CLEAN-CORE.IO · THE COMPLETE EXPLAINER

SAP Clean Core, explained without the jargon

What it means, why it suddenly matters, and what to actually do about your custom ABAP. Starts from nothing — no SAP background needed — and goes as far as grading every object in your estate.

---

7 PARTS ABOUT 20 MINUTES EVERY TERM DEFINED BEFORE USE

CLEAN-CORE.IO/CLEAN-CORE-EXPLAINED

Contents

THE SHORT ANSWER

Clean Core means running SAP standard software without modifying it, and adding your own behaviour only through interfaces SAP has formally released and promised to keep stable.
The purpose is not tidiness. It is that upgrades stay routine instead of becoming projects, and that a move to the cloud remains possible at all — because SAP's cloud offerings simply do not run the older techniques. Everything below explains how to tell which of your code is affected, and what to do with each kind.

PART 1

The idea, from scratch

No SAP knowledge assumed. If you already know what a modification is, skip to Part 3 — nothing here will surprise you.

1.1 What is "the core", and why does it need to be clean?

> **The core is the standard SAP software your company runs. It is clean when nobody has altered it — only extended it from the outside.**

SAP ships a large, standard business system. Every customer gets the same one. Almost no company can use it entirely as delivered, because every company does something slightly differently — a discount rule, an approval step, a report the auditors want in a particular shape.

For thirty years the obvious answer was to change the system itself. You could open SAP's own programs and edit them, or hook your own code directly into them. It worked, it was fast, and it is the reason a great many SAP systems today contain tens of thousands of pieces of custom code.

The bill arrives at upgrade time. When SAP ships a new version, it replaces its own programs — and every place where somebody edited them has to be found, understood, and reconciled by hand. A system with heavy modifications can turn a routine update into a project lasting months.

"Clean Core" is the discipline of getting the benefit of customisation without paying that bill: leave the standard software untouched, and put your own logic beside it, connected through interfaces SAP has promised to keep stable.

REMEMBER THIS

The whole idea in one sentence

Do not change the standard. Extend it from outside, through doors SAP has promised not to move.

1.2 What is custom code, and why is there so much of it?

> **Custom code is the software your company wrote itself inside SAP, usually in a language called ABAP. Most of it exists for good reasons.**

ABAP is SAP's own programming language. Programs written in it live inside the SAP system rather than on a separate server, which historically made them fast, convenient, and dangerously easy to entangle with SAP's own code.

Custom objects are conventionally named with a leading Z or Y — so a report called ZFI_INVOICE_CHECK is yours, and one called RFBILA00 is SAP's. That naming convention is the single most useful thing to know when you first look at an SAP system.

It is worth being fair to the people who wrote all this. Custom code is usually not carelessness; it is the accumulated record of decisions the business needed and the standard did not offer at the time. The problem is not that it exists. The problem is where it sits and what it touches.

ABAP	Advanced Business Application Programming — SAP's programming language, running inside the SAP system itself.
Z-object / Y-object	A custom object. The leading Z or Y is a naming convention reserved for customers, so SAP's own updates never collide with it.
S/4HANA	SAP's current generation of business software. The move to it is what forces most organisations to confront their custom code.

1.3 What actually goes wrong?

> **Three things: upgrades become expensive, cloud migration becomes impossible, and nobody can say which risk is real.**

The first is upgrade cost, described above. The second is sharper: SAP's cloud offerings simply do not permit the old techniques. Code that reads a standard table directly, writes files to the application server, or calls an unpublished function will not run there. It is not discouraged — it does not compile.

The third problem is the quiet one. A system with 40,000 custom objects contains some that are business-critical, some that nobody has executed since 2014, and no reliable way to tell them apart without looking. Programmes stall here, not on the technology.

WATCH OUT

The trap in the middle

Most modernisation projects do not fail on the difficult objects. They fail on not knowing which objects are difficult — so effort is spread evenly across code that deserves very different treatment.

