Wavect®

ENTERPRISE RWA / BLOCKCHAIN

ANONYMISIERTE CASE STUDY

Wie wir mit 1-2 Entwicklern eine 50-köpfige Agentur überholten

Ein vollständiges Enterprise-RWA-Protokoll. Nach einem kompromittierten Launch-Prototyp und rund 50.000 € Verlust laut Kunde: 6 Monate Rückstand, 1-2 Entwickler - wenige Monate später vorn bei Funktionsumfang, Security und UX.

SMART CONTRACTS

AI-GESTÜTZTE UMSETZUNG

KURZE FEEDBACK-ZYKLEN

REALER SECURITY-VORFALL

DER AUSGANGSPUNKT

6 MONATE

VORSPRUNG

Es ging nicht um ein Feature. Es ging um das Protokoll.

Enterprise-RWA verband Smart Contracts, Systemlogik, Integrationen und Enterprise-Prozesse zu einem Gesamtsystem.

DIE ENTSCHEIDENDE KOMPLEXITÄT

Jede Änderung berührte mehrere Ebenen gleichzeitig.

- **On-chain korrekt**
  Smart Contracts und Zustände mussten deterministisch funktionieren.
- **Off-chain integrierbar**
  APIs, Events und Betriebsprozesse mussten anschlussfähig bleiben.
- **Für Menschen bedienbar**
  Komplexität durfte nicht beim Enterprise-Nutzer landen.

RWA = Real-World Assets: reale Vermögenswerte, die über kontrollierte digitale und On-chain-Prozesse abgebildet werden.

Der Vergleich wirkte auf dem Papier eindeutig.

Wir konnten nicht über Headcount gewinnen. Also optimierten wir das System, in dem Engineering stattfand.

~50

SOFTWARE-AGENTUR

Größe der konkurrierenden Agentur mit eingespielter Organisation und deutlich mehr nomineller Kapazität.

+6 Monate

VORSPRUNG

Der Wettbewerber hatte bereits ein halbes Jahr Produkt- und Implementierungsvorsprung.

1-2

WAVECT ENGINEERS

Kleines Senior-Kernteam, das AI als Multiplikator und nicht als Ersatz für Verantwortung einsetzte.

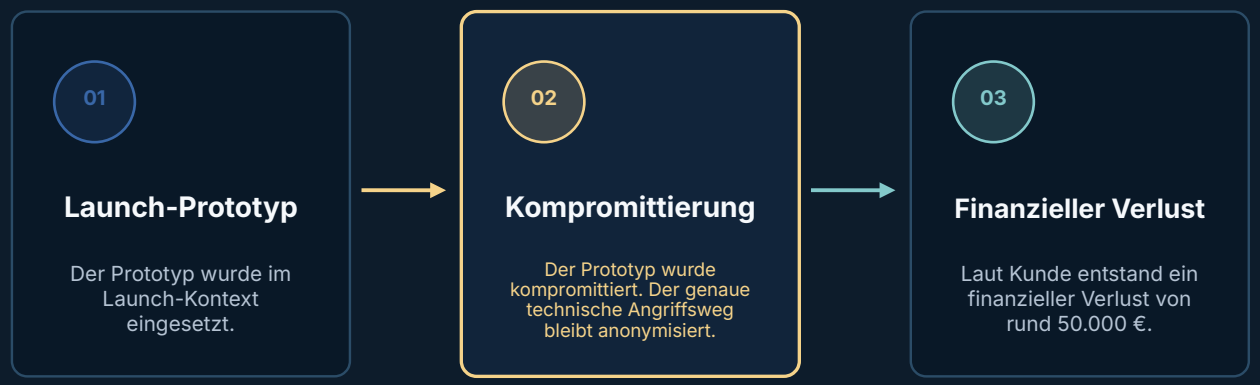
Die einzige sinnvolle Zielgröße: mehr verifizierte Lernzyklen pro Monat.

Hinweis: Die rund 50 Personen beziehen sich auf die Unternehmensgröße der Agentur, nicht zwingend auf die gleichzeitig eingesetzte Projektbesetzung.

Der Security-Vorfall war kein theoretisches Risiko: rund 50.000 € Verlust.

