VOM LEGACY-CHAOS
ZU KLARER SYSTEMARCHITEKTUR

Dein Praxis-Leitfaden für erfolgreiche Modernisierungsprojekte

openDEVS

Einleitung

Ein Feature, das eigentlich schon letzte Woche live gehen sollte, ist immer noch nicht produktiv. Beim letzten Deployment ist ein unerwarteter Fehler aufgetreten. Niemand hatte auf dem Schirm, dass eine alte Schnittstelle zu einem anderen System betroffen sein könnte. Jetzt analysiert das Team Logs, prüft Abhängigkeiten und versucht herauszufinden, warum eine scheinbar kleine Änderung plötzlich mehrere Prozesse beeinflusst.

Wenn du Softwareprojekte verantwortest, kommt dir diese Situation wahrscheinlich bekannt vor. Viele Systeme, die heute als „Legacy" gelten, sind nicht einfach schlecht gebaut worden. Sie sind über Jahre gewachsen. Mit neuen Features, zusätzlichen Integrationen, Workarounds und immer neuen Anforderungen aus dem Business. Was einmal sinnvoll und pragmatisch war, wird mit der Zeit jedoch zunehmend komplex. Änderungen werden schwieriger, Risiken steigen und ein immer größerer Teil der Entwicklungszeit fließt in Fehlerbehebung statt in die Weiterentwicklung.

Studien zeigen, dass Entwickler in Systemen mit hoher technischer Verschuldung bis zu 42 % ihrer Arbeitszeit mit Nacharbeit und Bugfixing verbringen.[1] Analysen von Organisationen wie McKinsey, IDC oder Stripe belegen zudem, dass ineffiziente Systeme Unternehmen jährlich erhebliche Kosten verursachen, oft in Millionen- oder sogar Milliardenhöhe. Die naheliegende Schlussfolgerung lautet deshalb oft: Das System muss modernisiert werden.

Aber hier beginnen die größten Herausforderungen. Denn Legacy-Modernisierung ist deutlich mehr als ein technisches Upgrade. Alte Systeme enthalten häufig jahrelang gewachsene Geschäftslogik, implizites Wissen und komplexe Abhängigkeiten zwischen Anwendungen, Daten und Prozessen. Wenn du ein solches System modernisierst, veränderst du deshalb nicht nur Code. Du greifst in gewachsene Strukturen deines Unternehmens ein.

Genau deshalb scheitern viele Modernisierungsprojekte nicht an Technologie, sondern an fehlender Klarheit über Architektur, Risiken und Organisation. In diesem Whitepaper zeigen wir dir vier typische Gründe, warum Legacy-Modernisierungen scheitern und welche Voraussetzungen du schaffen kannst, damit dein Modernisierungsprojekt stabil, kontrolliert und nachhaltig gelingt.

[1] Tornhill, A., & Borg, M. (2022). Code Red: The Business Impact of Code Quality – A Quantitative Study of 39 Proprietary Production Codebases. Proceedings of the International Conference on Technical Debt.

GRUND 01: FEHLENDE ZIELARCHITEKTUR

Das bestehende System funktioniert so nicht mehr. Eine klare Einschätzung, wenn Änderungen zu lange dauern, technische Schulden wachsen und neue Anforderungen sich nur mit großem Aufwand umsetzen lassen. Die Entscheidung zur Modernisierung ist deshalb oft schnell getroffen. Was jedoch häufig fehlt, ist ein ebenso klares Bild davon, wie die Zielarchitektur eigentlich aussehen soll.

Hier liegt eine der häufigsten Ursachen für gescheiterte Modernisierungsprojekte begraben. Denn ohne ein definiertes Architektur-Zielbild besteht die Gefahr, dass Modernisierung vor allem technologisch gedacht wird: Das alte Framework wird ersetzt, einzelne Services werden ausgelagert oder eine Microservices-Architektur wird eingeführt, weil sie als „moderner" gilt. **Solche Entscheidungen entstehen nicht selten aus dem Wunsch heraus, „technologisch aufzuholen", statt aus einer klaren Analyse der tatsächlichen Anforderungen.**

Das Problem dabei: Technologie allein löst keine strukturellen Probleme. Wenn kein gemeinsames Verständnis über die zukünftige Systemarchitektur existiert, entwickeln sich Modernisierungsprojekte schnell zu einer Sammlung einzelner technischer Maßnahmen. Teams modernisieren einzelne Komponenten, führen neue Technologien ein oder bauen zusätzliche Services auf, oft ohne klaren Bezug zu einem übergeordneten Architekturkonzept.

Besonders kritisch wird dies, wenn Architekturentscheidungen vor allem durch technologische Trends geprägt sind. Microservices, Event-Driven Architecture oder neue Cloud-Plattformen können in vielen Fällen sinnvolle Ansätze sein, vorausgesetzt, sie werden bewusst und im Kontext der konkreten Systemanforderungen eingesetzt. Werden solche Konzepte jedoch ohne klare Architekturvision eingeführt, entstehen neue Abhängigkeiten, zusätzliche Integrationspunkte und damit langfristig neue technische Schulden.

