

TATORT ALTSYSTEM: KI-WATSON UND DIE SUCHE NACH DEM VERBORGENEN WISSEN

Software-Dokumentation mit KI





Ralph Engelmann, Softwareexperte

Wenn es um die Modernisierung von IT-Landschaften geht, ist ein Projekt riskanter als jedes andere: die Ablösung eines geschäftskritischen Altsystems. Es ist das Projekt, vor dem selbst erfahrene CTOs und Abteilungsleiter*innen den größten Respekt haben.

Und das zu Recht. Es ist ein Vorhaben, bei dem Budgets von mehreren hunderttausend Euro keine Seltenheit sind – und das Risiko des Scheiterns ist signifikant.

1 Einführung ins Thema und in unsere Arbeitsweise	3
2 Erklärung Vorgehen	7
3 Kontakt	10



DAS MILLIONEN-EURO-RISIKO „LEGACY-ABLÖSUNG“

Ein neues System auf der „grünen Wiese“ zu bauen, ist eine Herausforderung. Ein bestehendes Legacy-System abzulösen, ist ungleich schwerer. Warum? Weil das neue System nicht im luftleeren Raum startet.

Es tritt direkt gegen einen etablierten, ausgereiften Gegner an: das Altsystem.

Neben der offiziellen Anforderungsliste, dem Lastenheft, gibt es oft tausend unausgesprochene Anforderungen, die tief im Altsystem stecken. Diese sind nirgendwo dokumentiert – sie leben nur im Code. Branchenberichte und IT-Studien belegen, dass Legacy-Ablösungen zu den riskantesten IT-Projekten überhaupt gehören. Der Hauptgrund: Das über Jahre gewachsene Wissen über Geschäftsregeln, Sonderfälle und Kernprozesse ist nicht mehr in den Köpfen der Mitarbeitenden oder in Dokumentationen, sondern nur noch im Quellcode.

Das Benchmark-Problem

Hinzu kommt das Benchmark-Problem: Ein Altsystem mag alt sein, aber es funktioniert. Es hat keine Kinderkrankheiten mehr und bietet einen Funktionsumfang, der in tatsächlich funktionierenden Geschäftsprozessen notwendig ist.

Ein „Minimum Viable Product“ (MVP) funktioniert bei einer Ablösung nicht: Anwender*innen werden ein neues System erst akzeptieren, wenn es mindestens den Funktionsumfang des Altsystems abbildet. Jede fehlende Funktion wird als Rückschritt empfunden und gefährdet die Akzeptanz.

Diese doppelte Herausforderung – vollständige Feature-Gleichheit plus sofortige Stabilität – macht die Ablösung so riskant. Wenn das im Code verborgene Prozesswissen nicht vollständig gehoben wird, explodieren die Kosten für Nachbesserungen, oder schlimmer: Die Prozesse funktionieren nicht und das neue System scheitert komplett. Das Projekt wird zum Glücksspiel.

Umso wichtiger ist es, all das implizite Wissen aus dem Altsystem verfügbar zu machen, um so das Fundament des Neusystems zu sichern.



DER SCHATZ: WARUM DER CODE (UND NICHT DIE DOKU) DIE WAHRHEIT IST

Dieser Umstand ist aber nicht nur ein Risiko. Er ist auch eine Chance. Wir greifen dafür einen Spruch aus dem Extrem-Programming auf: „Der Code ist die Dokumentation.“ Und in diesem Code liegt ein wahrer Schatz.

Warum ist er ein „Schatz“?

- Er ist aktuell: Der Code ist das einzige Artefakt, das zu 100 Prozent den Ist-Zustand beschreibt. Es ist das, was tatsächlich jeden Tag im Unternehmen läuft.
- Er ist vollständig: Veraltete Dokumentationen sind im besten Fall unvollständig, im schlimmsten Fall irreführend. Sie beschreiben oft einen „Soll-Zustand“, der nie erreicht wurde, oder eine Funktion, die längst wieder geändert wurde.
- Er enthält die „vergessene“ Logik: Der Code ist wie ein geologisches Sediment. Jede Schicht – jeder Bugfix, jede Anpassung an eine neue Marktregel, jeder „Quick-Fix“ von vor fünf Jahren – enthält wertvolles Domänenwissen, das oft selbst die ursprünglichen Entwickler*innen vergessen haben.

Dieses Wissen wieder zugänglich zu machen, ist kein „Nice-to-have“, sondern die Grundvoraussetzung für eine erfolgreiche Ablösung. Es ist die Landkarte, in die auch ein neues System passen muss.