Der Launch-Prototyp wurde kompromittiert. Nach Angaben des Kunden entstand dabei ein finanzieller Verlust von rund 50.000 €.

VORHERIGER LAUNCH-KONTEXT



Der genaue Angriffsvektor bleibt aus Vertraulichkeitsgründen anonymisiert.

~50.000 €

FINANZIELLER VERLUST

NACH ANGABEN DES KUNDEN

Security war damit keine abstrakte Qualitätsdimension, sondern ein unmittelbares Geschäfts- und Vermögensrisiko.

Für das neue Protokoll musste jeder Slice nicht nur funktionieren, sondern seine Sicherheitsannahmen beweisbar machen.

Das Rennen wurde nicht in Personentagen entschieden, sondern in Loop-Zeit.

Je kürzer der Weg von einer Annahme zu verifiziertem Kundenfeedback, desto schneller wächst die tatsächliche Produktreife.

KLASSISCHE BATCH-DELIVERY

01 Spezifikation
02 Übergabe
03 Implementierung
04 Integration
05 Review & Feedback

Feedback kommt spät. Änderungen werden teuer.

WAVECT: VERIFIZIERTER LOOP

Feedback fließt sofort in den nächsten Slice.

Nicht mehr Personen. Mehr verifizierte Loops.

Jeder Slice musste fachlich, technisch und operativ funktionieren.

Das Team lieferte keine isolierten Code-Pakete, sondern kleine, überprüfbare Protokollfortschritte.

01 Scope schneiden

Kleinste Einheit wählen, die echten Produktwert und testbares Verhalten liefert.

02 Protokoll bauen

Smart Contracts, Zustände und angrenzende Services gemeinsam implementieren.

03 Beweise erzeugen

Tests, Invarianten und Grenzfälle parallel zum Code entwickeln.

04 Security prüfen

Findings früh durch Engineering und internen Security Engineer schließen.

05 Workflow zeigen

Funktionierende Enterprise-Flows statt abstrakter Statusberichte demonstrieren.

06 Lernen & releasen

Feedback in den nächsten Slice überführen; Regressionen dauerhaft absichern.

Der Loop endete erst, wenn Verhalten, Sicherheit und Bedienbarkeit gemeinsam belastbar waren.

AI beschleunigte das Engineering - nicht die Verantwortung.

Die Produktivität kam aus Parallelisierung, semantischem Routing, kuratiertem Kontext und konsequentem Senior Ownership.

AI BESCHLEUNIGT

- **Repository & Protokoll verstehen**
  Kontext schneller kartieren und Abhängigkeiten sichtbar machen.
- **Implementierungsvarianten erzeugen**
  Optionen parallel explorieren, statt seriell zu warten.
- **Tests & Edge Cases erweitern**
  Mehr relevante Pfade früher in die Prüfung bringen.
- **Refactoring & Dokumentation**
  Wiederholbare Arbeiten günstiger und kontinuierlicher erledigen.
- **Traces & Findings analysieren**
  Fehlerbilder verdichten und Ursachen schneller eingrenzen.

AI-HEBEL

SENIOR-VERANTWORTUNG

- **Architektur & Vertrauensgrenzen**
  Was darf welchem System, Nutzer oder Schlüssel vertraut werden?
- **Invarianten & Zustandslogik**
  Welche Regeln dürfen unter keinen Umständen brechen?
- **Berechtigungen & Admin-Rechte**
  Wer kann was verändern, pausieren oder freigeben?
- **Abwägungen & Release-Entscheidungen**
  Welche Lösung ist sicher, wartbar und produktseitig sinnvoll?
- **Review & Verantwortung**
  Kein AI-Output ging ungeprüft in produktionsrelevante Pfade.

AI erzeugte Geschwindigkeit. Das Engineering-System machte sie reproduzierbar.

Der Vorsprung kam aus einem Engineering-System.

Nicht aus einem einzelnen Modell: Routing, Kontext, Skills, Tools und Quality Gates wirkten in jedem kleinen Delivery-Loop zusammen.

