



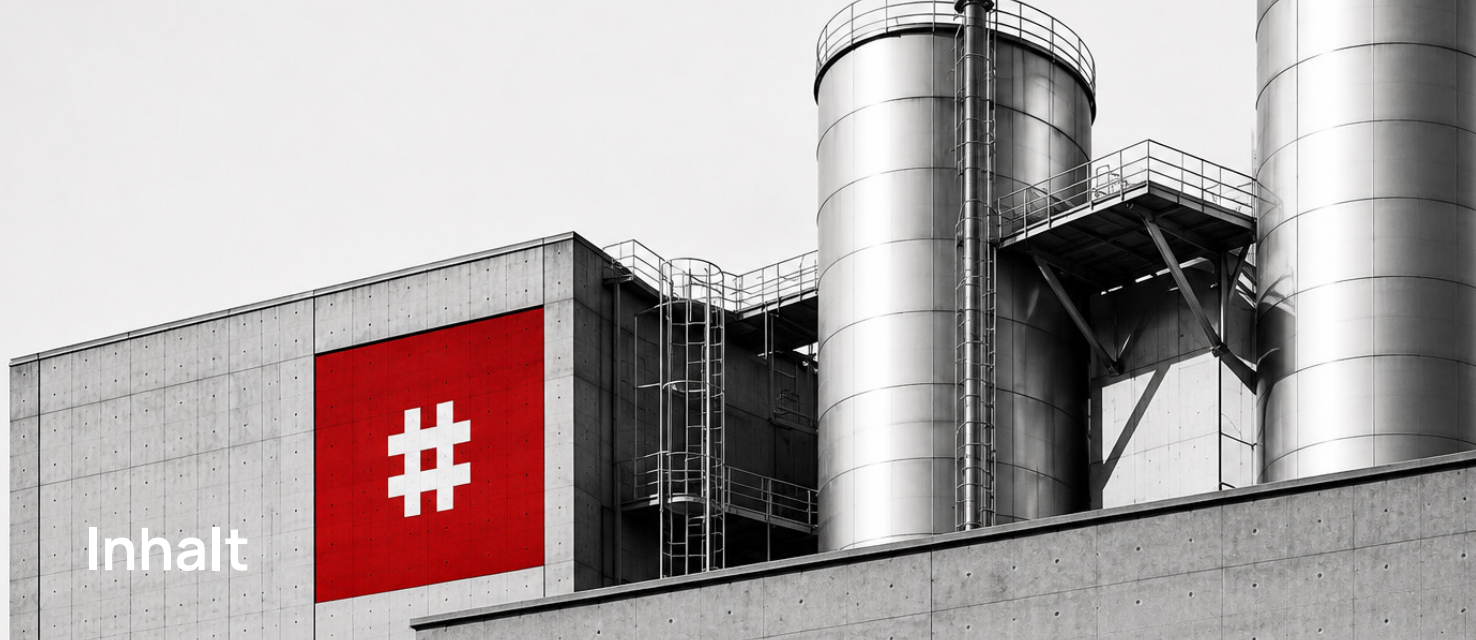
# Skill-basierte Automatisierung Die Software-definierte Fabrik

Schluss mit Rip and Replace:  
Die Fabrik denkt Steuerung neu.

---

Zwischen harten **Echtzeit-Anforderungen**, **funktionaler Sicherheit** und **jahrzehntelangem Bestandsschutz** stehen Fertigungsunternehmen im Brownfield vor einem scheinbaren Widerspruch: **Mehr Flexibilität wagen, ohne bewährte Steuerungstechnik zu riskieren.** Dieses Whitepaper zeigt, wie eine **Skill-basierte Architektur** genau diesen Spagat löst.

Erfahren Sie, warum die **Kapselung von SPS-Funktionen in wiederverwendbare Skills zur neuen Kernkompetenz Software-definierter Fabriken** wird und wie sich Maschinen- und Anlagenbauer<sup>1</sup> schon heute technologisch und organisatorisch auf die **Orchestrierung von morgen** vorbereiten können.



# Inhalt

## KAPITEL 1

**Das Brownfield-Dilemma – Warum „SPS raus, Software rein“ in der Praxis scheitert** 4

---

## KAPITEL 2

**Vom SPS-Baustein zum Skill – Die Zielarchitektur** 7

---

## KAPITEL 3

**Migrationspfade – Wie der Umbau konkret läuft** 11

---

## KAPITEL 4

**Edge-Plattformen im Vergleich** 14

---

## KAPITEL 5

**Umsetzung in der Praxis** 17

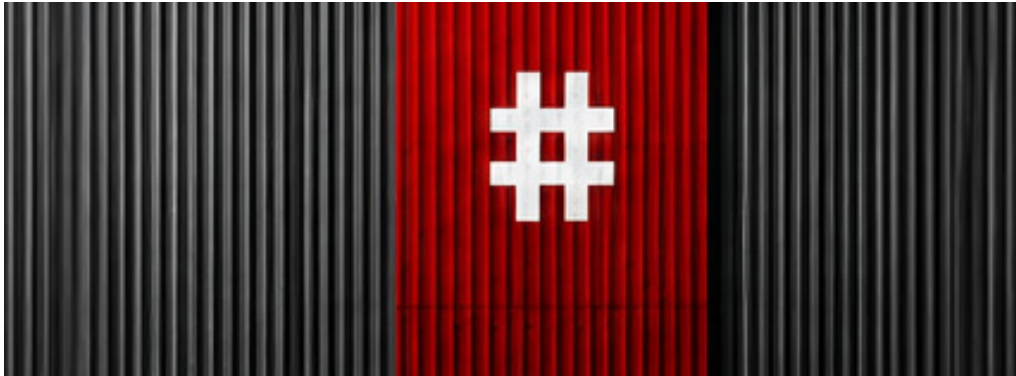
---

## KAPITEL 6

**Fazit und Ausblick – Evolution statt Revolution im Brownfield** 20

# Zusammenfassung

---



**Muss die SPS raus, damit Software rein kann?** Wer die Debatte zur **Software-definierten Fabrik** so führt, übersieht die technischen und wirtschaftlichen Realitäten des Brownfields: Echtzeit-Anforderungen, zertifizierte Safety-Funktionen und Anlagen, die auf 15 bis 30 Jahre Laufzeit ausgelegt sind.

**Ein radikaler Umbau der Steuerungsebene ist unter diesen Bedingungen weder technisch geboten noch wirtschaftlich vertretbar.**

Dieses Whitepaper zeigt eine Lösung mit **Schichtung statt Ablösung**. Die SPS bleibt deterministisch, zuverlässig, sicherheitskritisch. Was sich ändert, ist ihre Rolle im Gesamtsystem. Statt alleinige Trägerin der gesamten Prozesslogik zu sein, wird sie zum spezialisierten Ausführungsorgan innerhalb einer geschichteten Architektur. Zeitkritische und sicherheitsrelevante Funktionen verbleiben auf der Steuerung,

variable Abläufe, Orchestrierung und Wiederverwendung wandern in modular verwaltbare Software-Skills. Atomar oder zusammengesetzt, je nach Komplexität des Prozessschritts.

Enthalten sind **Migrationskriterien für die Entscheidung**, welche Funktion auf der SPS bleibt und welche auf die Edge-Ebene wandert, einen **szenariobasierten Vergleich gängiger Edge-Plattformtypen** jenseits reiner Datenblatt-Kriterien sowie **Tools zur direkten Anwendung**: Skill-Steckbrief, SPS-to-Skill-Migrationscanvas, Edge-Auswahlmatrix und eine 90-Tage-Pilot-Checkliste.