Ein weiteres Risiko besteht in der fehlenden Roadmap für die Modernisierung selbst. Ohne ein strukturiertes Zielbild bleibt oft unklar, welche Systemteile zuerst modernisiert werden sollten, welche Komponenten langfristig bestehen bleiben und wie Übergangsphasen organisiert werden. Teams arbeiten dann parallel an verschiedenen Teilen des Systems, ohne dass klar ist, wie sich diese Veränderungen später zu einer konsistenten Gesamtarchitektur zusammenfügen.

Eine erfolgreiche Modernisierung beginnt deshalb nicht mit Technologieentscheidungen, sondern mit Architekturklarheit. Dazu gehört ein gemeinsames Verständnis darüber, welche Rolle das System künftig im Unternehmen spielen soll, welche Anforderungen aus Business-Sicht relevant sind und welche Architekturprinzipien daraus abgeleitet werden.

TECHNOLOGIE-GETRIEBEN

- *Reaktive Entscheidungen*
- *Fokus auf Tools und Trends*
- *Einzelne Refactorings ohne Blick aufs Gesamtbild*
- *Einführen von Micoservices ohne Bedarfserhebung*

NEUE KOMPLEXITÄT

VS

ARCHITEKTUR-GETRIEBEN

- *Start mit Zielbild und Systemrolle*
- *Bedürfnisse abgeleitet aus Business-Anforderungen*
- *klare Domänen und Schnittstellen*
- *geplante Transition und klare Roadmap*

SKALIERBARE STRUKTUR

GRUND 02: FEHLENDE DOKUMENTATION

Viele Legacy-Systeme sind über Jahre oder sogar Jahrzehnte gewachsen. Neue Funktionen wurden ergänzt, Integrationen hinzugefügt und bestehende Prozesse immer wieder angepasst. In dieser Entwicklung entsteht häufig ein System, das zwar funktioniert, dessen Struktur und Logik jedoch nur noch teilweise dokumentiert sind.

Ein großer Teil des Wissens über das System existiert dann nicht in Dokumentationen, sondern im Kopf einzelner Entwickler:innen. Geschäftslogik steckt tief im Code, historische Entscheidungen sind kaum nachvollziehbar und wichtige Zusammenhänge werden nur informell im Team weitergegeben.

Gerade in stabil laufenden Systemen fällt dieses Problem lange kaum auf. Solange dieselben Personen an dem System arbeiten, lassen sich viele Fragen schnell klären. Kritisch wird es jedoch, sobald eine Modernisierung geplant wird oder sich Teamstrukturen verändern.

LIES NOCH MEHR AUF UNSEREM BLOG!

Fehlende Dokumentation erschwert zunächst die realistische Einschätzung des Modernisierungsaufwands. Ohne klares Verständnis über Datenflüsse, Abhängigkeiten und Geschäftslogik lassen sich Komplexität und Risiken nur schwer abschätzen. Projekte werden dann auf Basis unvollständiger Informationen geplant mit entsprechend hohen Unsicherheiten.

Hinzu kommt, dass viele Legacy-Systeme im Laufe der Zeit informelle Workarounds und Sonderlogiken entwickelt haben. Bestimmte Funktionen existieren vielleicht nicht aus technischer Notwendigkeit, sondern weil sie einmal ein spezielles Geschäftsproblem gelöst haben. Wenn solche Zusammenhänge nicht dokumentiert sind, besteht die Gefahr, dass sie bei der Modernisierung übersehen werden.

Die Folgen zeigen sich oft erst spät im Projekt. Funktionen verhalten sich anders als erwartet, Integrationen brechen oder wichtige Geschäftsprozesse funktionieren nach der Umstellung nicht mehr zuverlässig. Solche Probleme führen nicht nur zu zusätzlichen Entwicklungsaufwänden, sondern können auch zu unerwarteten Produktionsproblemen und Projektverzögerungen führen.

Deshalb ist Transparenz über bestehende Systeme eine zentrale Voraussetzung für erfolgreiche Modernisierung. Dazu gehört nicht nur klassische Dokumentation, sondern vor allem ein strukturiertes Verständnis der Geschäftslogik, Abhängigkeiten und Systemgrenzen.

Je besser dieses Wissen sichtbar und im Team verfügbar ist, desto realistischer lassen sich Risiken einschätzen und Modernisierungsschritte planen. Fehlende Dokumentation ist daher kein rein organisatorisches Detail. Sie ist ein entscheidender Faktor dafür, ob ein Modernisierungsprojekt kontrolliert verlaufen kann oder von unerwarteten Problemen ausgebremst wird.