01	**SMART PROJECT MARKDOWNS**	Versionierte Regeln für Brand, Architektur, Protokoll, Security, UX und Done.
02	**CONTEXT MANAGEMENT**	Relevantes Wissen strukturieren, filtern, komprimieren und handoff-fähig halten.
03	**SEMANTISCHES MODEL ROUTING**	Aufgabe, Risiko und Context-Budget bestimmen Modell, Agent und Toolchain.
04	**AI SKILLS & SPEZIALISIERTE AGENTS**	Wiederverwendbare Playbooks statt jedes Mal neuer Ad-hoc-Prompts.
05	**DETERMINISTISCHE TOOLING-LAYER**	Builds, Tests, Analyzer, isolierte Scopes und CI liefern harte Evidenz.
06	**HUMAN OWNERSHIP & SECURITY**	Senior Review, Audit, Freigabe und reale Kundendemos schließen den Loop.

JEDER KLEINE SLICE

Intent → Context → Route

Build → Verify → Learn

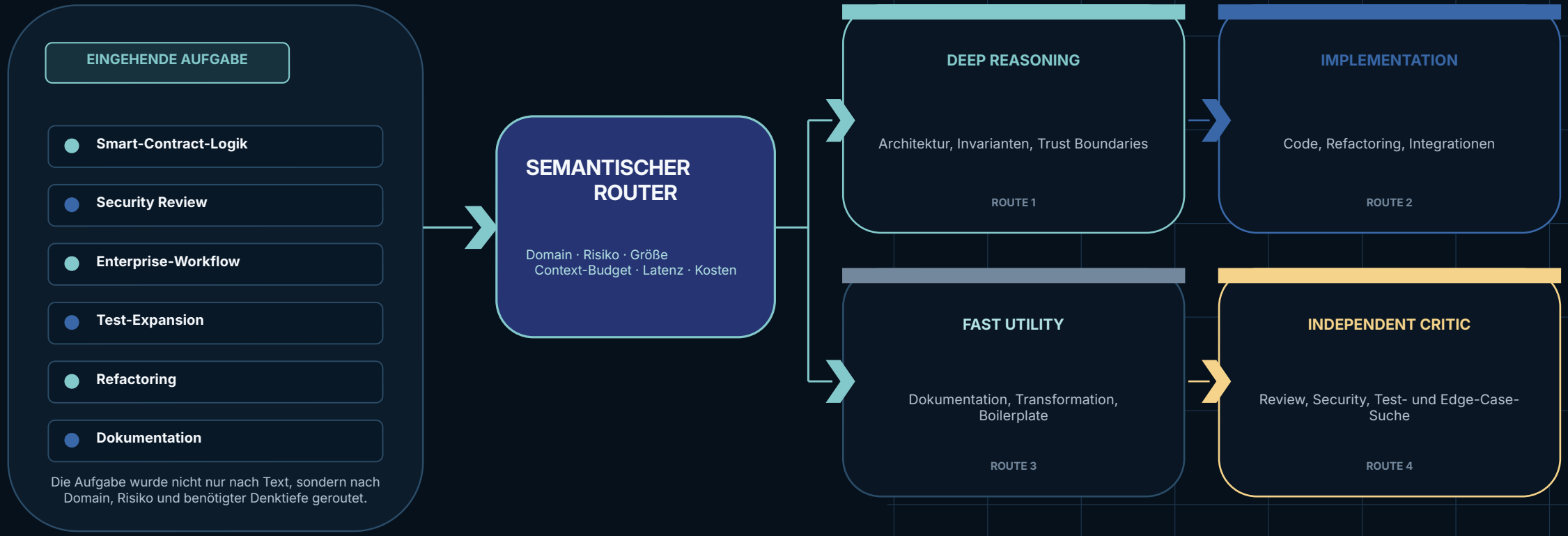
1	**INTENT**	Ziel + Done-Kriterien
2	**CONTEXT**	nur relevante Regeln & Dateien
3	**ROUTE**	passendes Modell / Skill
4	**BUILD**	isolierter, kleiner Scope
5	**VERIFY**	Tests, Review, Security
6	**LEARN**	Feedback zurück ins System

1-2 Senior Engineers · weniger Übergaben · dichteres Feedback

Wir optimierten nicht Prompt-Qualität. Wir optimierten verifizierte Lernzyklen pro Monat.

