

---

# ABAP - Enhancement, Tracing & Upgrade Intelligence

---

## 1 ANST – Automated Note Search Tool

**Purpose:**

Trace a transaction or program to identify underlying components (function modules, BADIs, classes) and propose relevant OSS Notes.

**Scenario:**

You execute VA01 and receive an error message — ANST traces runtime and suggests all notes or enhancements related to it.

**How to Use:**

1. Run ANST.
2. Enter transaction (e.g., VA01).
3. Reproduce issue.
4. Check “*Technical Component Tree*” → see enhancement spots, BADIs, includes.

**Pros:**

- Extremely accurate enhancement discovery.
- Links runtime trace with SAP Notes automatically.
- No debug skills required.

**Cons:**

- Requires reproducible transaction execution.
- Can generate large traces if not filtered.
- May miss deep custom code (Z-class) layers.

**When Not to Use:**

- For static code analysis — prefer SE84 or SCI.
- For performance tuning — prefer ST05/SAT.

---

## 2 SCOV – Test Coverage Analyzer

**Purpose:**

Evaluates which lines of code were executed during tests.

**Scenario:**

After applying a user exit enhancement, you run functional tests — SCOV reveals if your enhancement logic executed.

**How to Use:**

1. Run SCOV → Create new coverage session.
2. Execute your test (manual or automated).
3. Analyze which programs, includes, or methods executed.

**Pros:**

- Great for upgrade readiness checks.
- Ensures test completeness for custom logic.

**Cons:**

- Consumes system resources during long runs.
- Not effective for background jobs or RFCs.

**When Not to Use:**

- On production systems.
- For performance analysis (use ST05/SAT instead).

---

## 3 ST05 – SQL Trace

**Purpose:**

Capture all database SQL statements triggered by ABAP or CDS/AMDP.

**Scenario:**

Sales order creation is slow — you trace VA01 to identify SELECT \* or missing indexes.

**Pros:**

- Very precise at DB call level.
- Shows exact query, runtime, table, and plan.

**Cons:**

- Verbose output, needs expertise to interpret.
- Cannot trace across RFCs.

**When Not to Use:**

- On high-traffic production systems (heavy trace).
- For runtime logic flow (use ANST or SAT).

---

## 4 SAT – ABAP Runtime Analysis

**Purpose:**

Measure performance time per ABAP method, form, or function.

**Scenario:**

A BAdI in VA01 is slow — SAT identifies which method consumes time.

**Pros:**

- Accurate method-level timing.
- Works with both classical and OO code.

**Cons:**

- Doesn't capture HANA-side time for AMDPs.

**When Not to Use:**

- For HANA SQL analysis (use ST05 or SQLSCRIPT\_TRACE).

---

## 5 SE84 / SE85 – Repository Information System

**Purpose:**

System-wide metadata search (enhancements, BAdIs, exits, etc.).

**Scenario:**

Find all enhancement points for `SAPMV45A` (VA01).

**Pros:**

- No runtime needed.
- Covers all object types (BAdIs, includes, spots).

**Cons:**

- Static only (doesn't show dynamic BADI calls).

**When Not to Use:**

- When debugging runtime (use ANST).
-

## 6 SPAU / SPDD – Upgrade Adjustment Tools

### **Purpose:**

Reconcile modified or enhanced objects after SAP upgrade.

### **Scenario:**

After S/4 upgrade, customer exits in MV45AFZZ conflict with new release.

### **Pros:**

- Safeguards customizations during upgrade.
- Offers side-by-side comparison.

### **Cons:**

- Requires functional testing after adjustment.
- Can't handle bad coding merges automatically.

### **When Not to Use:**

- During normal development (only post-upgrade).
- 

## 7 SCU3 – Change History Analyzer

### **Purpose:**

View table-level change logs (Change Documents).

### **Scenario:**

You need to verify who changed delivery status in LIKP.

### **Pros:**

- Gives transparent audit of table changes.
- Works without debugging.

### **Cons:**

- Only logs if Change Document Object is active.

### **When Not to Use:**

- For transient runtime debugging (use ANST or SLG1).
- 

## 8 SE95 – Modification Browser

**Purpose:**

List all modified SAP standard objects.

**Scenario:**

Before upgrading, you analyze which SAP programs have direct Z-code insertions.

**Pros:**

- Comprehensive view of all modifications.
- Helps cleanup old manual changes.

**Cons:**

- Doesn't distinguish clean enhancements vs bad mods.

**When Not to Use:**

- For enhancement-only systems (use SPAU\_ENH).