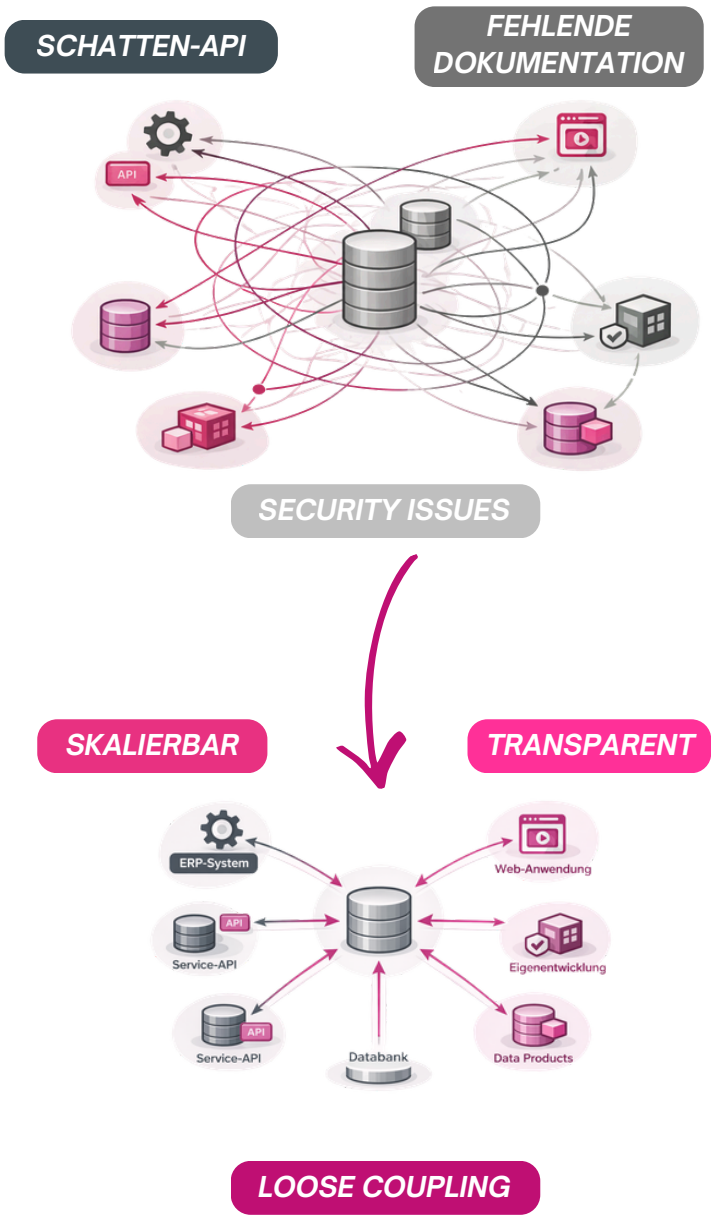
GRUND 03: UNBEKANNTE SCHNITTSTELLEN UND ABHÄNGIGKEITEN

Mit der Zeit ist in Legacy-Systemen ein dichtes Netz aus Integrationen entstanden. Anwendungen greifen auf gemeinsame Datenbanken zu, Systeme kommunizieren über interne APIs, und zusätzliche Schnittstellen wurden oft pragmatisch ergänzt, um neue Anforderungen schnell umzusetzen. Viele dieser Verbindungen sind allerdings nie sauber dokumentiert oder zentral gesteuert worden.

So entstehen sogenannte Schatten-APIs, direkte Datenbankzugriffe oder Integrationen, die außerhalb definierter Architekturprinzipien aufgebaut wurden. Solange das System stabil läuft, bleiben diese Abhängigkeiten oft unbemerkt. Erst wenn Änderungen vorgenommen werden, zeigt sich, wie stark einzelne Komponenten tatsächlich miteinander verknüpft sind. Für Modernisierungsprojekte stellt das ein erhebliches Risiko dar.

Wird eine Komponente verändert oder ersetzt, können unerwartet andere Systeme betroffen sein. Kleine Änderungen lösen dann Dominoeffekte aus, weil weitere Anwendungen auf dieselben Datenstrukturen oder Schnittstellen angewiesen sind. Neben funktionalen Problemen entstehen dadurch auch Sicherheitsrisiken.

Direkte Datenbankzugriffe und unkontrollierte Schnittstellen erschweren Governance, Monitoring und Zugriffskontrollen. Eine erfolgreiche Modernisierung setzt daher voraus, diese Abhängigkeiten sichtbar zu machen. Erst wenn klar ist, wie Systeme tatsächlich miteinander verbunden sind, lassen sich Änderungen kontrolliert planen und Risiken gezielt reduzieren.



FOLGE UNS AUF LINKEDIN FÜR NOCH MEHR BRANCHENINFOS UND BEST PRACTICES!

SCAN HERE

GRUND 04: PERSONELLE ABHÄNGIGKEITEN

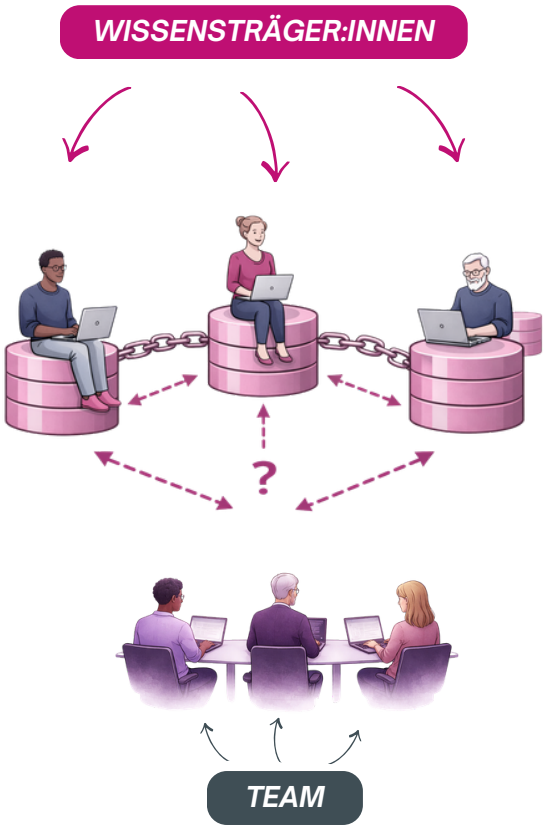
Viele Legacy-Systeme werden über Jahre von denselben Entwicklerinnen und Entwicklern betreut. Sie kennen die Architektur, wissen, warum bestimmte Entscheidungen getroffen wurden und verstehen die oft komplexe Geschäftslogik im Detail. Dieses Wissen ist für den stabilen Betrieb des Systems enorm wertvoll, wird aber häufig nicht systematisch dokumentiert oder im Team geteilt.