Der zweite Teil der Wahrheit: Der Schatz ist versteckt und gleicht eher einer Schnitzeljagd

Die Gründe für die Ablösung eines Altsystems sind vielfältig. Entweder ist es technisch derart veraltet, dass ein sicherer Betrieb nicht mehr gewährleistet ist, oder es wurde über Jahre so stark modifiziert, dass Wartung und Fehlerbehebung unwirtschaftlich geworden sind. In beiden Fällen ist der Code oft schwer zu durchdringen. Der Schatz liegt nicht offen da, er ist in einer komplexen, unübersichtlichen Höhle versteckt. Ihn zu finden, ist eine sehr langwierige Detektivarbeit. Und Tatsache ist: Das Erstellen von Dokumentation gehört zu den unbeliebtesten Aufgaben in der Softwareentwicklung. Wäre es einfach und ruhmreich, bestünde das Problem nicht – das System wäre perfekt dokumentiert.

Der Trugschluss – Warum KI allein scheitert

Die Hoffnung, ein System einfach mit einem Prompt à la „Liebe KI, dokumentiere diese Software!“ zu analysieren, hält genau bis zum ersten Versuch. Denn das Ergebnis ist, um es milde auszudrücken, ernüchternd. Wenn eine KI ungeführt eine komplexe Codebasis dokumentieren soll, ist das Ergebnis entweder extrem oberflächlich („Dieses System verwendet eine Datenbank“) und verliert sich in nutzlosen Details oder, was viel gefährlicher ist, die KI halluziniert. Sie erfindet Zusammenhänge, interpretiert Code falsch und produziert eine „Doku-Fiktion“, in der Wahrheit und Fiktion verschmelzen.

Eine KI ist ein Werkzeug. Ein unglaublich leistungsfähiges, aber ihr fehlt die Erfahrung. Einer ungeschulten Assistenzkraft würde man ja auch nicht die Leitung einer komplexen forensischen Untersuchung anvertrauen.



Unser Ansatz: Die geführte Ermittlung (Mensch & KI im Dialog)

Genau deshalb braucht es einen anderen Ansatz: einen iterativen Dialog in einem Mensch-Maschine-Team, in dem jeder seine Stärken ausspielt:

- Unser Architekt ist Sherlock Holmes: Er ist der Chefermittler. Mit jahrelanger Erfahrung in Software-Archäologie leitet er die Untersuchung. Er stellt die entscheidenden, vertiefenden Fragen, kombiniert die Fakten und trennt das Wichtige vom Unwichtigen.
- KI-Watson: Ein brillanter, unermüdlicher Assistent. Watson kann in Minuten Tausende von Akten (Code-Dateien) durchwühlen, Muster erkennen und Berichte erstellen. Aber er braucht die Führung durch Holmes, um die richtigen Schlüsse zu ziehen.
- Der Kunde (Mandant): Er beauftragt die Ermittlung und ist unser wichtigster Partner. Er legt fest, was untersucht werden soll („Was ist mit dem 'Fall Bestellprozess'?“ oder „Prüfen Sie die Anbindung an unseren Zahlungsdienstleister!“). Durch seine „Zeugenaussage“ – sein Fachwissen als Fachexperte oder Product Owner – liefert er entscheidende Hinweise.

Unsere Ermittlung ist ein iterativer Dialog!

Watson liefert einen ersten Bericht („Es gibt ein 'Repository Pattern'“). Holmes (unser Softwarearchitekt) zieht daraus seine Schlüsse und fordert weitere Nachforschung. Er instruiert Watson: „Zeige mir alle Implementierungen im Code“ oder „Beschreibe genau, wie dieses Pattern dem Standard in Spring Data JPA folgt.“ Unser Softwarearchitekt bewertet alle Informationen, die die KI ausgräbt, und nutzt dabei seine jahrelange Erfahrung, um an den relevanten Stellen nachzubohren oder stutzig zu werden, wenn die KI Antworten liefert, die halluziniert sein könnten. Der Mandant ist ebenfalls Teil des Dialogs, indem er die Detailfragen einbringt: Watson liefert eine Beschreibung der gefundenen Prozesse, und der Mandant fragt: „Für den 'Bestellprozess': Was genau passiert, wenn der schiefgeht? Welche Ausnahmen sind definiert?“

Durch diese Priorisierung und unser kritisches Nachfragen graben wir uns iterativ vom Offensichtlichen zum verborgenen Detail vor. Der Vorteil: Durch diese enge Zusammenarbeit können wir auf spezifische Fragestellungen eingehen, die sich erst während der Analyse ergeben. Der Kunde beobachtet nicht nur, er ist Teil der Ermittlung.

Spezielle Instruktionen und ein klares Vorgehen stellen dabei sicher, dass die erstellte Dokumentation so nah wie möglich an die Realität herankommt.



DIE „AKTE“: WAS SIE AM ENDE DER ERMITTLUNG IN DER HAND HALTEN

Am Ende dieses Prozesses steht kein lebloses Dokument, das gießkannenartig alle Themen anspricht, sondern eine „Akte“ voller Fakten, die exakt die Fragen der Ermittlung beantwortet.