Ziel ist ein **Praxisleitfaden**, um Brownfield-Anlagen Schritt für Schritt in Richtung Software-definierter Fertigung weiterentwickeln ohne eine einzige Zeile sicherheitskritischen SPS-Codes anzutasten und ohne die betriebliche Stabilität aufs Spiel zu setzen. **#retrofit**

---

<sup>1</sup> Im Sinne der besseren Lesbarkeit wird in diesem Dokument das generische Maskulinum verwendet. Dies impliziert keine Wertung, sondern dient ausschließlich der sprachlichen Vereinfachung; gemeint sind stets alle Geschlechtsidentitäten.

# Kapitel 1

## Das Brownfield-Dilemma

Warum „SPS raus, Software rein“ in der Praxis scheitert

In der Debatte um die moderne, Software-definierte Fabrik dominieren oft radikale Zukunftsvisionen.

Die klassische Speicherprogrammierbare Steuerung (SPS) sei ein Relikt der Vergangenheit, inflexible monolithische Bausteine gehörten ausrangiert und durch rein cloud- oder edge-basierte Software-Architekturen ersetzt.

Für die **Praxis im Maschinenbau** und in der **produzierenden Industrie** greift diese Vorstellung jedoch zu kurz.

Gerade im Brownfield treffen neue Softwarekonzepte auf Anlagen, die über Jahre gewachsen sind, zuverlässig produzieren und häufig noch lange wirtschaftlich betrieben werden sollen.

Wer versucht, gewachsene SPS-Strukturen nach dem Motto „**Rip and Replace**“ radikal aus dem Spiel zu nehmen, trifft in der Realität schnell auf hohe Hürden:

- **Harte Echtzeit-Anforderungen:** Deterministische Taktzeiten im einstelligen Millisekundenbereich lassen sich auf hochgelagerten Software-Ebenen nicht ohne extremen Aufwand garantieren.
- **Funktionale Sicherheit (Safety):** Sicherheitskritische Abschaltungen, Not-Halt-Ketten und normenkonforme Schutzfunktionen (SIL / Performance Level) sind tief in der dedicated SPS-Hardware verankert und regulatorisch zertifiziert.

- **Wirtschaftlicher Bestandschutz:** Anlagen im Brownfield sind auf Laufzeiten von 15 bis 30 Jahren ausgelegt. Ein kompletter Austausch der Steuerungsebene ist finanziell und operational schlicht unrentabel.

*Das eigentliche Brownfield-Dilemma liegt damit nicht zwischen „alter“ und „neuer“ Technologie. Es liegt zwischen zwei berechtigten Anforderungen: Produktionsunternehmen benötigen einerseits **stabile, sichere und deterministische Steuerung**. Andererseits müssen sie **Abläufe schneller an neue Produkte, Varianten, Aufträge und digitale Services anpassen** können.*

**Die Software-definierte Fabrik löst dieses Spannungsfeld nicht durch Ablösung, sondern durch Schichtung.**

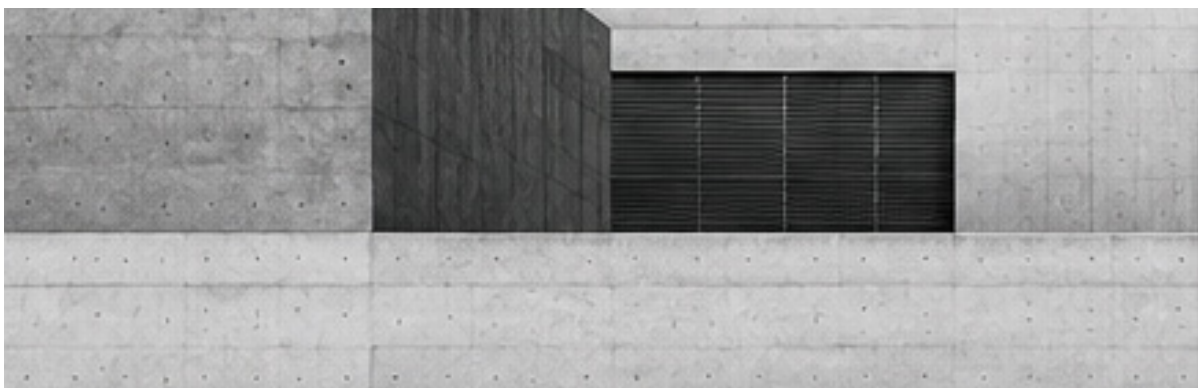
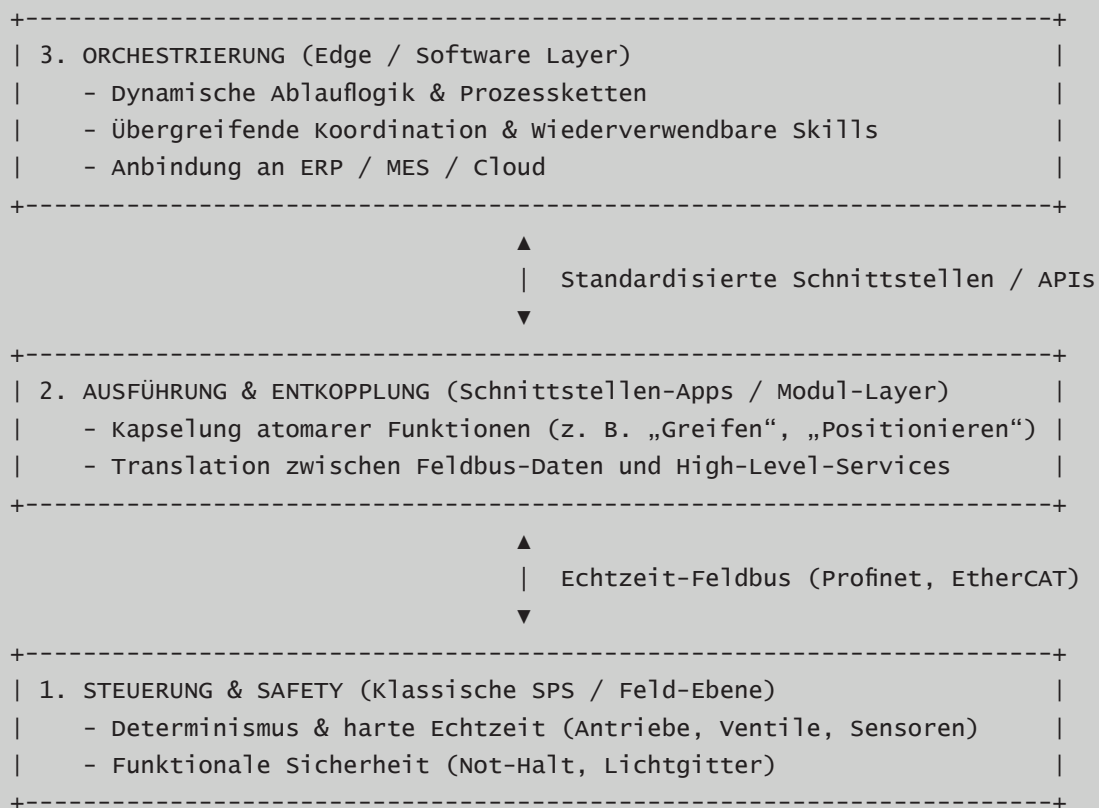
**Die SPS bleibt dort, wo ihre Stärken gebraucht werden.** Variable Abläufe, übergreifende Koordination und die Anbindung an IT-Systeme können dagegen schrittweise in flexiblere Softwareebenen verlagert werden.