So entstehen Wissens-Silos. Einzelne Personen werden zu zentralen Wissensträgern für bestimmte Systembereiche, Schnittstellen oder Geschäftslogiken. Solange diese Personen im Projekt verfügbar sind, funktioniert die Zusammenarbeit meist gut. Kritisch wird es jedoch, wenn sich Teamstrukturen verändern. Verlässt eine Schlüsselperson das Unternehmen oder wechselt in ein anderes Projekt, geht häufig ein großer Teil dieses Wissens verloren.

Neue Teammitglieder müssen sich mühsam in bestehende Strukturen einarbeiten, während wichtige Zusammenhänge zunächst unklar bleiben. Das erhöht das Risiko von Fehlentscheidungen und erschwert die Planung von Modernisierungsschritten erheblich.

Auch im laufenden Projekt können personelle Abhängigkeiten problematisch sein. Wenn nur wenige Personen bestimmte Systembereiche wirklich verstehen, entstehen schnell blockierte Entscheidungswege. Teams müssen auf Rückmeldungen warten, Änderungen können nicht eigenständig umgesetzt werden und wichtige Architekturentscheidungen hängen an einzelnen Expertinnen oder Experten.

Für Modernisierungsprojekte bedeutet das ein erhebliches Risiko. Wissen muss deshalb bewusst im Team verteilt werden, etwa durch gemeinsame Architekturarbeit, Pair Programming oder strukturierte Dokumentation zentraler Systembereiche. Erst wenn kritisches Systemwissen nicht mehr an einzelne Personen gebunden ist, lassen sich Modernisierungsprojekte stabil planen und langfristig erfolgreich umsetzen.



RISIKEN

Verlust von Schlüsselwissen

Person verlässt Team, das Wissen geht mit

Mühsame Einarbeitung

Neue Mitglieder sind lange auf Unterstützung angewiesen

LÖSUNGEN

Wissenstransfer & Dokumentation

Systematisch Wissen teilen und dokumentieren

Gemeinsame Architekturarbeit

Gemeinsam Architektur designen und Entscheidungen besprechen

STEP-BY-STEP DURCHS MODERNISIERUNGSPROJEKT

Wenn Legacy-Systeme modernisiert werden sollen, ist die Versuchung groß, möglichst schnell mit der technischen Umsetzung zu beginnen. In der Praxis zeigt sich jedoch immer wieder: Ohne strukturiertes Vorgehen entstehen neue Risiken, zusätzliche Komplexität oder sogar neue technische Schulden.

Erfolgreiche Modernisierung folgt deshalb keinem Big-Bang-Ansatz, sondern einem klar strukturierten, schrittweisen Vorgehen. **Bevor erste Komponenten verändert werden, braucht es Transparenz über das bestehende System, ein gemeinsames Zielbild für die zukünftige Architektur und eine realistische Migrationsstrategie.** Gleichzeitig müssen Teams sicherstellen, dass Wissen über das System langfristig verfügbar bleibt und nicht erneut in einzelnen Silos verschwindet.

Der folgende Ablauf zeigt ein mögliches Vorgehensmodell für Modernisierungsprojekte. Die einzelnen Schritte bauen aufeinander auf und helfen dabei, Risiken frühzeitig sichtbar zu machen, Architekturentscheidungen fundiert zu treffen und die Transformation der Systemlandschaft kontrolliert umzusetzen.

STEP 01: TRANSPARENZ SCHAFFEN UND DAS SYSTEM WIRKLICH VERSTEHEN

Der erste Schritt in jedem Modernisierungsprojekt besteht darin, Transparenz über das bestehende System zu schaffen. Teams kennen zwar einzelne Teile der Anwendung sehr gut, doch ein vollständiges Verständnis der gesamten Systemlandschaft fehlt häufig. Über Jahre gewachsene Software enthält komplexe Abhängigkeiten, implizite Geschäftslogik und Integrationen, die nur teilweise dokumentiert sind. Ohne ein klares Bild dieser Strukturen wird jede Modernisierung zum Risiko.

Bevor also über neue Technologien oder Architekturansätze entschieden wird, muss zunächst genau bekannt sein wie das bestehende System funktioniert. **Eine systematische Analyse des Codes und der Architektur bildet dafür die Grundlage.** Dabei geht es nicht nur darum, einzelne Komponenten zu betrachten, sondern die Gesamtstruktur des Systems zu verstehen.

- **Welche Module existieren?**
- **Wie sind sie miteinander verbunden?**
- **Wo befinden sich zentrale Geschäftslogiken?**
- **Welche Teile des Systems sind besonders komplex oder kritisch für den Betrieb?**

Gerade bei großen Legacy-Systemen ist diese Analyse eine anspruchsvolle Aufgabe. Häufig wurden Anwendungen über viele Jahre von unterschiedlichen Teams weiterentwickelt. Entscheidungen wurden unter Zeitdruck getroffen, Integrationen kurzfristig umgesetzt und Workarounds eingeführt, um akute Probleme zu lösen. Das Ergebnis ist ein System, dessen tatsächliche Struktur oft deutlich komplexer ist, als es auf den ersten Blick scheint.