---

## 9 SE18 / SE19 – BAdI Definition & Implementation

**Purpose:**

Core entry point for enhancement management.

**Scenario:**

Customer needs extra validation during billing — create new implementation for BADI\_SD\_BILLING.

**Pros:**

- Safe, upgrade-stable.
- Multiple implementations allowed (filterable).

**Cons:**

- Dynamic BAdIs can be hard to locate.

**When Not to Use:**

- For simple field manipulations — prefer implicit enhancement or user exit.

---

## 10 SE93 – Transaction Utility

**Purpose:**

Find the technical program behind a transaction.

**Scenario:**

You open VA01 → Find `SAPMV45A` to search exits inside.

**Pros:**

- Simple but powerful starting point for any trace.

**Cons:**

- Only reveals initial program, not subsequent includes.

**When Not to Use:**

- For indirect or service-based transactions (like Fiori apps).
- 

## 11 ST12 – Combined Trace

**Purpose:**

Unified ABAP + SQL trace capturing both logic and DB.

**Scenario:**

You want one trace to analyze both ABAP flow and HANA queries for VA01.

**Pros:**

- Single step for full-stack analysis.
- Easier than combining ST05 + SAT manually.

**Cons:**

- More resource intensive.

**When Not to Use:**

- For small lightweight tests (SAT alone suffices).
- 

## 12 SE24 + CL\_EXITHANDLER Breakpoint

**Purpose:**

Debug dynamically called BAdIs.

**Scenario:**

You don't know which BAdI triggers during VA01 → set breakpoint in

`CL_EXITHANDLER=>GET_INSTANCE.`

**Pros:**

- Finds runtime BAdI names.
- Accurate even for dynamic calls.

**Cons:**

- Requires debugging access.

**When Not to Use:**

- For production issues (use ANST trace).

---

## 13 SNOTE – OSS Note Implementation

**Purpose:**

Applies SAP correction notes.

**Scenario:**

After ANST trace, SAP suggests OSS note 1234567.

**Pros:**

- Direct implementation with dependency check.

**Cons:**

- Risky without proper testing.

**When Not to Use:**

- Directly in production; always validate in DEV first.

---

## 14 SCI / ATC – Code Inspector & Test Cockpit

**Purpose:**

Analyze code quality, performance, and upgrade readiness.

**Scenario:**

You need to find all SELECT \* or obsolete syntax in Z-code.

**Pros:**

- Automated code quality enforcement.

**Cons:**

- False positives in some legacy areas.

**When Not to Use:**

- For runtime trace — it's static only.
- 

## 15 SWO1 / BAPI Explorer

**Purpose:**

Explore business object interfaces and APIs.

**Scenario:**

Instead of direct table insert, you find a suitable BAPI.

**Pros:**

- Reduces risk of direct DB updates.

**Cons:**

- Some BAPIs are obsolete or incomplete.

**When Not to Use:**

- When no official BAPI exists — check modern APIs or CDS entities.
- 

## 16 /IWFND/TRACES & /IWBEP/TRACES

**Purpose:**

Trace OData service calls.

**Scenario:**

You debug a Fiori app failing on `POST SalesOrder`.

**Pros:**

- Maps OData calls to ABAP stack.

**Cons:**

- Requires Gateway expertise.

**When Not to Use:**

- For classic GUI transactions.

---

## 17 SLG1 – Application Log Viewer

### **Purpose:**

Check logs created by BAL (Business Application Log).

### **Scenario:**

Custom BAdI writes log for failed validations — review via SLG1.

### **Pros:**

- Central, simple log analysis.

### **Cons:**

- Needs logging logic in code.

### **When Not to Use:**

- If no BAL implemented (useless).

---

## 18 SQLSCRIPT\_TRACE / HDBSQLTRACE

### **Purpose:**

Analyze AMDP or HANA SQLScript execution plan.

### **Scenario:**

AMDP is slow — check HANA trace to see query pushdown.

### **Pros:**

- Low-level visibility into SQL engine.

### **Cons:**

- Requires HANA Studio access.

### **When Not to Use:**

- Non-HANA DBs (irrelevant).
-

## 19 SU3 / SU01 Parameters

**Purpose:**

Review or maintain user-specific parameters.

**Scenario:**

ANST trace differs for user A and B — check SU3 parameters.

**Pros:**

- Identifies user-dependent behavior.

**Cons:**

- Only applicable for GUI users.

**When Not to Use:**

- For background RFC or job debugging.
- 

## 20 SE84 → Enhancements Directory

**Purpose:**

Central listing of all enhancement options and implementations.