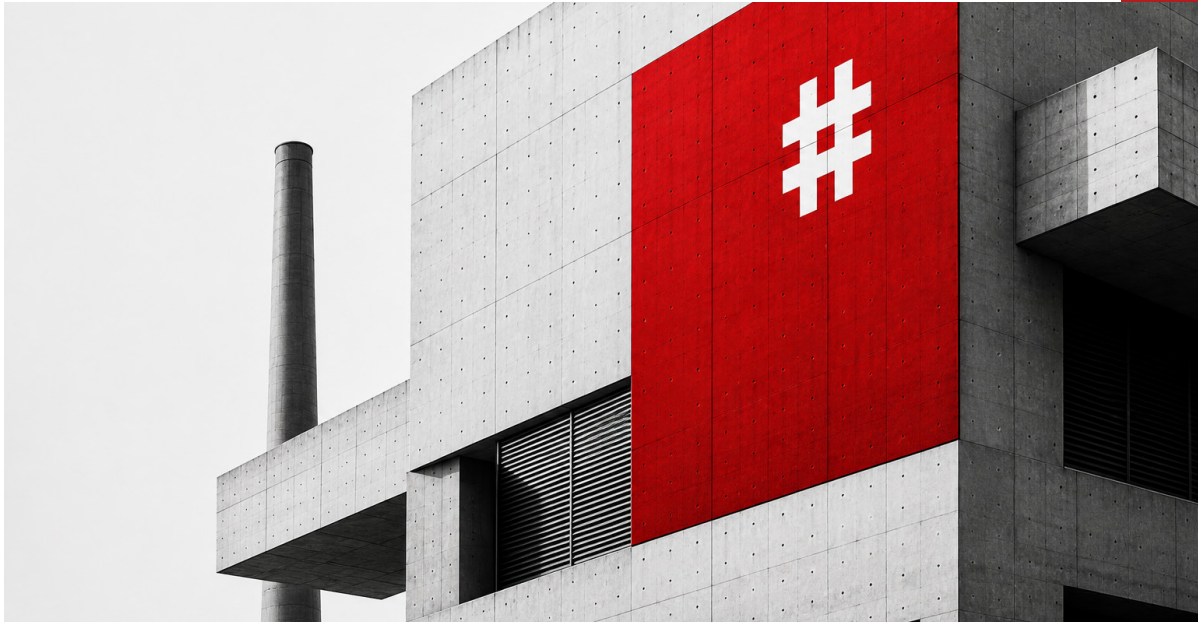
## Drei Ebenen statt eines radikalen Umbaus

Für den Einstieg genügt zunächst eine einfache Unterscheidung. Die SPS wird damit vom alleinigen Träger der gesamten Prozesslogik zu einem spezialisierten Ausführungsorgan innerhalb einer geschichteten Architektur. Änderungen an Produktionsabläufen müssen dann nicht zwangsläufig tief im Steuerungscode erfolgen.

## Schichtung statt Ablösung: Die Steuerung neu denken

Die Software-definierte Fabrik ist keineswegs SPS-frei. **Die Steuerung wird nicht aus dem Werk getragen und feierlich im Hof verbrannt.** Um Flexibilität im Brownfield zu erreichen, müssen Verantwortliche in IT und OT die Funktionsblöcke der Produktion klar voneinander abgrenzen und neu schichten.





## Die drei Ebenen auf einen Blick

Für den Übergang zur modularen Produktion gilt es, Aufgaben anhand ihrer Anforderung an Echtzeit und Variabilität auf die passenden Ebenen zu verteilen:

### 1. Steuerung

*(Control Layer – SPS)*

- **Aufgaben:** Harte Echtzeit, Signalverarbeitung, direkte Aktorik/Sensorik, Sicherheitsfunktionen.
- **Charakter:** Extrem stabil, selten geändert, proprietär verankert.
- **Fokus:** Determinismus und physische Zuverlässigkeit.

### 2. Ausführung

*(Execution Layer – Modul / App)*

- **Aufgaben:** Kapselung konkreter Teilfunktionen zu definierten Grundbausteinen (atomare Skills), Übersetzung von SPS-Zuständen in aufrufbare Schnittstellen.
- **Charakter:** Abstrahiert die zugrunde liegende Hardware-Hardware-Topologie.
- **Fokus:** Standardisierung und Entkopplung.

### 3. Orchestrierung

*(Orchestration Layer – Edge / Server)*

- **Aufgaben:** Zusammensetzung von Prozessketten (Composite Skills), variable Abläufe, Auftragssteuerung, Anbindung an übergeordnete IT-Systeme.
- **Charakter:** Flexibel anpassbar, in modernen Programmiersprachen (z. B. Python, Go, C#) umgesetzt, dynamisch skalierbar.
- **Fokus:** Wiederverwendbarkeit und Transparenz.

Wir sehen: Das Ziel einer zukunftsfähigen Architektur ist es nicht, die SPS durch Software zu ersetzen – sondern die SPS durch Skill-basierte Kapselung so von der Prozesslogik zu entkoppeln, dass Änderungen an den Fertigungsabläufen ohne Eingriff im IEC-61131-Code der Steuerung möglich werden.

## Kapitel 2

# Vom SPS-Baustein zum Skill Die Zielarchitektur

Die entscheidende Veränderung besteht nicht darin, SPS-Funktionen einfach an einen anderen Ort zu verschieben.

Sie besteht darin, Funktionen so zu kapseln, dass andere Systeme sie eindeutig aufrufen, überwachen und kombinieren können.

## An dieser Stelle kommen Skills in Spiel.

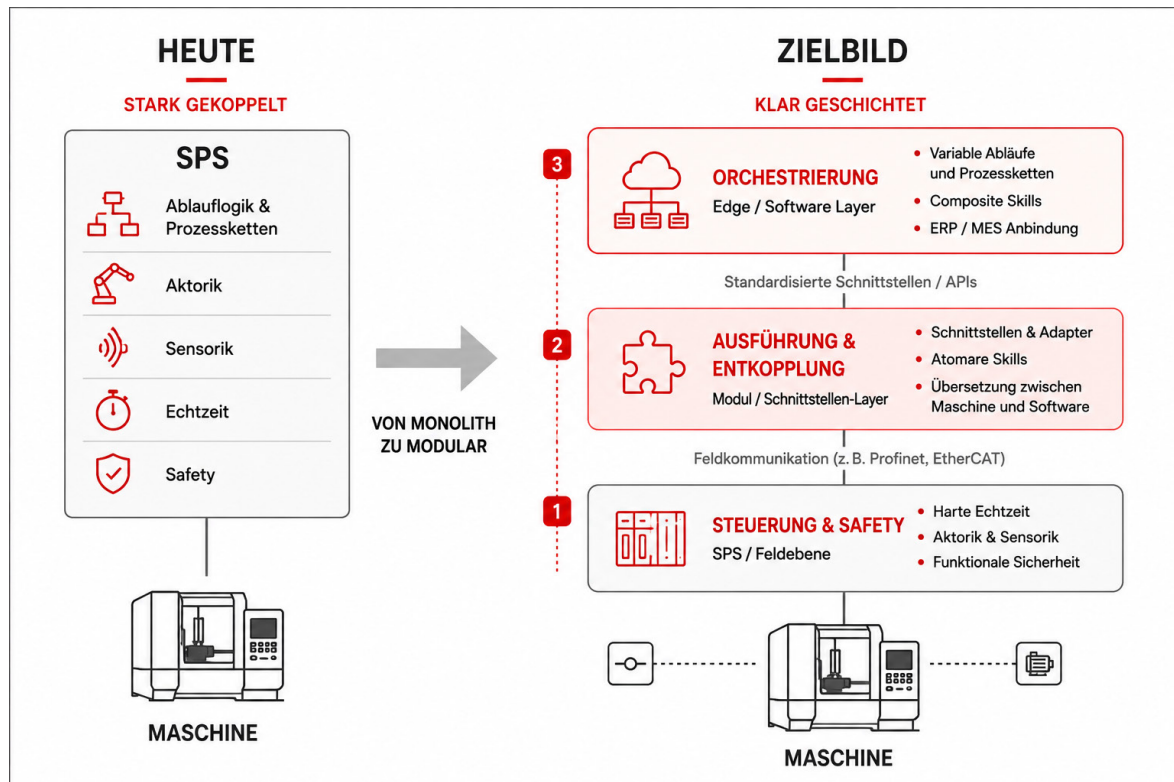
Ein Skill beschreibt eine klar abgegrenzte Fähigkeit einer Maschine oder Komponente, beispielsweise „Greifer schließen“, „Achse positionieren“ oder „Bauteil von Position A nach Position B bewegen“. Entscheidend ist dabei nicht, in welcher Programmiersprache oder auf welchem Gerät diese Funktion intern umgesetzt ist. **Entscheidend ist der Vertrag nach außen.**