Deshalb ist es sinnvoll, diese Analyse gezielt durch erfahrene Senior Developer mit Architekturkompetenz durchführen zu lassen. Sie bringen nicht nur technisches Know-how mit, sondern auch die Erfahrung aus anderen Modernisierungsprojekten. Dadurch können sie typische Problemstellen schneller erkennen, etwa stark gekoppelte Komponenten, problematische Datenstrukturen oder besonders risikoreiche Integrationen.

Ziel ist es, ein realistisches Bild über den aktuellen Zustand der Software zu erhalten. Dazu gehört auch die Identifikation von Bereichen mit besonders hoher technischer Verschuldung. Komplexe Module, unübersichtliche Datenflüsse oder kritische Abhängigkeiten werden sichtbar gemacht und priorisiert.

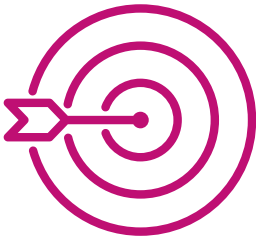
Eng damit verbunden ist ein **systematisches Risiko-Mapping**. Nicht alle Teile eines Systems sind gleichermaßen kritisch für das Geschäft. Einige Funktionen können relativ problemlos verändert werden, während andere direkt geschäftskritische Prozesse beeinflussen. Wenn diese Unterschiede frühzeitig erkannt werden, lassen sich Modernisierungsschritte deutlich besser planen. Ein strukturiertes Risiko-Mapping beantwortet diese Fragen:

- **Welche Komponenten sind besonders geschäftskritisch?**
- **Wo bestehen hohe technische Risiken?**
- **Welche Teile des Systems eignen sich für erste Modernisierungsschritte?**
- **Wo müssen Veränderungen besonders vorsichtig umgesetzt werden?**

Parallel zur Analyse sollte auch mit dem Aufbau einer systematischen Dokumentation begonnen werden. **Viele Legacy-Systeme verfügen nur über fragmentarische oder veraltete Dokumentationen.** Moderne Architekturdiagramme, nachvollziehbare Beschreibungen zentraler Geschäftslogiken und transparente Datenflüsse schaffen jedoch eine wichtige Grundlage für alle weiteren Schritte. Dabei geht es nicht darum, jedes Detail des bestehenden Systems vollständig zu dokumentieren. Entscheidend ist vielmehr, ein gemeinsames Verständnis über die wichtigsten Strukturen und Abhängigkeiten zu schaffen. Eine gute Dokumentation unterstützt Teams dabei, Architekturentscheidungen besser nachzuvollziehen und reduziert langfristig das Risiko neuer Wissens-Silos.

Der größte Mehrwert dieser Phase liegt darin, Unsicherheiten sichtbar zu machen. Viele Risiken werden erst durch eine strukturierte Analyse erkennbar. Abhängigkeiten zwischen Systemen, versteckte Geschäftslogiken oder technische Schwachstellen lassen sich frühzeitig identifizieren, bevor sie im Verlauf der Modernisierung zu Problemen führen. Transparenz ist damit keine rein technische Vorarbeit, sondern die Grundlage für fundierte Entscheidungen im gesamten Modernisierungsprojekt.

STEP 02: ZIELBILD DEFINIEREN UND EINE KLARE ARCHITEKTURVISION ENTWICKELN



Sobald Transparenz über das bestehende System geschaffen wurde, folgt der nächste entscheidende Schritt: die Definition eines klaren Zielbilds für die zukünftige Systemarchitektur. Dieser Schritt wird in vielen Modernisierungsprojekten unterschätzt. Häufig beginnen Teams bereits mit der technischen Umsetzung einzelner Maßnahmen, ohne zuvor ein gemeinsames Verständnis darüber entwickelt zu haben, wie das System in Zukunft strukturiert sein soll. Doch genau dieses Zielbild ist entscheidend. Ohne eine klare Architekturvision besteht die Gefahr, dass Modernisierung vor allem reaktiv erfolgt: einzelne Komponenten werden ersetzt, neue Technologien eingeführt oder Services ausgelagert, jedoch ohne übergreifendes Konzept. Das Ergebnis sind oft neue Abhängigkeiten und zusätzliche Komplexität statt einer nachhaltigen Verbesserung der Systemstruktur.

Ein gutes Zielbild beschreibt daher nicht nur Technologien, sondern vor allem die grundlegenden Architekturprinzipien des zukünftigen Systems. Dazu gehört beispielsweise die Frage, wie fachliche Domänen strukturiert sind, welche Systemgrenzen existieren und wie Datenflüsse zwischen Komponenten organisiert werden sollen. Ebenso wichtig ist die Definition von Architekturprinzipien, etwa in Bezug auf Modularität, Integrationsmechanismen oder Governance für neue Schnittstellen.