PART 2

The vocabulary you actually need

Clean Core discussions are dense with terminology, and most of it is simpler than it sounds. These are the terms you will meet in the first hour.

2.1 Twelve terms, plainly

> **Learn these and most SAP architecture conversations become followable.**

Released API	An interface SAP has formally promised to keep stable, with a documented contract and a deprecation process. Building on one is what makes an extension upgrade-safe. Building on anything else is a private arrangement SAP never agreed to.
Extensibility	The supported ways of adding your own behaviour without modifying the standard. SAP distinguishes several, described in Part 3.
In-app extensibility	Your code runs inside the SAP system, but only through released interfaces. Close to the data, tightly governed.
Side-by-side extensibility	Your code runs outside the SAP system, on a separate platform, and talks to SAP over published interfaces. More freedom, more moving parts.
SAP BTP	Business Technology Platform — SAP's cloud platform where side-by-side extensions live. Think of it as the sanctioned place to put things that no longer belong inside the ERP.
ABAP Cloud	The restricted ABAP dialect for the cloud era. It permits only released interfaces, which is precisely why code written for it survives upgrades.
RAP	ABAP RESTful Application Programming Model — the modern way to build an application in ABAP Cloud. Used for in-app extensions.
CAP	Cloud Application Programming Model — SAP's framework for building services on BTP, typically in Node.js or Java. Used for side-by-side extensions.
Modification	A change to SAP's own code. The thing Clean Core exists to eliminate. Distinct from an extension, which leaves the standard intact.
Technical debt	The future cost of a decision that was convenient at the time. In this context: every shortcut that has to be unwound before a cloud move.
abapGit	An open-source tool for moving ABAP code in and out of Git. Relevant because it is how generated code gets into a real system for review.
ADT	ABAP Development Tools — the Eclipse-based environment where modern ABAP is actually written and tested.

JARGON, DECODED

"Released" is the load-bearing word

If you remember one term from this page, make it this one. Almost every Clean Core rule reduces to: are you using something SAP released, or something you found?

PART 3

The five dimensions

Clean Core is not only about code. SAP frames it across several dimensions, and a programme that addresses only the code dimension tends to stall on one of the others.

3.1 Where "clean" applies

> **Software stack, extensions, data, integrations and processes — each can be clean or dirty independently.**

It is tempting to treat Clean Core as a code-cleanup exercise, because code is visible and countable. In practice a programme is only as clean as its weakest dimension: pristine extensions built on a tangle of point-to-point integrations still block an upgrade.

THE FIVE DIMENSIONS, AND WHAT "CLEAN" MEANS IN EACH

DIMENSION	CLEAN MEANS	TYPICAL SMELL
Software stack	Current release, no modifications to SAP code	Years behind, upgrade repeatedly deferred
Extensions	Built only on released interfaces	Direct table access, unpublished function calls
Data	Governed, deduplicated, with clear ownership	Custom tables shadowing standard ones
Integrations	Via published APIs and an integration layer	Point-to-point file drops and database links
Processes	Close to the standard, deviations justified	Every process customised, nobody remembers why

IN PRACTICE

Where to start

Extensions, almost always. They are the dimension you can measure objectively, and progress there produces evidence that funds the rest.

PART 4

The decision that matters: in-app or side-by-side

For any given piece of custom code there is one architectural question worth arguing about. This part is for practitioners.

4.1 RAP or CAP — how to actually decide

> **Data gravity and transactional coupling pull in-app; independent lifecycle and non-SAP concerns push side-by-side.**

The choice is not a matter of taste, and it is not "cloud is modern so everything goes to BTP". It follows from what the object does.

Logic that reads and writes SAP business data in the same transaction — a validation on a sales order, a derivation during posting — belongs in-app, as RAP. Moving it outside means network round trips inside a transaction boundary, which is both slow and fragile.