Ein solcher Vertrag beantwortet vor allem folgende Fragen:

- Was kann die Funktion?
- Welche Informationen benötigt sie für den Start?
- Welche Rückmeldungen liefert sie?
- Woran erkennt ein übergeordnetes System, ob sie bereit, aktiv, abgeschlossen oder fehlerhaft ist?
- Welche Fehler und Zeitüberschreitungen können auftreten?
- Welche Verantwortung verbleibt ausdrücklich bei der Steuerungs- und Safety-Ebene?

**Damit wird der Skill zur stabilen Schnittstelle zwischen Automatisierung und Software. Seine interne Implementierung kann sich verändern, solange der vereinbarte Vertrag erhalten bleibt.**





## Der Skill als technischer Vertrag

Die Umstellung von einer monolithischen SPS-Programmierung auf eine modulare Architektur erfordert eine Neudefinition der Schnittstellen zwischen Steuerungsebene und Softwarearchitektur. Anstelle von direkt aufgerufenem Funktionsbaustein-Code oder unstrukturierten E/A-Signalen tritt das Konzept des Skills.

**Ein Skill ist eine gekapselte, wiederverwendbare Fähigkeit einer Maschine oder Komponente.**

Aus technischer Sicht handelt es sich um einen definierten Vertrag (Contract) zwischen dem Anbieter der Funktion (der Maschine bzw. dem Modul) und dem Konsumenten (der Orchestrierungsebene).

**Ein Skill verbirgt die interne Implementierung.** Der aufrufenden Ebene ist nicht bekannt, ob die Funktion intern durch einen IEC-61131-Code auf einer SPS, eine Python-Anwendung auf einem Edge-Device oder eine Kombination aus beidem ausgeführt wird.

## Atomare und zusammengesetzte Skills

Für die Strukturierung von Produktionsabläufen unterscheidet die Architektur zwei Skill-Klassen:

### 1. Atomare Skills

Atomare Skills repräsentieren die kleinste, nicht weiter zerlegbare Funktionseinheit einer Hardwarekomponente. Sie greifen direkt auf die Aktorik und Sensorik zu und verbleiben in der Regel auf der Steuerungsebene.

Beispiel für einen atomaren Skill wären: *PneumatikZylinder\_Ausfahren*, *Greifer\_Schließen*, *Achse\_MussPositionieren*.

### 2. Zusammengesetzte Skills

*Composite Skills* bilden komplexe Prozessschritte ab, indem sie mehrere atomare Skills in einer definierten Abfolge oder Parallelität verknüpfen.

Die Logik zur Koordinierung dieser Abläufe liegt bevorzugt auf einer höheren Software-Ebene, z. B. einem Edge-Knoten.

Zum Beispiel: Ein Pick-and-Place-Vorgang setzt sich aus mehreren atomaren Schritten zusammen:

1. **Skill Achse\_Positionieren**  
(über der Zielposition)
2. **Skill PneumatikZylinder\_Ausfahren**  
(Greifer senken)
3. **Skill Greifer\_Schließen**  
(Bauteil fassen)
4. **Skill PneumatikZylinder\_Einfahren**  
(Greifer anheben)

Durch diese Trennung muss bei Änderungen des Pick-and-Place-Ablaufs nicht der Code in der SPS geändert werden; es wird lediglich der Ablauf der Skill-Aufrufe im Composite Skill auf der Edge-Ebene angepasst.

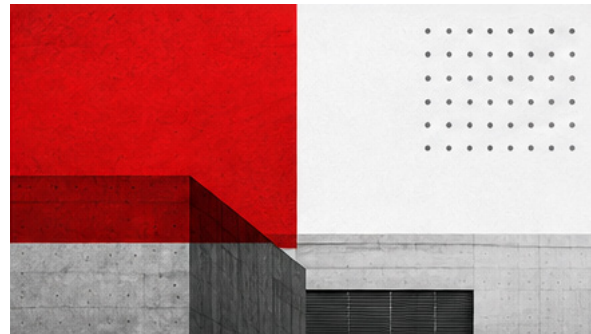
### Zustandslogik und Schnittstellen-Standardisierung

Damit Skills zuverlässig orchestriert werden können, benötigen sie ein einheitliches Zustandsmodell. Ein bewährter Industrie-Standard ist die

Ausrichtung an Zustandsmodellen wie **PackML (ISA-TR88.00.02)** oder der **Weihenstephaner Standards**.

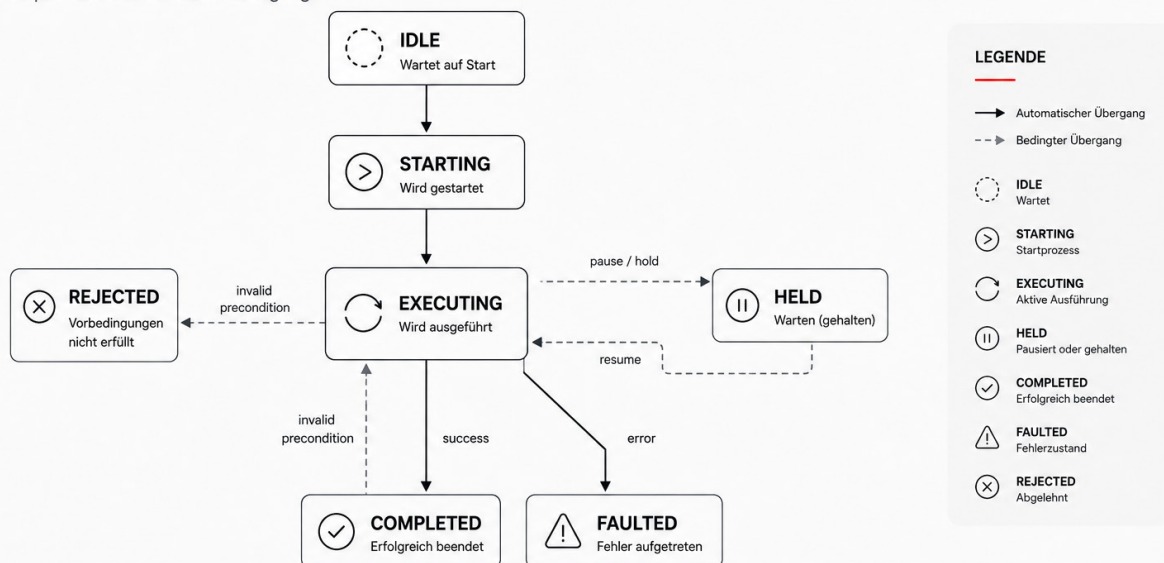
Jeder Skill durchläuft klar definierte Zustände:

- **IDLE**: Der Skill ist bereit zur Ausführung und wartet auf Parameter und das Start-Signal.
- **EXECUTE**: Die Funktion wird aktiv ausgeführt. Die Schnittstelle liefert zyklisch Fortschritts- und Statusdaten.
- **COMPLETED**: Die Aufgabe wurde erfolgreich beendet.
- **HELD / FAULTED**: Die Ausführung wurde unterbrochen oder aufgrund eines Fehlers gestoppt.