Gerade bei komplexen Legacy-Systemen kann es hilfreich sein, diesen Prozess bewusst strukturiert zu gestalten. Workshops zur gemeinsamen Analyse von Geschäftsprozessen und Systemverhalten helfen Teams dabei, ein gemeinsames Verständnis über die fachlichen Zusammenhänge zu entwickeln. Methoden wie Event Modeling ermöglichen es beispielsweise, Geschäftsereignisse, Prozesse und Systeminteraktionen sichtbar zu machen und daraus geeignete Systemgrenzen abzuleiten. Solche Workshops bringen Fachbereiche, Entwicklerinnen und Entwickler sowie Architektinnen und Architekten zusammen und schaffen eine gemeinsame Grundlage für spätere Architekturentscheidungen.

Auf Basis dieses Verständnisses lässt sich eine Architekturvision entwickeln. Sie beschreibt, welche Rolle das System künftig im Unternehmen spielen soll, welche Komponenten langfristig bestehen bleiben und welche Teile schrittweise ersetzt oder neu aufgebaut werden sollen. Wichtig ist dabei, realistische Zielstrukturen zu definieren, die sowohl technische als auch organisatorische Rahmenbedingungen berücksichtigen.

Eng verbunden mit der Architekturvision ist die Migrationsstrategie. Selbst wenn das Zielbild klar ist, stellt sich immer noch die Frage, wie der Weg dorthin konkret aussehen soll. In den meisten Fällen ist ein vollständiger Neubau eines Systems mit erheblichen Risiken verbunden. Geschäftskritische Systeme können oft nicht einfach abgeschaltet und ersetzt werden.

Deshalb setzen viele erfolgreiche Modernisierungsprojekte auf evolutionäre Migrationsstrategien. **Ein bekanntes Beispiel ist das Strangler Pattern.** Dabei wird ein bestehendes System schrittweise durch neue Komponenten ergänzt und ersetzt. Neue Funktionalitäten entstehen zunächst außerhalb des Monolithen, während bestehende Teile nach und nach abgelöst werden. Auf diese Weise kann das System kontinuierlich modernisiert werden, ohne den laufenden Betrieb zu gefährden.

Eine weitere Möglichkeit besteht darin, das bestehende System zunächst stärker zu modularisieren, bevor einzelne Komponenten ersetzt werden. Klare Modulgrenzen erleichtern es später, einzelne Teile unabhängig voneinander zu modernisieren und reduzieren gleichzeitig die Komplexität der Systemlandschaft. Neben technischen Entscheidungen spielt auch die Business-Priorisierung eine zentrale Rolle.

Nicht alle Systemteile müssen gleichzeitig modernisiert werden. In vielen Fällen ist es sinnvoll, zunächst solche Bereiche zu betrachten, die besonders häufig verändert werden, hohe Betriebskosten verursachen oder strategisch wichtige Funktionen unterstützen.

Die Kombination aus Architekturvision, Migrationsstrategie und Business-Priorisierung schafft eine belastbare Grundlage für die weiteren Schritte des Modernisierungsprojekts. Teams wissen, wohin sich das System entwickeln soll, welche Maßnahmen sinnvoll sind und in welcher Reihenfolge Veränderungen erfolgen sollten. Ein klar definiertes Zielbild sorgt damit für Orientierung im gesamten Projekt. Es verhindert technologische Einzelentscheidungen ohne langfristige Perspektive und stellt sicher, dass alle Modernisierungsschritte auf ein gemeinsames architektonisches Ziel einzahlen.

LIES MEHR AUF UNSEREM BLOG!

IT Leads Topics 07.11.2025

Planungssicherheit durch Partnerschaft: Softwareprojekte retten!

Viele Softwareprojekte scheitern an fehlender Planungssicherheit. Erfahre, wie du mit stabilen Teams, verlässlichen Partnerschaften und klarer Ressourcenplanung Projekterfolg langfristig sicherst.

› MEHR LESEN

STEP 03: VERÄNDERUNGEN KONTROLLIERT UMSETZEN

Nachdem Transparenz über das bestehende System geschaffen und ein klares Zielbild definiert wurde, beginnt der eigentliche Transformationsprozess: die schrittweise Modernisierung der Softwarelandschaft. In dieser Phase entscheidet sich, ob Architekturvision und Strategie tatsächlich in stabile, funktionierende Systeme übersetzt werden können.

Eine der wichtigsten Grundregeln lautet dabei: Modernisierung sollte evolutionär erfolgen, nicht revolutionär. Der Versuch, ein gewachsenes System vollständig neu zu bauen und in einem Schritt zu ersetzen (der sogenannte Big-Bang-Ansatz) ist mit erheblichen Risiken verbunden. **Gerade bei geschäftskritischen Anwendungen kann ein solcher Ansatz zu langen Projektlaufzeiten, hohen Kosten und im schlimmsten Fall zu Produk-tionsproblemen führen.**

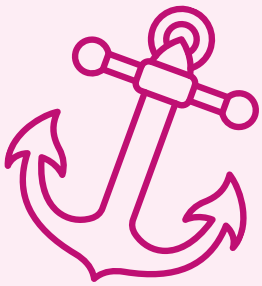
Deshalb setzen erfolgreiche Modernisierungs-projekte auf schrittweise Refactoring-Strategien. Dabei wird das bestehende System kontinuierlich verbessert, strukturiert und teilweise neu aufgebaut, ohne den laufenden Betrieb zu unterbrechen. Einzelne Komponenten werden gezielt refaktoriert, klar abgegrenzt oder in neue Services überführt. So entsteht nach und nach eine modernisierte System-struktur, während die Anwendung weiterhin produk-tiv genutzt werden kann. Refactoring bedeutet in diesem Zusammenhang mehr als nur Code zu bereinigen. Es geht darum, Strukturen gezielt zu verbessern, technische Schulden zu reduzieren und die Wartbarkeit des Systems langfristig zu erhöhen. Durch kleine, kontrollierte Veränderungen lassen sich Risiken deutlich besser steuern als durch groß angelegte Umbauten.