Logic that serves a different audience, changes on a different schedule, or needs libraries the ABAP stack does not have — a partner portal, a machine-learning scoring service, a mobile back end — belongs side-by-side, as CAP on BTP.

Between those poles sits a large grey zone, and this is where honest architecture happens. The tie-breaker worth applying: if the extension must be deployed in lockstep with the ERP to remain correct, it is in-app. If it can be released on its own cadence, it is side-by-side.

A DECISION AID, NOT A RULE BOOK

SIGNAL	POINTS TO
Reads and writes SAP tables in one transaction	In-app (RAP)
Needs sub-second access to business data	In-app (RAP)
Extends a standard SAP business object	In-app (RAP)
Serves non-SAP users or systems	Side-by-side (CAP)
Has its own release cycle	Side-by-side (CAP)
Needs libraries or runtimes ABAP does not offer	Side-by-side (CAP)
Would put unpredictable load on the ERP	Side-by-side (CAP)

FOR THE ADVANCED READER

The successor question

Before either route, ask whether the custom object should exist at all. A meaningful share of legacy code duplicates functionality the standard has since acquired. Retiring an object is cheaper than modernising it, and the analysis effort is the same.

4.2 The hard cases nobody warns you about

> **Some techniques have no cloud equivalent at all. Recognising them early saves the most time.**

A handful of patterns are not merely discouraged in the cloud — they have no replacement, and code relying on them needs a rethink rather than a translation.

PATTERNS WITH NO DIRECT CLOUD SUCCESSOR

PATTERN	WHY IT BREAKS	WHAT REPLACES IT
Application-server file access	No file system in ABAP Cloud	Object storage or an integration service
Direct writes to standard tables	Bypasses business logic and validation	Released APIs for the business object
Batch input / call transaction	Screen scraping is not available	Released APIs or mass-change services
Dynamic program generation	Not permitted	Rules in configuration, not in generated code
Unpublished RFC calls	The contract was never guaranteed	A released equivalent, or a new one

WATCH OUT

Estimate these separately

These objects distort averages badly. An estimate that treats a file-writing extractor as equivalent to a straightforward report will be wrong by an order of magnitude on that one item.

PART 5

Grading your code: the A–D model

Clean Core guidance has moved on from a binary "clean or not" to a four-grade classification. This is the part that turns an opinion into a work plan — and it is the model we walk through in detail in our SAP Community write-up.

5.1 Four grades, and what each one costs you

> **Every custom object lands in one of four grades, and the grade tells you what to do with it.**

The grades describe what an object depends on, not how well it is written. A beautifully engineered report that writes directly to a standard table is still grade D, because the dependency is what breaks in the cloud — not the code quality.

This matters for estimating. Grades A and B need review; grade C needs an interface; grade D needs a decision about whether the object should exist at all. Treating them as one bucket is how modernisation budgets go wrong.

THE FOUR GRADES AND THE WORK EACH IMPLIES

GRADE	WHAT IT DEPENDS ON	WHAT TO DO	EFFORT
A	Released SAP APIs and extension points	Nothing — this is where you want to be. Build here.	None
B	Classic SAP APIs, still SAP-recommended	Keep, and plan for released successors over time.	Low
C	Internal SAP APIs — conditionally clean	Wrap behind a clean interface, verify each release.	Medium
D	Direct writes to standard tables, unreleased dependencies, classic screen programs, low-level kernel calls	Replace or re-architect. These are the upgrade blockers.	High

IN PRACTICE

The working sequence

Identify each object → look it up in the SAP Cloudification Repository → assign a grade → decide the remediation (map to a released API or CDS view, wrap it, or re-architect) → confirm with SAP ADT and the ABAP Test Cockpit.

FOR THE ADVANCED READER

The grades line up with ATC priorities

They are not a parallel universe. The A–D grades map onto ABAP Test Cockpit priorities, so a classification exercise and an ATC run should broadly agree — and where they disagree, that disagreement is itself worth looking at.

