defines what success looks like.
2. A set of **Operating Principles** that guide decisions.
3. A collection of **working procedures** that describe how recurring work is done.

Instead of viewing AppSec as a set of tools (GHAS, SAST, DAST, etc.), you define and document the **systems** that use those tools:

- Inputs (events, triggers, artifacts),
- Process (steps, decisions, owners),
- Outputs (approvals, tickets, metrics).

### For AppSec

For AppSec, *Work the System* provides a way to:

- Treat each AppSec flow (e.g. “secure PR”, “secrets handling”, “vulnerability triage”) as a **system**,
- Write down clear, step-by-step procedures that reflect how the work is actually done today,
- Slowly improve these procedures as you learn more, instead of trying to design a “perfect” process up front.

In practice, this means you can:

- Create an AppSec **Strategic Objective** that describes what “good” looks like (e.g. “We detect and remediate critical application vulnerabilities before they are exploitable in production, with minimal friction for developers.”).

- Define a handful of **Operating Principles**, such as:
  - “We integrate security checks into existing developer workflows wherever possible.”
  - “We prioritize fixing the highest-risk issues first.”
  - “We document processes in simple language and keep them as short as possible.”
- Document **working procedures** for key AppSec systems:
  - “Secure Pull Request” procedure,
  - “Secrets Management” procedure,
  - “Vulnerability Triage” procedure,
  - “Incident Response for AppSec Findings” procedure.

This aligns strongly with AppSec process modeling because it forces you to:

- Be explicit about **who does what, when, and how**,
- Turn ad-hoc practices into repeatable, improvable systems,
- Keep the focus on outcomes (e.g. reduced risk, faster remediation) rather than on tools alone.

## 2.3 *Traction*

### Core Idea

*Traction* introduces the Entrepreneurial Operating System (EOS), which focuses on:

- Vision (shared understanding of where the organization is going),
- People (right people in the right seats),
- Data (simple, objective metrics),
- Issues (identifying and solving root problems),
- Process (documented and followed systems),
- Traction (execution and accountability).

It is very operational and pragmatic, with tools such as:

- Accountability chart (who owns what),
- Rocks (90-day priorities),
- Scorecards (weekly metrics),
- Level 10 meetings (structured weekly meetings),
- Documented core processes.

## How It Helps AppSec

For AppSec, *Traction* helps you:

- Clarify **who owns AppSec processes**:
  - Who owns the secure SDLC?
  - Who owns the CI/CD pipeline gates?
  - Who owns vulnerability management?
- Define AppSec-related **Rocks**:
  - “Implement secrets scanning across all repos”,
  - “Document and roll out SOP for vulnerability triage”,
  - “Reduce mean time to remediate critical vulns by 30%.”
- Create **scorecard metrics**:
  - Number of open critical vulnerabilities,
  - Mean time to remediate by severity,
  - Percentage of services passing security gates,
  - Percentage of repos with GHAS enabled.
- Embed AppSec into the **leadership cadence**:
  - AppSec metrics appear on the weekly scorecard,
  - AppSec issues are raised and resolved in L10 meetings,
  - AppSec Rocks are reviewed quarterly.

## For AppSec

In short, *Traction* gives you an organizational frame so that AppSec is not “just a set of tools” but:

- Owned by specific people,
- Measured with specific metrics,
- Improved through a regular cadence of meetings and Rocks,
- Connected to the organization’s broader goals.

### 3 Recommended Reading Order for AppSec Modeling

If your specific goal is to model Application Security processes, a pragmatic order that also matches the navigation flow in Section 2 is:

1. ***This Is Service Design Doing***

Start here to learn the tools for mapping journeys and services. Your first outcome can be one or two **service blueprints** for key AppSec flows (for example, “secure PR” or “secrets handling”). These blueprints give you a developer-centered view of how AppSec shows up in day-to-day work.

2. ***Work the System***

Next, use *Work the System* to turn those blueprints into **SOPs**—clear, written procedures that can be followed and improved by the team. Here you move from “this is the journey” to “this is the system we run every time.”

3. ***Traction***

Finally, use *Traction* to plug these SOPs into a broader organizational system (EOS):

- clarify who owns each AppSec system (secure PR, secrets management, vulnerability triage, etc.),
- ensure AppSec KPIs show up on scorecards,
- set Rocks and metrics around AppSec improvements,
- create a regular cadence for reviewing and refining the processes.

This sequence moves from:

1. **Understanding the experience** (service design) – see Section 2, *This Is Service Design Doing*,
2. **Documenting the system** (SOPs) – see Section 2, *Work the System*,
3. **Embedding it organizationally** (EOS / Traction) – see Section 2, *Traction*.

**Navigation tip.** Treat this section as your reading roadmap. As you work through the books, follow the order above and use the cross-references into Section 2 to jump straight to the summaries and AppSec-specific notes. If you just want the integrated view of how all three books combine into one AppSec modeling approach, you can also jump directly to Section 5.

## 4 Sketch: Modeling the Current AppSec Pipeline

This section gives a concrete sketch of how to model an existing AppSec pipeline—with CI/CD gates, GHAS scans, and secrets scanning—using:

- Service blueprints, and
- *Work the System*-style SOPs.

The goal is not to capture every technical detail, but to create a model that:

- Is easy to understand for both AppSec and engineering leadership,
- Makes responsibilities and handoffs visible,
- Can be iteratively improved.

### 4.1 High-Level View of the AppSec Pipeline

At a very high level, your AppSec pipeline might look like:

- Developer writes code and opens a PR,
- CI pipeline runs:
  - Unit tests,
  - Static analysis (SAST),
  - Dependency scanning (SCA),
  - Secrets scanning,
  - Other checks as needed.
- Results are surfaced on the PR (GHAS, CI status checks),
- If all checks pass, the PR can be merged,
- If there are issues, the developer must address them or request an exception,
- Critical findings and exceptions feed into a vulnerability management process.