Jede Aufgabe ging zum passenden Modell.

Architektur, Smart Contracts, UX, Tests und Review verlangen unterschiedliche Stärken. Vor der Ausführung wurde die Aufgabe semantisch klassifiziert.



Kritische Pfade erhielten Fallback, Cross-Check oder einen bewusst unabhängigen Reviewer.

Kontext war Infrastruktur, nicht Prompt-Handwerk.

Agents bekamen weder das gesamte Repository noch zufällige Ausschnitte, sondern den kleinsten belastbaren Kontext für die konkrete Aufgabe.

PERSISTENT PROJECT TRUTH
Versionierte Regeln: Brand, Architektur, Protokoll, Security, UX und Done.
PROJECT MD

CODEBASE- & KNOWLEDGE-MAP
Strukturen, Beziehungen und relevante Abhängigkeiten statt blindem Vollscan.
GRAFT · GRAPHIFY

TASK CONTEXT ASSEMBLY
Nur betroffene Dateien, Entscheidungen, Rollen, Risiken und Akzeptanzkriterien.
LEANCTX

TOOL-OUTPUT COMPRESSION
Shell-, Test- und Analyzer-Ausgaben verdichten, ohne das relevante Signal zu verlieren.
RTK / RTK-AI

HANDOFF & LEARNING LEDGER
Findings, Entscheidungen und offene Punkte bleiben über Agent-Läufe hinweg anschlussfähig.
MEMORY · HANDOFF

DER CONTEXT-BUDGET-TEST

Was ein Agent sieht, bestimmt ebenso stark wie das Modell, wie gut er entscheidet.

ZU VIEL	Rauschen · Kosten · Drift
ZU WENIG	Lokale Fixes · verpasste Abhängigkeiten
KURATIERT	Fokus · Wiederholbarkeit · prüfbare Entscheidungen

Tooling änderte sich. Das Prinzip blieb: High signal in, verifizierbare Änderung out.

Der Agent sollte nicht alles sehen. Er sollte das Richtige sehen - plus die Regeln, die er nicht brechen darf.

Projektwissen wurde ausführbar.

Kurze, smarte Markdown-Dateien hielten Entscheidungen im Repository. AI Skills verwandelten diese Regeln in wiederholbare Arbeitsabläufe.

SMART PROJECT MARKDOWNS

kurz · versioniert · direkt im Repo · agentenlesbar

brand.md

Tonalität, UI-Prinzipien, visuelle Regeln

architecture.md

Grenzen, Services, Integrationen, Entscheidungen

protocol-rules.md

Rollen, Invarianten, Zustände, Upgrade-Regeln

security.md

Trust Boundaries, Secrets, Threat Model

ux-guidelines.md

States, Fehlerfälle, Enterprise-Flows

definition-of-done.md

Tests, Evidenz, Review und Freigabe

RULES → ACTION

AI SKILLS

Wiederverwendbare, projektspezifische Playbooks

01	IMPLEMENT SLICE	Scope verstehen, relevante Regeln laden, kleinsten vollständigen Diff bauen.
02	REVIEW INVARIANTS	Smart-Contract-Zustände, Rollen, Rechte und kritische Übergänge prüfen.
03	EXPAND TESTS	Edge Cases, Property-/Fuzz-Tests und Regressionen aus Findings ableiten.
04	SECURITY PREFLIGHT	Trust Boundaries, Zugriffspfade, Secrets und Integrationsrisiken prüfen.
05	UX & RELEASE EVIDENCE	Workflow-Konsistenz, Done-Kriterien, Demo- und Release-Evidenz erzeugen.

Wissen wurde nicht neu erklärt. Es wurde versioniert, wiederverwendet und bei jedem Agent-Lauf angewandt.

Agents konnten liefern. Shippen durften sie erst nach Gates.

Geschwindigkeit kam nicht aus weniger Kontrolle, sondern aus früher, automatisierter und wiederholbarer Kontrolle im selben Arbeitskontext.