In vielen Projekten stellt sich an dieser Stelle die grundlegende Frage: **Refactoring oder Rewriting?** Soll das bestehende System Schritt für Schritt weiterentwickelt werden oder ist ein vollständiger Neubau die bessere Option? In der Praxis zeigt sich, dass ein vollständiges Rewriting nur selten sinnvoll ist. Zu groß ist das Risiko, dass wichtige Geschäftslogik übersehen wird oder neue Systeme erst nach langer Entwicklungszeit den gleichen Funktionsumfang erreichen wie die bestehende Lösung.

Ein evolutionärer Ansatz ermöglicht dagegen eine kontinuierliche Verbesserung der Architektur, während das System weiterhin geschäftskritische Prozesse unterstützt. Ein zentraler Erfolgsfaktor in dieser Phase ist der Parallelbetrieb von Alt- und Neusystemen. **Neue Komponenten werden schrittweise eingeführt, während bestehende Funktionen zunächst weiterlaufen.** Dadurch können Teams neue Funktionalitäten testen, Erfahrungen sammeln und mögliche Probleme frühzeitig erkennen, bevor alte Teile des Systems vollständig ersetzt werden. Dieser Parallelbetrieb stellt jedoch auch hohe Anforderungen an die technische Umsetzung. Datenkonsistenz, Synchroni-sation zwischen Systemteilen und klare Schnitt-stellen spielen eine wichtige Rolle. Nur wenn diese Aspekte sauber umgesetzt werden, lassen sich Übergangsphasen stabil gestalten.

Gleichzeitig gewinnt in dieser Phase die Qualitäts-sicherung stark an Bedeutung. Jede Veränderung am System birgt potenzielle Risiken für bestehende Funktionen. Um diese Risiken zu minimieren, müssen Tests, Continuous Integration und automatisierte Deployments professionell etabliert sein. Automatisierte Tests helfen dabei, sicher-zustellen, dass bestehende Geschäftslogik weiterhin korrekt funktioniert. Continuous-Integration-Prozesse ermöglichen schnelle Rückmeldungen über die Auswirkungen neuer Änderungen. Klar definierte Deployment-Prozesse sorgen dafür, dass neue Versionen zuverlässig in Produktion gebracht werden können.

Gerade bei komplexen Legacy-Systemen ist es sinnvoll, diese Qualitätsmechanismen im Zuge der Modernisierung gezielt auszubauen. Moderne Entwicklungspraktiken tragen nicht nur zur Stabilität der Transformation bei, sondern verbessern langfristig auch die Wartbarkeit und Entwicklungsge-schwindigkeit des Systems. Modernisierung ist damit kein einmaliges Projekt, sondern ein kontinuierlicher Transformationsprozess. Durch schrittweises Re-factoring, kontrollierten Parallelbetrieb und profes-sionelle Qualitätssicherung können Teams ihre Systeme nachhaltig weiterentwickeln, ohne den lau-fenden Betrieb zu gefährden.



STEP 04: WISSEN IM TEAM VERANKERN

Technische Modernisierung allein reicht nicht aus, um ein System langfristig stabil und wartbar zu machen. Ebenso entscheidend ist, dass das Wissen über Architektur, Geschäftslogik und Systemzusammenhänge dauerhaft im Team verankert wird. Genau hier liegt in vielen Organisationen eine der größten Herausforderungen: Kritisches Systemwissen ist oft auf wenige Personen verteilt und nur teilweise dokumentiert.

Während der Modernisierung wird dieses Risiko besonders sichtbar. Neue Teammitglieder kommen hinzu, bestehende Rollen verändern sich und Teile des Systems werden neu aufgebaut. Wenn Wissen in dieser Phase nicht bewusst geteilt und strukturiert gesichert wird, entstehen schnell neue Wissens-Silos. Das sind genau jene Abhängigkeiten, die bereits viele Legacy-Systeme geprägt haben.

Ein wichtiger Schritt besteht deshalb darin, Wissensaustausch aktiv in den Entwicklungsprozess zu integrieren. Pair Programming ist dafür eine bewährte Methode. Zwei Entwicklerinnen oder Entwickler arbeiten gemeinsam an einer Aufgabe, diskutieren Architekturentscheidungen und teilen ihr Verständnis über den Code. Dadurch verbreitet sich Wissen automatisch im Team, und kritische Systembereiche werden nicht nur von einzelnen Personen verstanden. Neben Pair Programming spielt auch strukturierte Dokumentation eine wichtige Rolle.

Innovative Softwareprojekte benötigen keine umfangreichen Dokumentationshandbücher, aber sie brauchen klar nachvollziehbare Informationen über die wichtigsten Architekturentscheidungen, Systemgrenzen und Datenflüsse. Architekturdiagramme, Entscheidungsprotokolle oder technische Übersichten helfen Teams dabei, Zusammenhänge schnell zu verstehen und neue Teammitglieder effizient einzuarbeiten.