This high-level view can be turned into:

- One or more **service blueprints** that show the developer journey and the supporting systems,
- One or more **SOPs** that describe the step-by-step flows in more procedural detail.



## 4.2 Service-Blueprint Perspective

From a service-design perspective, consider the flow “Secure Pull Request and GHAS Scans”.

### Blueprint 1: Secure Pull Request and GHAS Scans

#### Customer (Developer) Actions

- Create feature branch,
- Implement changes,
- Commit and push,
- Open PR against main branch,
- Respond to feedback (including security feedback),
- Merge when approvals and checks pass.

#### Frontstage (Visible) Actions

- CI pipeline status on PR,
- GHAS alerts and annotations on PR,
- Comments from reviewers (including AppSec if applicable),
- Dashboard views (e.g. security overview in GitHub).

#### Backstage Actions

- CI jobs running tests and scans,
- GHAS running SAST and SCA checks,
- Secrets scanning jobs,
- Notification hooks to Slack/Teams or ticketing systems,
- Automation that creates tickets for certain findings.

#### Supporting Processes and Systems

- CI/CD platform configuration,
- GitHub repository settings and branch protections,
- GHAS configuration (enabled repos, rules),
- Secrets management policies,
- Vulnerability management tooling.

## **Evidence**

- PR history,
- CI logs,
- GHAS alerts and resolution status,
- Tickets created for critical vulns,
- Audit trail of approvals.

## **Blueprint 2: Vulnerable Dependency Management**

A second blueprint could focus on handling vulnerable dependencies:

### **Customer (Developer) Actions**

- Introduce or update a dependency,
- See an alert about a vulnerable dependency,
- Decide whether to update, replace, or request an exception,
- Implement the chosen mitigation,
- Verify that alerts are resolved.

### **Frontstage Actions**

- GHAS dependency alerts on PRs and in the repo,
- Notifications in dashboards or email,
- Ticket(s) for critical dependency issues,
- Documentation or guidance surfaced to developers.

### **Backstage Actions**

- GHAS scanning dependency manifests,
- Jobs that aggregate dependency findings,
- Automation that opens tickets for high/critical issues,
- Periodic jobs that rescan repos.

## Supporting Processes and Systems

- Dependency management tooling,
- Vulnerability database integrations,
- Ticketing system,
- Exception and risk-acceptance processes.

## Evidence

- Dependency alert history,
- Ticket history,
- Records of exceptions and approvals.

### 4.3 *Work the System*: SOP for Secure Pull Request Flow

Once you have a blueprint for the secure PR flow, you can translate it into a *Work the System*-style SOP.

#### **SOP: Secure Pull Request Flow**

**Purpose** Ensure that all changes merged into the main branch have passed appropriate security checks (GHAS, tests, etc.) and that exceptions are handled consistently.

**Scope** Applies to all repositories using the standard CI/CD pipeline and GHAS integration.

**Owner** AppSec lead (or designated pipeline owner).

#### **Procedure (Happy Path)**

1. Developer creates a feature branch from the main branch.
2. Developer commits changes with appropriate tests.
3. Developer opens a Pull Request to the main branch.
4. CI pipeline triggers automatically:
  - Unit tests,
  - GHAS SAST and SCA scans,
  - Secrets scanning,

- Other configured checks.
5. PR shows status checks:
    - All required checks must pass,
    - No blocking GHAS alerts for critical issues.
  6. If all checks pass and required reviewers approve, the PR is merged.

## Procedure (Exceptions)

1. If GHAS finds a critical issue:
  - Developer investigates and attempts to fix,
  - If not fixable within acceptable time, developer or team lead requests an exception.
2. Exception requests must include:
  - Description of the issue,
  - Justification for the exception,
  - Proposed mitigation and timeline.
3. AppSec reviews and either:
  - Approves the exception (with conditions),
  - Rejects the exception and requires a fix.
4. Approved exceptions are logged in the ticketing system and tracked until resolved.

## Records and Evidence

- GHAS scan reports attached to PRs,
- CI logs,
- Tickets for critical findings and exceptions,
- Audit trail of approvals and merges.

## Improvement Loop

- On a monthly or quarterly basis:
  - Review aggregate GHAS data,
  - Identify recurring issues,
  - Update the SOP and training materials.

**Navigation tip.** Treat Section 5 as your high-level implementation checklist; you can skim it first to get the big picture, then dive back into Sections 2 and 4 as needed.

## 5 Putting It All Together

In practice:

1. Use *This Is Service Design Doing* techniques to:
  - Map developer journeys,
  - Identify key AppSec touchpoints,
  - Build 1–3 service blueprints for your most important AppSec flows.
2. For each blueprint, write a *Work the System*-style SOP:
  - Clear purpose and scope,
  - Step-by-step procedures,
  - Owners and escalation paths,
  - Evidence and improvement loop.
3. Use *Traction* to:
  - Assign owners in an accountability chart,
  - Make AppSec-related improvements into Rocks,
  - Create scorecard metrics tied to your blueprints and SOPs,
  - Ensure AppSec is reviewed regularly at the leadership level.
4. Iterate:
  - Update blueprints as tooling or org structures change,
  - Refine SOPs based on real usage and incidents,
  - Adjust metrics and Rocks each quarter as needed.

The combination gives you:

- A **service view** of AppSec (developer experience, touchpoints),
- A **systems view** (SOPs and procedures you can improve),
- An **organizational view** (ownership, metrics, cadence).

Done well, this moves AppSec from a loose collection of tools and gates to a coherent, documented system that is easier to operate, explain, and improve.