**Scenario:**

List all implicit enhancement points in standard program SAPMV45A.

**Pros:**

- Covers all types: implicit, explicit, BAdI.

**Cons:**

- No runtime sequence.

**When Not to Use:**

- For dynamic BAdIs — use SE24 + CL\_EXITHANDLER.
- 

## Quick Summary Table

<b>TCode</b>	<b>Focus</b>	<b>Pros</b>	<b>Cons</b>	<b>When Not to Use</b>
ANST	Enhancement & Note trace	Runtime precision	Large traces	Static analysis
SCOV	Coverage	Great for test completeness	Heavy load	On production
ST05	DB trace	SQL-level details	Complex output	Runtime logic
SAT	Runtime Analysis	Accurate timing	No DB info	For HANA trace
SE84	Repository search	Complete listing	Static only	Runtime flow
SPAU	Upgrade adjust	Keeps mods intact	Needs manual effort	Non-upgrade time
SCU3	Change Logs	Audit trail	Only logged tables	Runtime debugging
SE95	Mod Browser	All modifications	No hierarchy	Enhancement-only
SE18	BAdI Mgmt	Stable enhancement	Dynamic confusion	For small logic
SE93	TCode lookup	Fast mapping	One-level only	Fiori services
ST12	Combined trace	Full-stack view	Heavy	Small tests
SE24	BAdI Debug	Finds dynamic BAdIs	Needs debug	Production
SNOTE	Note Mgmt	Easy OSS fixes	Risk of wrong import	Prod without test
ATC	Code Check	Quality enforcement	False positives	Runtime debug
SWO1	API Discovery	Safe BAPI use	Outdated BAPIs	None exist
/IWFND/TRACES	OData trace	Fiori debugging	Gateway-specific	GUI Txns
SLG1	App Log	Post-run review	Code-dependent	No BAL used
SQLSCRIPT_TRACE	AMDP trace	SQL engine depth	HANA only	Non-HANA DB
SU3	User Params	Runtime variation	User-specific	Background jobs
SE84 Enh	Directory	All enhancement types	Static	Runtime sequence

---

Excellent — let’s complete this **enterprise-grade SME guide** with the “**Usage Matrix**” section.

This section will show *how and when* to use each tool across the **ABAP Enhancement Lifecycle** — from discovery to post-upgrade validation — just like a senior technical architect would plan it.

---

## Usage Matrix –TCodes Across the ABAP Enhancement Lifecycle

---

### Lifecycle Phases

- 1. **Discovery & Scoping** – Identify where to enhance
  - 2. **Design & Implementation** – Apply the enhancement (BAdI, User Exit, Implicit, etc.)
  - 3. **Testing & Validation** – Ensure functional correctness
  - 4. **Performance Optimization** – Fine-tune ABAP, CDS, AMDP layers
  - 5. **Upgrade & Regression Assurance** – Keep custom code intact and compliant
  - 6. **Runtime Support & Audit** – Monitor logs, errors, and coverage
- 

### 1 Discovery & Scoping Phase

Objective	Recommended TCodes	How to Use	Pros	Cautions
Identify technical component for a transaction	ANST	Run trace for transaction, get component, exits, BADIs	Pinpoints runtime logic	Avoid very broad trace without filters
Find enhancement points or BADIs statically	SE84 / SE85	Search → Enhancements → Enhancement Spots	Covers all spots & BADIs	Static only
Check actual BAdI runtime	SE24 (CL_EXITHANDLER)	Set breakpoint in GET_INSTANCE	Identifies dynamic BADIs	Debug authorization required
Find transaction’s main program	SE93	Enter TCode → view program name	Quick navigation	Doesn’t reveal submodules
List existing modifications	SE95	Analyze by object class	Prepares for safe enhancement	Too technical for business users

---

### 2 Design & Implementation Phase

Objective	Recommended TCodes	How to Use	Pros	Cautions
Implement BAdI or enhancement	<b>SE18 / SE19</b>	Find definition → create Z-impl	Stable, object-oriented	May require filter setup
Create implicit/explicit enhancements	<b>SE80 → Include → Enhance</b>	Insert code within standard	No modification	Maintain naming standards
Add field-level change logic	<b>SE11 → Append / SE14</b>	Extend structure/tables	Upgrade-safe if append	Avoid modification of standard table
Use available APIs	<b>SWO1 / BAPI Explorer</b>	Search by object name	Safe alternative to DB update	BAPIs may be obsolete