Wir liefern klare Antworten auf die dringendsten Fragen:

- Architektur: „Wie ist das System wirklich aufgebaut? Nicht, wie es geplant war, sondern wie es heute funktioniert.“
- Domänenwissen: „Was sind die zentralen Geschäfts-Entitäten (z.B. 'Kunde', 'Vertrag') und welche Regeln sind im Code vergraben?“
- Abhängigkeiten: „Welche externen Systeme (z.B. Zahlungsanbieter, ERP) werden von welchen Funktionen aufgerufen?“
- Prozesse: „Wie funktioniert der 'Bestellprozess' wirklich? Wir zeichnen ihn anhand der Veränderungen der Daten exakt nach.“
- Geschäftsregeln: „Welche Ausnahmen, Sonderfälle und 'Wenn-Dann'-Bedingungen gibt es, die über Jahre implementiert wurden?“
- Verborgene Abläufe: „Gibt es versteckte Prozesse, 'Ecken und Kanten' oder nächtliche Läufe ('Cronjobs'), die im Altsystem existieren, aber nirgends dokumentiert sind?“

[Der „Code Checkup“: Starten wir die Ermittlung](#)

Schritt 1: Der „Code Checkup“ Workshop.

Jeder Fall ist anders. Bevor wir starten, prüfen wir in einem kurzen „Code Checkup“ Workshop gemeinsam: Passt unser Ansatz zu Ihrem Code? Was sind Ihre dringendsten Fragen (die „Leitfragen“ der Ermittlung)? Entscheidend: Wir klären mit Ihnen zusammen Informationssicherheits-Themen.

Schritt 2: Die gemeinsame Analyse-Phase.

Unsere Software-Architekt*innen und Ihre Domänen-Experten arbeiten iterativ Hand in Hand. Wir werten die Hinweise der KI aus, validieren sie gemeinsam mit Ihnen und graben uns tief in die verborgene Logik Ihres Systems.

Schritt 3: Der Wissenstransfer (Die 'Akte').

Am Ende erhalten Sie nicht nur die aufbereiteten „Ermittlungsakten“ (Architektur, Prozesse sowie die Abhängigkeiten), sondern ein tiefes Verständnis für Ihr eigenes System. Wir machen Ihr Team (Architekt*innen, Product Owner, Entwickler*innen) wieder handlungsfähig und sichern die Investition in Ihr Neusystem.

[Lassen Sie Ihr Altsystem nicht zum ungelösten „Cold Case“ werden, der Ihr Neusystem gefährdet. Der Code ist die Wahrheit, und wir haben die Werkzeuge und die Methodik, diese Wahrheit zu entschlüsseln. Kontaktieren Sie uns für einen unverbindlichen „Code Checkup“ und lassen Sie uns gemeinsam herausfinden, welche Schätze in Ihrem Code verborgen liegen.](#)



DAS VORGEHEN

KI-Agenten dienen in diesem Ansatz nicht als autonome Autoren, sondern als schnelle Recherche-Assistenten.

KI - Coding Agent Claude Code:

- Kann Code-Dateien analysieren, durchsuchen, mit RegEx finden
- Claude Code mit Opus bzw. Sonnet 4.5
- DevContainer wegen
`claude --dangerously-skip-permissions.`
- Git für Versionierung der Dokumentation
- Visual Studio Code als Editor
- Trick: GitHub Copilot zum schneller schreiben

Vorbereitung: Code bereinigen, Widersprüche beseitigen

Entfernen von allem, was nicht zuträglich ist:

- Tests
- Toter Code
- Für den Ansatz spielt es keine Rolle, ob der Code kompiliert oder ausführbar ist.
- Dokumentation, (nur wenn sie zur Version passt!)
- veraltetes
- nicht-implementierten Features
- nach Markdown konvertiert

Der initiale Prompt

- Claude
`/init`

```
>Analysiere die Anwendung und erstelle eine Architekturdokumentation in ARCHITEKTUR-DOKU.md -- ultrathink
```

- ohne weitere Details
- Dauert 10 Minuten
- Entfernen von Kapiteln zu Verbesserungspotentialen

Systemprompt

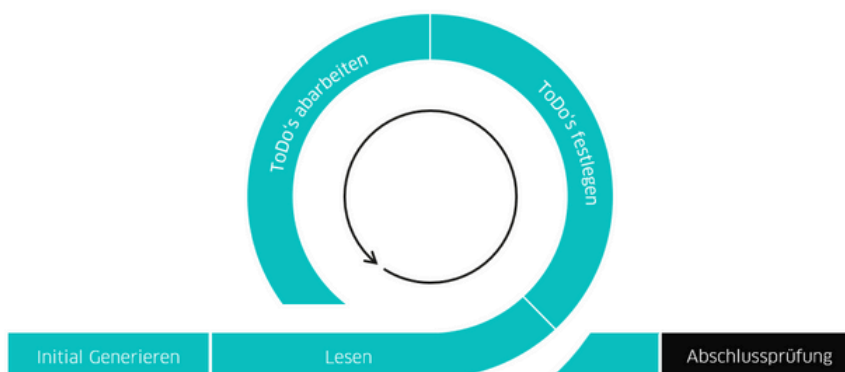
Der Code ist immer die Wahrheit, nicht die Dokumentation!