## ZUSTANDSMODELL EINES SKILLS

Beispielhafte Zustände und Übergänge





## SKILL-STECKBRIEF



### 1. METADATEN

Skill-Name: .....

Skill-Typ: ☐ Atomar ☐ Composite

Ebene der Ausführung: ☐ SPS (IEC 61131) ☐ Edge / Gateway ☐ Cloud

ID / Version: .....

Kurzbeschreibung (1–2 Sätze): .....



#### TIPP

Wählen Sie einen klaren, sprechenden Namen und halten Sie die Beschreibung kurz und präzise.



### 2. FUNKTIONALE BESCHREIBUNG

Was macht der Skill und welchen Nutzen schafft er?

.....

.....

.....



#### TIPP

Beschreiben Sie den Geschäftsnutzen und die Funktion in einfachen Worten – ohne technische Details.



### 3. SCHNITTSTELLEDEFINITION (CONTRACT)

Eingangsparameter (Input):

| Parameter | Datentyp | Einheit | Beschreibung |
|-----------|----------|---------|--------------|
|           |          |         |              |
|           |          |         |              |
|           |          |         |              |

Ausgangswerte (Output / Telemetrie):

| Parameter | Datentyp | Einheit | Beschreibung |
|-----------|----------|---------|--------------|
|           |          |         |              |
|           |          |         |              |
|           |          |         |              |



#### TIPP

Definieren Sie nur das Nötigste. Halten Sie Datentypen, Einheiten und semantische Bedeutung konsistent.



### 4. ZUSTANDSÜBERGÄNGE & TIMEOUTS

Maximale Ausführungszeit (Timeout): .....

Verhalten bei Not-Halt: .....

Weitere Zustände / Übergänge (optional): .....



#### TIPP

Definieren Sie klare Timeouts und ein vorhersehbares Verhalten für Sicherheit und Stabilität.



### 5. TECHNISCHE ANBINDUNG / PROTOKOLL

Protokoll / Schnittstelle: .....

Kommunikationsweg / Adresse: .....

Sicherheits- / Safe-State-Verantwortung: .....

Abhängigkeiten (z. B. Skills, Systeme): .....

Versionierung / Kompatibilität: .....



#### TIPP

Dokumentieren Sie die Anbindung und klare Verantwortlichkeiten für Sicherheit und Betrieb.



#### NOTIZEN

.....

.....

.....



Erstellt am: .....

Erstellt von: .....

Letzte Änderung: .....

Freigegeben von: .....

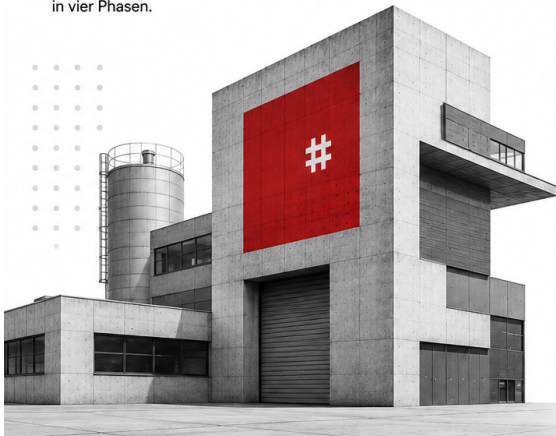
## Kapitel 3

# Migrationspfade – Wie der Umbau konkret läuft

### #BB

## Schrittweiser Umbau im Brownfield

Ein plötzlicher Systemwechsel birgt in bestehenden Fertigungsanlagen unkalkulierbare Risiken bezüglich Ausfallzeiten und Validierung. Die Migration von einer monolithischen SPS-Architektur hin zu einer Skill-basierten, Software-definierten Struktur erfolgt daher schrittweise in vier Phasen.



#### PHASE 1: Entkopplung & Transparenz

- Lesen der SPS-Daten über Read-Only-Schnittstellen (z. B. OPC UA)
- Keine Eingriffe in die bestehende Ablauflogik



#### PHASE 2: Kapselung atomarer Funktionen

- Strukturierung des SPS-Codes in eigenständige Funktionsbausteine
- Bereitstellung definierter Steuerungsschnittstellen (Skills)



#### PHASE 3: Auslagerung der Orchestrierung

- Verlagerung der übergeordneten Sequenzlogik auf eine Edge-Plattform
- Die SPS führt nur noch aufgerufene Befehle aus



#### PHASE 4: Dynamische Rekonfiguration

- Flexibles Einbinden neuer Abläufe und Module ohne SPS-Re-Kompilierung

Je zeitkritischer und sicherheitsnäher eine Funktion ist, desto näher bleibt sie an der Steuerung.

Je variabler, datengetriebener und häufiger veränderbar sie ist, desto interessanter wird die Softwareebene.

### Rollenverteilung: Was verbleibt auf der SPS, was wandert ab?

Die Entscheidung, welche Funktionen auf der Steuerung verbleiben und welche auf höhere Software-Ebenen verlagert werden, erfolgt nach klaren Kriterien:

| Kriterium                   | Verbleibt auf der SPS<br>(Control Layer)                         | Wandert auf Edge / Server<br>(Orchestrierung)        |
|-----------------------------|------------------------------------------------------------------|------------------------------------------------------|
| Echtzeit & Determinismus    | Harte Echtzeit (<10" ms" ),<br>zyklische Task-Abarbeitung        | Weiche oder keine Echtzeit (>50" ms" )               |
| Funktionale Sicherheit      | Not-Halt, Lichtgitter, SIL- /<br>PL-Logik                        | Monitoring von Sicherheits-<br>zuständen (Read-Only) |
| Aktorik- & Sensorik-Zugriff | Direkte E/A-Verarbeitung, Achsre-<br>gelungen, Ventilsteuerungen | Aufruf aggregierter Befehle (z. B.<br>Positionieren) |
| Prozess- & Ablauflogik      | Starre, zeitkritische Verriege-<br>lungen                        | Variable Reihenfolgen, Rezeptur-<br>verwaltung       |
| Datenverarbeitung           | Keine komplexe Datenverarbei-<br>tung                            | Datenaggregation, KI-Analysen,<br>IT-Anbindungen     |

### SPS-to-Skill-Migration

Der Umstieg von einer *monolithischen SPS-Programmierung* auf eine *Skill-basierte Architektur* beginnt im Bestand: Jede Anlage trägt **Jahre gewachsenen Steuerungscode** in sich, dessen Funktionen selten von Anbeginn dokumentiert und noch seltener nach Echtzeit- oder Sicherheitsanforderungen sortiert sind. Das folgende Arbeitsmittel unterstützt Projektteams bei der **Analyse bestehender SPS-Programme und der Einteilung von Funktionen in die neue Zielarchitektur**.

Das Canvas führt in **fünf strukturierten Schritten** durch die Analyse eines bestehenden Anlagenbereichs. Zunächst wird der **Ist-Zustand** erfasst: *Welche Funktionsbausteine, Merker und Datenbausteine steuern den Prozess heute tatsächlich?* Im **zweiten Schritt** folgt die **Klassifizierung nach Sicherheit und Echtzeitanforderung**. Diese beiden Fragen allein entscheiden meist schon, ob eine Funktion auf der Steuerung verbleiben muss oder für eine Verlagerung infrage kommt.