### 3 Testing & Validation Phase

Objective	Recommended TCodes	How to Use	Pros	Cautions
Verify runtime execution	<b>SCOV</b>	Start coverage → run test	Confirms logic executed	Heavy for long tests
Capture trace of program + SQL	<b>ST12</b>	Combine ABAP + SQL trace	Full stack timing	System load
Check message logs	<b>SLG1</b>	View BAL logs by Object/Subobj	Central log access	Needs BAL implemented
Validate user-dependent behavior	<b>SU3 / SU01</b>	Compare parameter sets	Detect user-specific variance	Only GUI users

### 4 Performance Optimization Phase

Objective	Recommended TCodes	How to Use	Pros	Cautions
Check SQL performance	<b>ST05 / SQLSCRIPT_TRACE</b>	Trace DB layer / HANA SQL	Detect slow queries	Avoid on production
Measure ABAP timing	<b>SAT</b>	Analyze report/module	Detect slow subroutines	No SQL data
Analyze full stack (ABAP + DB)	<b>ST12</b>	Unified runtime trace	End-to-end insight	Large data footprint
Static quality check	<b>SCI / ATC</b>	Run inspection / check variant	Detect SELECT *, obsolete code	Might raise false positives

### 5 Upgrade & Regression Assurance Phase

Objective	Recommended TCodes	How to Use	Pros	Cautions
Reconcile modifications	<b>SPAU / SPDD</b>	Adjust modified objects	Retains customer logic	Manual validation required
Check coverage post-upgrade	<b>SCOV</b>	Run regression tests	Ensures test completeness	High runtime
Validate quality and syntax	<b>ATC</b>	Run with upgrade variant	Detect obsolete statements	Ignore non-critical warnings
Check OSS fixes	<b>SNOTE</b>	Implement or review Notes	Syncs system with SAP fixes	Wrong note may conflict

## 6 Runtime Support & Audit Phase

Objective	Recommended TCodes	How to Use	Pros	Cautions
Trace issues in production	<b>ANST</b>	Capture runtime trace	Quick component identification	Filter carefully
Analyze change history	<b>SCU3</b>	View change document log	Audit compliance	Requires active change doc
Log business messages	<b>SLG1</b>	Filter by object	Fast incident resolution	Only if BAL used
Trace Fiori/OData runtime	<b>/IWFND/TRACES, /IWBEP/TRACES</b>	Execute service → review	Links Fiori error to backend	Not for GUI txns
View active enhancements	<b>SE84 → Enhancements</b>	List enhancement impls	Transparency	Static view only

## Expert Flow – ABAP Enhancement Intelligence

Stage	Tool Flow	Outcome
Discover Enhancement	SE93 → ANST → SE84	Find where to enhance
Design Enhancement	SE18 / SE19 → SE80 (Enh Points)	Implement logic safely
Validate Execution	SCOV → SLG1 → SU3	Verify behavior
Optimize Performance	ST12 → ST05 → SAT	Measure performance
Ensure Upgrade Safety	SPAU → ATC → SCOV	Maintain continuity
Audit & Support	SCU3 → SLG1 → /IWFND/TRACES	Ongoing monitoring

## Pro-Tier Tips & Tricks

Tip	Description
<b>ANST + SE24 Combo</b>	Use ANST first to narrow down program, then set breakpoint in CL_EXITHANDLER to pinpoint runtime BADI.
<b>SCOV + ATC Synergy</b>	Run SCOV after ATC cleanup → guarantees all cleaned code executes in test coverage.

Tip	Description
<b>SPAU + SE95 Pairing</b>	SE95 lists modifications → SPAU helps reconcile them post-upgrade.
<b>ST05 Filter Discipline</b>	Always trace by <i>user + time window</i> to avoid irrelevant SQL noise.
<b>Never Modify Standard Directly</b>	Prefer enhancement framework (BAdI, implicit, extension include).
<b>SLG1 Logging Culture</b>	Always integrate BAL logs in enhancements — invaluable during testing and ANST tracing.

---

### Caution (When Not to Use Certain Tools)

Tool	Avoid Using When	Reason
ANST	In mass/batch jobs	Trace overload
SCOV	On production	High runtime
ST05 / ST12	During peak usage	DB performance impact
SPAU	Mid-upgrade	Can corrupt adjustment state
SQLSCRIPT_TRACE	Non-HANA systems	Not applicable
SE24 (Debug)	Production systems	Risk of session lock
SLG1	Without BAL logging	Empty result
SNOTE	Without sandbox test	May override local code

---