WATCH OUT

Our grade is an estimate, not a verdict

The A–D grade Clean-Core.io produces is an experimental preview estimate — a fast orientation aid, not an authoritative SAP ATC classification. Always confirm with SAP ADT and ATC for your specific target release before acting on it.

5.2 You cannot clean what you cannot see

> **Before remediation comes measurement. Without numbers per object, a Clean Core programme has no way to show progress or defend a decision.**

The extensibility dimension is the one you can actually measure, which is why it is the sensible place to start. The useful figures are not "how many objects do we have" but questions with consequences: how many are grade D, how many objects are touched by the next upgrade, how many were executed at all in the last year.

That last one deserves emphasis. A meaningful share of any large custom-code estate is dead — written for a process that no longer exists, kept because deleting felt riskier than ignoring. Usage data turns that from a suspicion into a number, and retiring an object is always cheaper than modernising it.

Once the figures exist per object, progress becomes reportable: grade D count going down over quarters is a sentence a steering committee can act on, in a way that "the team is working on Clean Core" is not.

REMEMBER THIS

Three numbers worth tracking

Objects by grade (is the D pile shrinking), objects never executed (what can simply be retired), and objects on the upgrade path (what forces a decision this year).

PART 6

How Clean-Core.io helps, concretely

Seven stages, one ABAP object at a time. Each is listed with what it produces, what it saves you, what it costs — and where it stops. The last column is the one worth reading.

STAGE 1 Evidence-based analysis

A Clean Core score, an A–D readiness estimate, a list of findings with line numbers, the database coupling, a code inventory, and complexity and criticality scores — sealed as a signed, immutable Run.

BENEFIT

Replaces "we think this object is risky" with a per-object finding somebody can check. A deterministic engine parses the source before any AI is involved, so the evidence does not depend on a model's mood.

EFFORT

One click, about two minutes. Costs one of your five transformations. Re-analysing the same source is free.

WHERE IT STOPS

The A–D grade is an experimental preview estimate, not an authoritative ATC classification. Confirm with SAP ADT and ATC for your target release.

STAGE 2 Extensibility routing

A recommended route — in-app ABAP Cloud (RAP) or side-by-side BTP (CAP) — with a confidence score and the reasoning that produced it.

BENEFIT

The RAP-or-CAP argument, resolved per object with stated criteria instead of preference. The reasoning is inspectable, so it survives a review rather than only a demo.

EFFORT

Included in the analysis. No extra step, no extra unit.

WHERE IT STOPS

It is a recommendation from static evidence. It does not know your organisation's platform strategy, licensing, or team skills — all of which legitimately override it.

STAGE 3 Solution design

A target architecture mapped onto the recommended route, released SAP APIs proposed in place of direct table access, and the non-functional requirements spelled out.

BENEFIT

Turns "use released APIs" into named APIs from the SAP Business Accelerator Hub for the tables your code actually touches.

EFFORT

Two to three minutes. Included — no further unit is charged.

WHERE IT STOPS

API mapping is grounded in SAP's published catalogue. Where no released successor exists, it says so rather than inventing one.

STAGE 4

Code transformation

A RAP or CAP implementation generated from the legacy source, shown side by side with the original, scroll-synchronised.

BENEFIT

The first compliant draft, in minutes rather than days — and reviewable statement by statement instead of as a black box.

EFFORT

Three minutes. Included.

WHERE IT STOPS

A draft for an architect to review, not a deployment artefact. It is generated by a third-party AI model and provided as-is. Nothing here removes the need for an expert to read it.

STAGE 5

Test generation and execution

ABAP Unit test classes against the transformed logic, executed in an isolated sandbox with per-case results.

BENEFIT

Modernised code arrives with tests attached, so the review has something to run rather than only something to read.

EFFORT

Two minutes. Included.