Gerade im Kontext einer Modernisierung ist es sinnvoll, Dokumentation nicht als nachgelagerte Aufgabe zu betrachten. Vielmehr sollte sie parallel zur Entwicklung neuer Systemteile entstehen. Wenn Architekturentscheidungen direkt dokumentiert werden, bleiben ihre Hintergründe nachvollziehbar, auch Jahre später.

Ein weiterer wichtiger Aspekt ist das Team Enablement. Neue Technologien, Architekturprinzipien oder Entwicklungspraktiken müssen von den Teams verstanden und aktiv angewendet werden können.

Schulungen, gemeinsame Architektur-Workshops oder interne Wissensformate unterstützen Teams dabei, neue Kompetenzen aufzubauen und Verantwortung für moderne Systemstrukturen zu übernehmen.

Besonders wichtig ist in diesem Zusammenhang die Einführung klarer Ownership-Strukturen. In vielen Legacy-Systemen ist oft unklar, wer für bestimmte Systembereiche verantwortlich ist. Entscheidungen werden verzögert, Änderungen bleiben liegen oder werden nur zögerlich umgesetzt, weil Zuständigkeiten fehlen.

Dadurch entstehen kürzere Entscheidungswege, schnellere Verbesserungen und eine stärkere Identifikation mit dem eigenen Systembereich. Ownership bedeutet dabei nicht, Wissen exklusiv zu besitzen. Im Gegenteil: Gute Ownership-Strukturen fördern Transparenz und Zusammenarbeit. Teams kennen ihre Verantwortungsbereiche, teilen jedoch gleichzeitig ihr Wissen mit anderen.

Langfristig entsteht so eine Organisation, in der Wissen nicht mehr an einzelne Personen gebunden ist, sondern im Team und in den Arbeitsprozessen verankert wird. Für Modernisierungsprojekte ist diese Wissenssicherung ein entscheidender Erfolgsfaktor.

Sie stellt sicher, dass die neu geschaffenen Strukturen dauerhaft verstanden, weiterentwickelt und gepflegt werden können. Ohne diesen Schritt besteht die Gefahr, dass auch ein modernisiertes System mit der Zeit wieder dieselben Probleme entwickelt wie sein Vorgänger.

Erfolgreiche Modernisierung endet daher nicht mit der technischen Transformation eines Systems. Sie endet erst dann, wenn Wissen, Verantwortung und Kompetenz nachhaltig im Team verankert sind.

Fazit

Ein neues System ist nicht automatisch ein besseres System. Es ist zunächst einfach nur neu. Wird ein bestehendes System ersetzt, ohne zuvor Architektur, Prozesse und Wissen wirklich zu verstehen, entstehen schnell neue Probleme an anderer Stelle. **Ohne ein klares Zielbild fehlt die Orientierung für Architekturentscheidungen. Technologien werden eingeführt, ohne dass sie in eine langfristige Systemstruktur eingebettet sind. Die Folge sind neue Abhängigkeiten, fragmentierte Systemlandschaften und zusätzliche technische Schulden.**

Ebenso kritisch ist fehlende Transparenz über bestehende Schnittstellen und Systemzusammenhänge. Werden versteckte Integrationen oder Abhängigkeiten übersehen, können bereits kleine Änderungen unerwartete Auswirkungen auf andere Systemteile haben. Solche Dominoeffekte gehören zu den häufigsten Risiken in Modernisierungsprojekten.

Nicht zuletzt spielt auch der Umgang mit Wissen eine entscheidende Rolle. Wenn Geschäftslogik nur im Code oder im Kopf einzelner Entwicklerinnen und Entwickler existiert, geht dieses Wissen im Verlauf eines Transformationsprojekts leicht verloren. Erfolgreiche Modernisierung bedeutet daher nicht, ein System einfach zu ersetzen. Sie bedeutet, bestehende Strukturen schrittweise und kontrolliert weiterzuentwickeln, mit einer klaren Architekturvision, einer realistischen Migrationsstrategie und stabilen Ownership-Strukturen im Team. Nur so entsteht langfristig eine Systemlandschaft, die sowohl technisch stabil als auch organisatorisch tragfähig ist.

Modernisierung geplant? Dann starte mit Klarheit statt mit Code.

Legacy-Modernisierung gehört zu den komplexesten Vorhaben in der Softwareentwicklung. Ohne Transparenz über Architektur, Abhängigkeiten und Risiken wird aus einem Modernisierungsprojekt schnell ein kostspieliges Experiment. Erfahrene Senior Devs analysieren bestehende Systeme strukturiert, machen Risiken sichtbar und entwickeln eine belastbare Modernisierungsstrategie. Gemeinsam mit deinem Team schaffen wir Transparenz über bestehende Systeme, entwickeln eine klare Architekturvision und begleiten die schrittweise Modernisierung deiner Softwarelandschaft.

Du planst ein Modernisierungsprojekt und brauchst Hochkaräter für dein Team? Wir staffen dein Team mit erfahrenen Senior Devs!

+43 677 629 084 09

www.opendevs.net

... oder buche direkt einen Videocall per QR-Code

ROBERT GITTENBERGER

GESCHÄFTSFÜHRER