Erst danach wird es konstruktiv: **Schritt drei** definiert, welche Variablen als **Parameter und Statuswerte künftig als Skill-Schnittstelle** nach außen geführt werden müssen. **Schritt vier** überträgt die Analyse in eine konkrete Ziel-Klassifizierung. **Jeder Funktionsbaustein bekommt eine künftige Rolle (atomarer oder zusammengesetzter Skill) und eine Ziel-Ebene zugewiesen**. Der **fünfte Schritt** schließlich macht den Übergang messbar. Eine **Checkliste** objektiver Kriterien, anhand derer ein Team *selbst* beurteilen kann, ob ein Baustein reif für die nächste Migrationsphase ist; statt sich auf Bauchgefühl zu verlassen.

Das Ergebnis ist eine dokumentierte, nachvollziehbare **Entscheidungsgrundlage**. *Was bleibt aus gutem Grund auf der SPS, was wandert wohin, und woran erkennt das Team, dass der nächste Schritt sicher gegangen werden kann?* Ausgefüllt am Beispiel einer Förderstrecke mit Hubstation zeigt das Canvas, wie eine solche Analyse in der Praxis aussieht.

Sehen Sie ein Beispiel auf der nächsten Seite.

## #BB

## SPS-TO-SKILL-MIGRATIONSCANVAS



Anlagenbereich / Modul:  
**Förderstrecke & Hubstation\_02**

Datum:  
**2026-08-06**

Bearbeiter:  
**IT/OT-Architekturteam**

**1. BESTANDSERFASSUNG (AS-IS SPS-CODE)**

Welche Bausteine, Merker oder Variablen steuern aktuell den Prozess?

- FB\_Hubtisch\_Control (Steuert Hydraulikventile und Endschalter)
- FC\_Band\_Ablauf (Enthält Schrittkette für den Weitertransport)
- DB\_Prozessdaten (Speichert Positionen und Statuswerte)

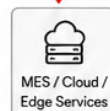
AS-IS  
(SPS-WELT)

SPS



TO-BE (SKILL-WELT)

Orchestrierung

**2. SICHERHEITS- & ECHTZEIT-KLASSIFIZIERUNG**

- Enthält der Baustein funktionale Sicherheit (Safety)?

☒ Ja ☐ Nein

**Maßnahme:** Sicherheitsrelevante Abschaltungen verbleiben unverändert im Safety-Teil der SPS.

- Erfordert die Funktion harte Echtzeit (<10 ms)?

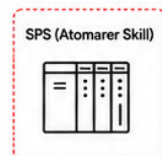
☒ Ja ☐ Nein

**Maßnahme:** Die direkte Ansteuerung der Hydraulikventile bleibt auf der SPS.

**3. SCHNITTSTELLEN-KAPSELUNG (SPS-EBENE)**

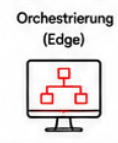
Welche Variablen müssen als Parameter für den zukünftigen Skill nach außen geführt werden?

- **Eingänge (Control):** Target\_Height (INT), Start\_Move (BOOL)
- **Ausgänge (Status):** Position\_Reached (BOOL), Fault (BOOL), Current\_Height (INT)



Parameter  
(Command)

Status / Telemetry  
(Response)

**4. ZIEL-KLASSIFIZIERUNG & ZIEL-EBENE**

| Funktionsbaustein / Logik           | Zukünftige Rolle                  | Ziel-Ebene                         | Begründung / Hinweise |
|-------------------------------------|-----------------------------------|------------------------------------|-----------------------|
| Hydraulik-Ansteuerung & Endschalter | Atomarer Skill: Hubtisch_Fahren   | SPS (IEC 61131)                    |                       |
| Band-Antrieb & Lichtschranken       | Atomarer Skill: Band_Fördern      | SPS (IEC 61131)                    |                       |
| Schrittkette (Ablauflogik)          | Composite Skill: Transfer_Palette | Edge-Plattform (Python / Node-RED) |                       |
| Rezeptur- & Auftragsdaten           | Orchestrierungs-Service           | MES / Cloud / Edge                 |                       |

**5. KRITERIEN FÜR DIE ERREICHUNG DER NÄCHSTEN PHASE**

- ☐ SPS-Code ist in sich geschossen und ohne direkte Zugriffe auf globale Merker testbar.
- ☐ Standardisiertes Zustandsmodell (z. B. PackML) ist im SPS-Baustein implementiert.
- ☐ Der Skill ist über OPC UA / MQTT von externen Systemen aufrufbar.
- ☐ Ein manueller Testaufruf des Skills über die Edge-Ebene war erfolgreich.



**TIPP**

Starten Sie mit einem überschaubaren Funktionsbaustein, kapseln Sie die Schnittstelle sauber und beweisen Sie den Mehrwert in einem Pilot.

## Kapitel 4

# Edge-Plattformen im Vergleich

### Kriterien jenseits von Datenblatt-Auswertungen

Die Wahl der geeigneten Edge-Plattform ist eine *zentrale Entscheidung* bei der Umsetzung einer Software-definierten Fabrikarchitektur. In der Praxis greifen reine Hardwaredatenblätter (wie Prozessorleistung, Arbeitsspeicher oder Gehäuse-schutzart) meist zu kurz. Für den stabilen Betrieb einer Skill-basierten Orchestrierung im Brownfield sind vor allem Kriterien der Software-Architektur, des Betriebes und der OT-Integration entscheidend.

Folgende fünf Kernkriterien stehen bei der Bewertung im Vordergrund:

- 1. OT-Konnektivität & Protokollunterstützung:**  
Nativer Support für etablierte Industrieprotokolle (OPC UA, Profinet, Modbus TCP, EtherCAT) sowie die Fähigkeit zur Übersetzung in IT-Protokolle (MQTT, REST/JSON).
- 2. Entkopplung & Containerisierung:**  
Unterstützung von Container-Technologien (z. B. Docker, WebAssembly), um Orchestrierungs-Software, Schnittstellen-Apps und Logikmodule isoliert voneinander auszuführen und zu aktualisieren.
- 3. Lifecycle- & Device-Management:**  
Möglichkeiten für zentrale Konfiguration, Over-the-Air-Updates (OTA), Rollbacks, Zertifikatsverwaltung und Monitoring der Edge-Geräte über große Flotten hinweg.
- 4. Echtzeit- & Determinismus-Fähigkeiten:**  
Eignung des Betriebssystems (z. B. Real-Time Linux mit PREEMPT\_RT) für Aufgaben mit geringer Latenz oder deterministischen Anforderungen.

- 5. Herstellerbindung (Lock-in) / Interoperabilität:**  
Offenheit der Schnittstellen, Nutzung von Open-Source-Standards und die Möglichkeit, Code unabhängig von spezifischer Hardware zu portieren.

### Szenariobasierter Vergleich von vier Edge-Plattformtypen

Um die Entscheidungsfindung zu erleichtern, werden im Folgenden vier repräsentative Typen von Edge-Plattformen verglichen, wie sie in der Industrie vorzufinden sind:

#### Typ A: SPS-integrierte Edge (Steuerungs-Add-on)

**Charakteristika:** Die Edge-Funktionalität läuft direkt als Coprozessor oder in einer geschützten Laufzeitumgebung auf der bestehenden SPS-Hardware.

**Vorteile:** Sehr geringe Latenz zum Steuerungskern, keine zusätzliche Hardware im Schaltschrank erforderlich, direkte Nutzung bestehender Busstrukturen.