WHERE IT STOPS

Tests are generated from the code, so they encode its behaviour — including any bug it already had. They verify the transformation, not the original business requirement.

STAGE 6

Documentation

BPMN 2.0 process diagrams for import into SAP Signavio or SAP Build, plus business-facing procedures and control points, exportable to Confluence.

BENEFIT

The documentation that normally never gets written, produced as a by-product of the analysis rather than as a separate project.

EFFORT

Two minutes. Included.

WHERE IT STOPS

Generated from the code. It describes what the object does, not why the business wanted it.

STAGE 7

Delivery and audit evidence

An abapGit-compatible ZIP with sources and tests, plus a signed audit evidence pack recording exactly what was analysed, by which engine and catalogue version, and what it concluded.

BENEFIT

The package goes into your own Eclipse ADT environment for compilation and verification — and the signature makes the analysis tamper-evident, which is what lets a recommendation hold up in a governance review.

EFFORT

One minute. Included.

WHERE IT STOPS

The signature proves the analysis was not altered after the fact. It does not certify that the conclusion is correct — that is what your architect is for.

Honest scope

The governing principle of this project is *belegt, nicht behauptet* — proven, not claimed. A capability list without limits is a claim, so here are the limits.

Is this a code translator?	No. It is an evidence-first accelerator. The generated code matters less than the evidence and the reasoning that led to it.
Does it replace architecture governance?	No. It does not replace enterprise-architecture approval, SAP release checks, privacy review, penetration testing, or production migration governance.
Can I deploy the output to production?	Not directly. Every artefact is a draft for expert evaluation. Review, test and approve it with a qualified SAP architect first.
What happens to my source code?	It is processed by the Google Gemini API for the AI stages and is not used to train Google's models under those terms. Your data is stored in the EU (europe-west1, Belgium). Bring your own Gemini key and it is used exclusively through a server-side proxy, encrypted at rest with AES-256-GCM, never exposed to the browser.
What does it cost?	Nothing. Five transformations per account, no locked features, no paid tier. Your own Gemini key removes the limit entirely.

PART 7

Questions people actually ask

Six answers, straight.

What does SAP Clean Core mean?

Clean Core means running SAP standard software without modifying it, and adding your own behaviour only through interfaces SAP has formally released and promised to keep stable. The purpose is that upgrades stay routine instead of becoming projects.

Why is Clean Core suddenly important?

Because SAP's cloud offerings do not permit the older techniques. Code that reads standard tables directly, writes to the application server, or calls unpublished functions does not run there — so the move to S/4HANA in the cloud forces the question that on-premise systems allowed organisations to defer.

What is the difference between in-app and side-by-side extensibility?

In-app extensions run inside the SAP system using ABAP Cloud and RAP, and suit logic that is transactionally coupled to SAP data. Side-by-side extensions run on SAP BTP, typically as CAP services, and suit logic with its own release cycle or a non-SAP audience.

Do I have to rewrite all my custom code?

No. A meaningful share of legacy custom code can be retired because the standard has since acquired the functionality, and much of the rest needs adjustment rather than rewriting. The expensive mistake is treating every object as equivalent instead of classifying them first.

How do I know which of my objects are the problem?

By analysing them against the Clean Core criteria — which interfaces they use, how they touch data, and whether the patterns they rely on exist in the cloud at all. That analysis is what Clean-Core.io automates, producing evidence per object rather than an overall impression.

Is Clean-Core.io free?

Yes. It is a free community project. Every account gets five transformations, and connecting your own Google Gemini API key removes the limit entirely. There is no paid tier and no locked feature.

Clean-Core.io — SAP Clean Core, explained without the jargon. The current version of this document is always at clean-core.io/clean-core-explained. Free to read, print and pass on.

Longer write-ups on the A–D model and on measuring an extensibility programme are on the SAP Community, linked from the web version. Corrections are welcome at info@clean-core.io — this document is better for them.