NO DIRECT-TO-MAIN · NO DIRECT-TO-PROD

01 BOUNDED AGENT WORK
Isolierter Scope, Worktree und erlaubte Dateien / Tools.

02 BUILD & STATIC
Compile, Type, Lint, Dependencies und Contract-Analyzer.

03 TEST EVIDENCE
Unit, Integration, Property-/Fuzz-Tests und Edge Cases.

04 INDEPENDENT CRITIC
Zweites Modell / Agent prüft Diff, Logik und Risiken.

05 SENIOR + SECURITY
Human Review; High-Risk-Pfade mit Security Engineer.

06 DEMO + RELEASE
Kundenworkflow verifizieren, dann Merge und kontrollierter Release.

FAIL → FINDING → SAME CONTEXT → FIX → REGRESSION → RE-RUN

FAIL CLOSED — Ein Gate stoppt den Pfad.

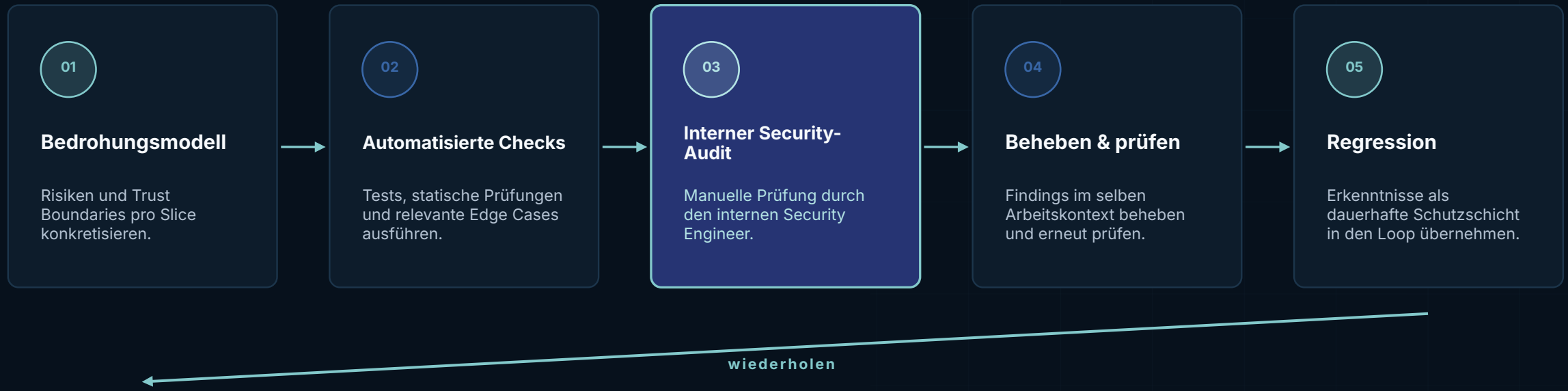
FINDING → TEST — Erkenntnisse werden dauerhaft.

WRITER ≠ ONLY REVIEWER — Unabhängige Kritik bleibt Pflicht.

Agents beschleunigten Arbeit. Gates entschieden, ob das Ergebnis belastbar genug war.

Nach einem realen Verlust war Security kein Gate am Ende.

Ein interner Security Engineer auditierte wiederkehrend während der Umsetzung. Findings wurden priorisiert, behoben und als Regressionstest dauerhaft abgesichert.