**Grenzen:** Begrenzte Rechenressourcen, oft enge Koppelung an das Ökosystem des jeweiligen SPS-Herstellers.

#### Typ B: Dedicated Industrial Edge (Proprietäres Ökosystem)

**Charakteristika:** Spezialisierte Industrial-PC-Hardware und abgestimmte Software-Stacks etablierter Automatisierungsanbieter inklusive eigenem App-Store.

**Vorteile:** Hohe Robustheit, vorgefertigte Konnektoren für die OT-Welt, komfortables Flottenmanagement aus einer Hand.

**Grenzen:** Abhängigkeit vom Update- und Lizenzmodell des Herstellers, eingeschränkte Flexibilität bei der Wahl eigener Entwicklungswerkzeuge.

#### **Typ C:** Offene Industrie-PC-Plattform (Linux-basiert)

**Charakteristika:** Standard-Industrie-Hardware (x86 / ARM) mit einem gehärteten Linux-Betriebssystem und einer Open-Source-Container-Laufzeitumgebung (z. B. Docker, K3s).

**Vorteile:** Maximale Flexibilität, keine Lizenzkosten für Runtimes, freie Wahl der Programmiersprachen (Python, Go, Rust, C#) und vollständige Vermeidung eines Vendor-Lock-ins.

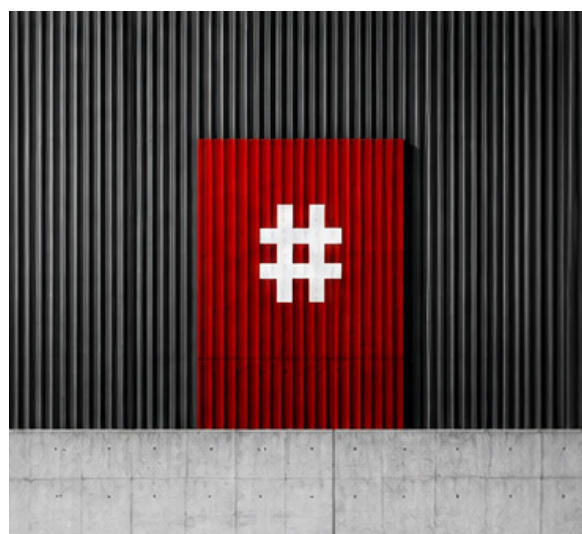
**Grenzen:** Höherer Eigenaufwand für Systemhärtung, Security-Patches und den Aufbau eines eigenen Lifecycle-Managements.

#### **Typ D:** Cloud-Hyperscaler Edge Extension

**Charakteristika:** Lokale Edge-Runtimes, die direkt von einem Cloud-Anbieter gesteuert und verwaltet werden.

**Vorteile:** Nahtlose Integration in Cloud-Dienste, einheitliche Entwicklungs-Pipeline von der Cloud bis zur Fabrik, hervorragende Analysemöglichkeiten.

**Grenzen:** Abhängigkeit von einer stabilen Cloud-Anbindung für Management-Aufgaben, potenzielle Latenzfragen und datenschutzrechtliche Vorgaben.



## Plattform-Typen

|       |                                                                  |                                                      |
|-------|------------------------------------------------------------------|------------------------------------------------------|
| Typ A | Harte Echtzeit (<10" ms" ),<br>zyklische Task-Abarbeitung        | Weiche oder keine Echtzeit<br>(>50" ms" )            |
| Typ B | Not-Halt, Lichtgitter,<br>SIL- / PL-Logik                        | Monitoring von Sicherheits-<br>zuständen (Read-Only) |
| Typ C | Direkte E/A-Verarbeitung, Achsre-<br>gelungen, Ventilsteuerungen | Aufruf aggregierter Befehle<br>(z. B. Positionieren) |
| Typ D | Starre, zeitkritische Verriegel-<br>ungen                        | Variable Reihenfolgen,<br>Rezepturverwaltung         |

**#BBKOMPACT:** EDGE-PLATTFORM-TYPEN IM ÜBERBLICK

# Edge-Auswahlmatrix

Die nachfolgende Bewertungsmatrix dient als strukturierter Rahmen für die Auswahl einer geeigneten Edge-Plattform auf Basis spezifischer Anforderungsprofile.

## Anwendungshinweis für Projektteams:

1. Tragen Sie in der Spalte Gewichtung die projektspezifischen Prioritäten Ihres Unternehmens ein (1–3).
2. Bewerten Sie die infrage kommenden Edge-Systeme anhand der Kriterien (1–5).

3. Multiplizieren Sie Punkte mit Gewichtung und summieren Sie die Werte.
4. **Empfehlung:** Für zeitkritische Skill-Orchestrierungen im Brownfield zeigen in der Regel Kombinationen aus Typ B (bei bestehenden Monokulturen) oder Typ C (bei heterogenen Anlagenparks) das beste Nutzen-Risiko-Verhältnis. **#BBTIPP**

GEWICHTUNG: 1 = NIEDRIGE PRIORITÄT, 3 = HOHE PRIORITÄT

BEWERTUNG: 1 = SCHLECHT GEEIGNET, 5 = HERVORRAGEND GEEIGNET

| Bewertungskriterium                                      | Gewichtung (G) | Typ A          | Typ B          | Typ C          | Typ D          |
|----------------------------------------------------------|----------------|----------------|----------------|----------------|----------------|
| 1. OT-Konnektivität (Protokollvielfalt, Feldbus-Zugriff) | 3              | 5              | 4              | 3              | 2              |
| 2. Flexibilität & Open Source (Container, Sprachen)      | 2              | 2              | 3              | 5              | 3              |
| 3. Flotten- & Patchmanagement (Central Deployment)       | 3              | 2              | 4              | 3              | 5              |
| 4. Latenz & Echtzeitverhalten (<20" ms" )                | 2              | 5              | 3              | 3              | 1              |
| 5. Unabhängigkeit (Vendor-Lock-in)                       | 2              | 1              | 2              | 5              | 2              |
| 6. Betriebskosten (TCO & Lizenzen)                       | 2              | 3              | 2              | 4              | 2              |
| <b>Gesamtwertung (Gewichtet):</b>                        | <b>-</b>       | <b>56 / 70</b> | <b>55 / 70</b> | <b>59 / 70</b> | <b>44 / 70</b> |

# Kapitel 5

## Umsetzung in der Praxis

### Referenzarchitektur in der Anwendung (Kompakter Walk-Through)

Dass das **Zusammenspiel von Steuerungsebene und Skill-basierter Orchestrierung im Brownfield *funktionsfähig*** ist, zeigt die praktische Umsetzung im Rahmen des Testbed-Aufbaus **PHUKET** (Smart-Factory-Kontext). Die Architektur veranschaulicht das Nebeneinander etablierter SPS-Technik und moderner Software-Komponenten.

#### Der Ablauf im System:

##### Sicherheits- und Basisfunktionen auf der SPS:

Eine klassische Steuerung führt weiterhin zeitkritische Bewegungen aus und stellt die funktionale Sicherheit (Safety) auf Hardware- und Firmware-Ebene sicher.