- Rolle: Spezialist für Softwarearchitektur-Dokumentation und Softwarearchäologe
- Bei Unterschieden zwischen Code und Doku vertraust du dem Code und fixt die Dokumentation!
- Bei Diskrepanzen in der Doku konsultierst du den Code und fixt die Dokumentation!
- Immer wenn du die Dokumentation korrigierst, prüfst du, ob auch andere Teile korrigiert werden müssen!
- Wenn dir in der Dokumentation etwas seltsam, unlogisch, inkonsistent oder ungenau erscheint, untersuche den Code, um das Problem zu klären, und aktualisiere die Dokumentation entsprechend!

Der Developer bestimmt, was ihn interessiert

```
//TODO: Beschreibe <das> genauer
//TODO: meinst du mit <>?
//TODO: Wo steht <das> im Code?
//TODO: Welche Implementationen gibt es noch?
//TODO: Vervollständige die Liste
//TODO: Welche Cron Jobs gibt es in der Anwendung?
```

>Arbeite die TODOs in ARCHITEKTUR-DOKU.md ab -- ultrathink





Nachverfolgen

Abgleich: TODO vs. Antwort

- //TODO: Prüfe ob das im Code tatsächlich vorkommt!
- //TODO: Ergänze Referenzen auf die Vorkommen im Code
- /clear
verwenden, um Halluzinationen zu beenden
- Selber prüfen
- //TODO: >Überprüfe ob alle Angaben in ARCHITEKTUR.md mit dem Code übereinstimmen, wenn nicht, korrigiere die Dokumentation. -- ultrathink

```
316 ### Datenbank-Features
322 2. Soft Delete:
323   - DeleteDate statt physischem Löschen
324   - Alle Queries filtern DeleteDate IS NULL
325
326 //TODO: ist das durch ein Hibernate-Feature gelöst
327 oder manuell in jeder Query?
```

```
316 ### Datenbank-Features
322 2. Soft Delete:
323   - DeleteDate statt physischem Löschen
324   - Alle Queries filtern DeleteDate IS NULL
325
326+ Implementierung: Das Soft Delete-Pattern wird
327+ manuell auf der DAO-Ebene implementiert, nicht
328+ durch ein automatisches NHibernate-Feature:
329+
330+ Lösch-Operation: Die Delete()-Methode in
331+ den DAOs ruft entity.MarkAsDeleted() auf, was das
332+ DeleteDate-Feld auf den aktuellen Zeitpunkt
333+ setzt, gefolgt von Save().
334+
335+ Query-Filterung: Jede Query muss manuell
336+ die Bedingung Restrictions.IsNull("deleteDate")
337+ hinzufügen.
338+
339+ Alle FindBy*-Methoden enthalten diese
340+ Filterung
```

Diagramme

- Diagramme lassen sich von Claude im Mermaid Format erzeugen.
- Mermaid ist ein Textformat für Diagramme.
- Für viele Markdown-Viewer gibt es Plug-ins, um diese Diagramme darzustellen.

Verschiedene Dokumentationen

Kontext reduzieren - Übersicht schaffen

- Getrennten Dokumente, um den Kontext kleiner zu halten

Architektur

Domainmodell

- z.B. über Entitäten oder Datenbankschema

Funktionen

- z.B. über Controller Endpunkte und Cron Jobs

Prozesse

- z.B. über Status-Flags, Status-Tabellen oder Enums



SPRECHEN SIE MICH AN!



Bei queo valueTec betrachten wir die intensive Auseinandersetzung mit KI nicht lediglich als eine vorübergehende Phase, sondern als einen festen Bestandteil unserer Strategie. Wir sehen Künstliche Intelligenz als ein wichtiges Werkzeug, das die Art und Weise, wie Software entwickelt wird, erheblich beeinflussen kann und wird.

Ralph Engelmann

Head of Java und
Geschäftsbereichsleitung
queo valueTec

✉ r.engelmann@queo-group.com

☎ +49 351 - 213 038 0

queo-valuetec.com



Bildnachweise:
Titel: Schubladen im Archiv
DegImages via Canva
Seite 2, 8, 9, 10: queo