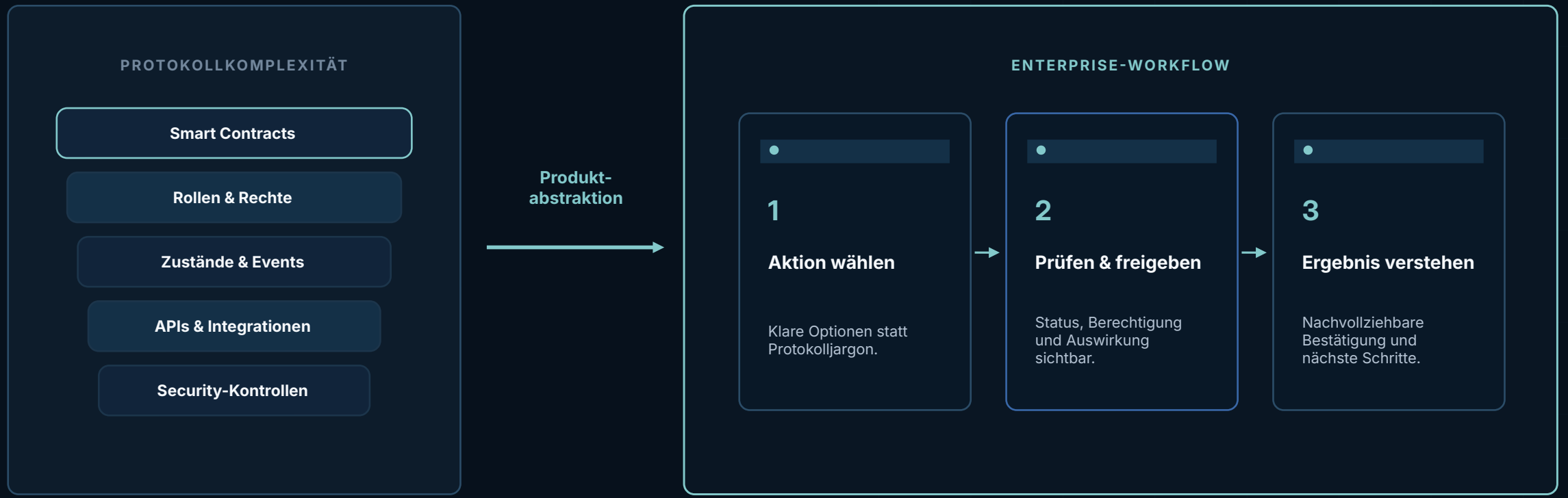
SMART-CONTRACT-SPEZIFISCHER PRÜFFOKUS

- ZUGRIFFE
- ZUSTÄNDE & INVARIANTEN
- ADMIN- & UPGRADE-RECHTE
- ASSET- & INTEGRATIONSFLÜSSE

Frühe Findings waren günstige Findings - und erhöhten die Sicherheit aller folgenden Slices.

Komplexe Protokolllogik musste sich einfach anfühlen.

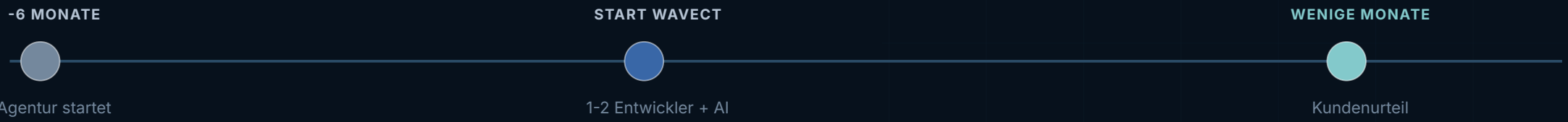
Kurze, funktionierende Demos ersetzten lange Spezifikationsketten. Der Kunde bewertete reale Workflows - nicht unsere Absichten.



FUNKTIONIERENDE DEMO → KUNDENFEEDBACK → NÄCHSTER SLICE → WIEDERHOLEN

Der 6-Monats-Vorsprung drehte sich innerhalb weniger Monate.

Laut Kunde lag Wavect danach bei Funktionsumfang, Security und UX vorn.



FUNKTIONSUMFANG

vorn

LAUT KUNDE

Relevantere und vollständigere Produktfähigkeiten im realen Projektkontext.

SECURITY

vorn

LAUT KUNDE

Nach realem Verlust: wiederkehrende Audits, Regression und klare Ownership.

UX

vorn

LAUT KUNDE

Besser verständliche und nutzbare Enterprise-Workflows trotz Protokollkomplexität.

Der Hebel war ein Engineering-System: kurze Loops, sauberes Routing, kuratierter Kontext und harte Gates.

Enterprise RWA & Smart Contract Engineering · wavect.io

Anonymisierte Case Study. Angaben zum Sicherheitsvorfall und zur relativen Leistung beruhen auf Kundenaussage; keine unabhängige forensische Verifikation oder Benchmark.