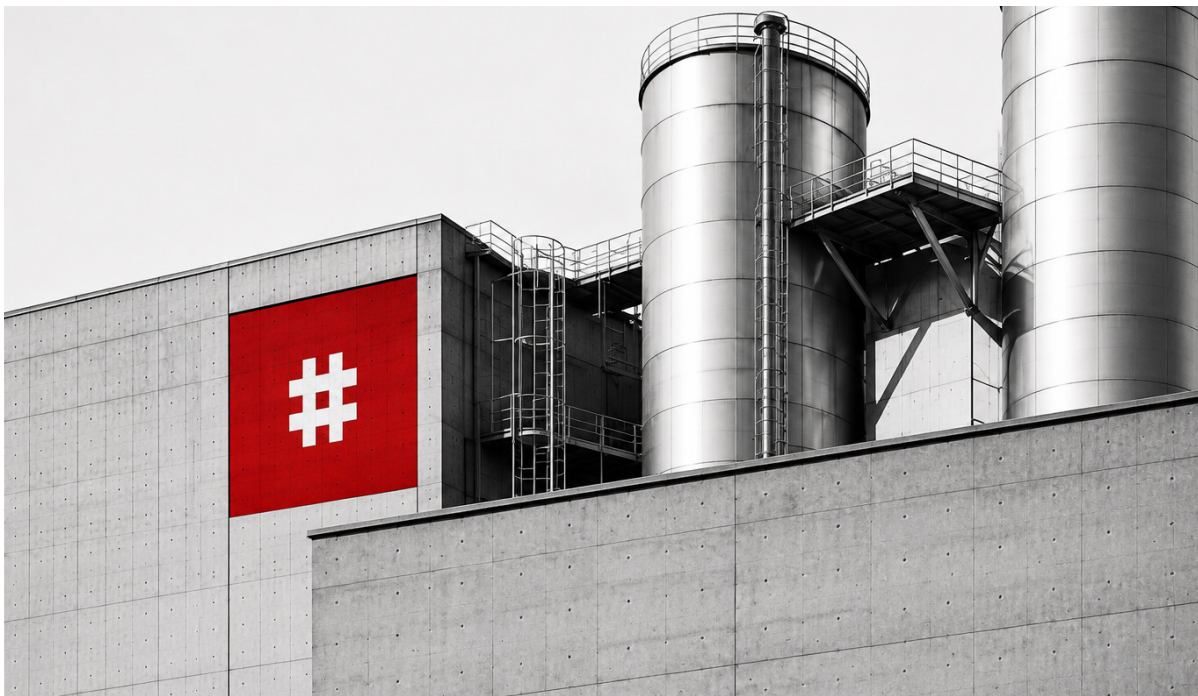
**Kapselung über Schnittstellen-Apps:** Feldbus-Daten und Steuerungsbefehle werden über leichte Schnittstellen-Applikationen abstrahiert und als aufrufbare Dienste bereitgestellt.

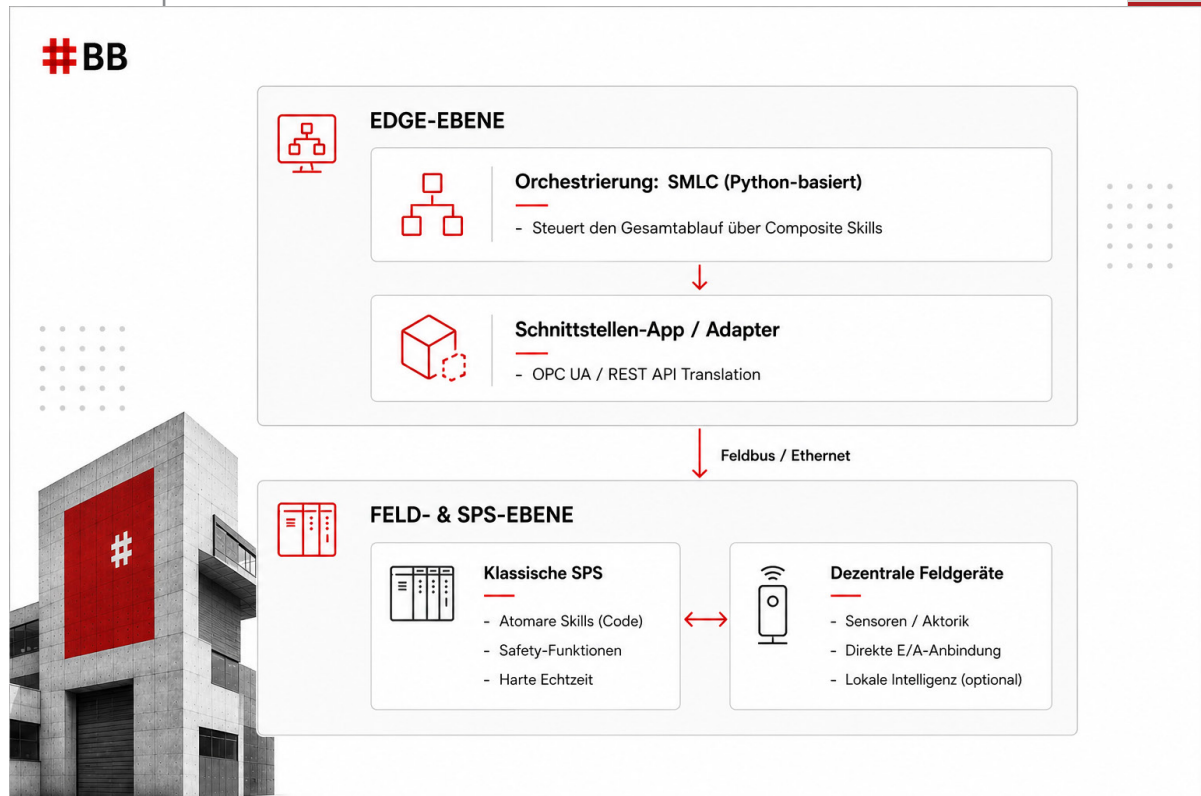
**Orchestrierung via SMLC:** Ein auf dem Edge-Gerät in Python umgesetzter Smart Manufacturing Level Controller (SMLC) übernimmt die Logik der atomaren und zusammengesetzten Skills. Er ruft die vorbereiteten Funktionen der SPS parametriert ab und koordiniert den Gesamtablauf.

**Dieser Aufbau zeigt: Es ist kein vollständiger Umbau der Automatisierungsebene erforderlich, um flexible Software-Abläufe in bestehenden Produktionsumgebungen zu etablieren.**

#### Wirtschaftlichkeitsbetrachtung und Business Case

Die Einführung einer Skill-basierten Architektur erfordert initiale *Investitionen* in Software-Entwicklung, Edge-Hardware und die Refaktorisierung von SPS-Code. *Den Aufwänden stehen jedoch konkrete Einsparpotenziale und Flexibilitätsgewinne gegenüber.*





### 1. Einmalige Aufwände (CapEx & Initial OpEx)

- **SPS-Refaktorisierung:** Aufwand für die Umstrukturierung bestehender IEC-61131-Programme in gekapselte Funktionsbausteine.
- **Edge-Infrastruktur:** Anschaffung und Installation von Edge-Knoten sowie Einteilung der Netzwerk-Zonen (OT-Sicherheit).
- **Entwicklung der Schnittstellen:** Aufbau wiederverwendbarer Skill-Module und Anbindung an übergeordnete IT-Systeme.

### 2. Wiederkehrender Nutzen (OpEx-Reduktion & Nutzenpotenziale)

- **Reduzierter Engineering-Aufwand bei Produktwechseln:** Neue Ablaufvarianten werden auf der Orchestrierungsebene konfiguriert. Eingriffe in den sicherheitsrelevanten SPS-Code entfallen in vielen Fällen, was Test- und Validierungszeiten um bis zu 60% [IK8.1] verkürzt.
- **Herstellerunabhängige Skalierung:** Durch die Standardisierung der Schnittstellen (z. B.

via OPC UA) lassen sich neue Anlagenteile verschiedener Hersteller ohne individuellen Schnittstellen-Code in die Orchestrierung einbinden.

- **Höhere Anlagenverfügbarkeit:** Fehler in höhergelagerten Software-Abläufen führen nicht zwangsläufig zum Stillstand der gesamten Steuerung. SPS-Safety-Funktionen bleiben unbeeinflusst.



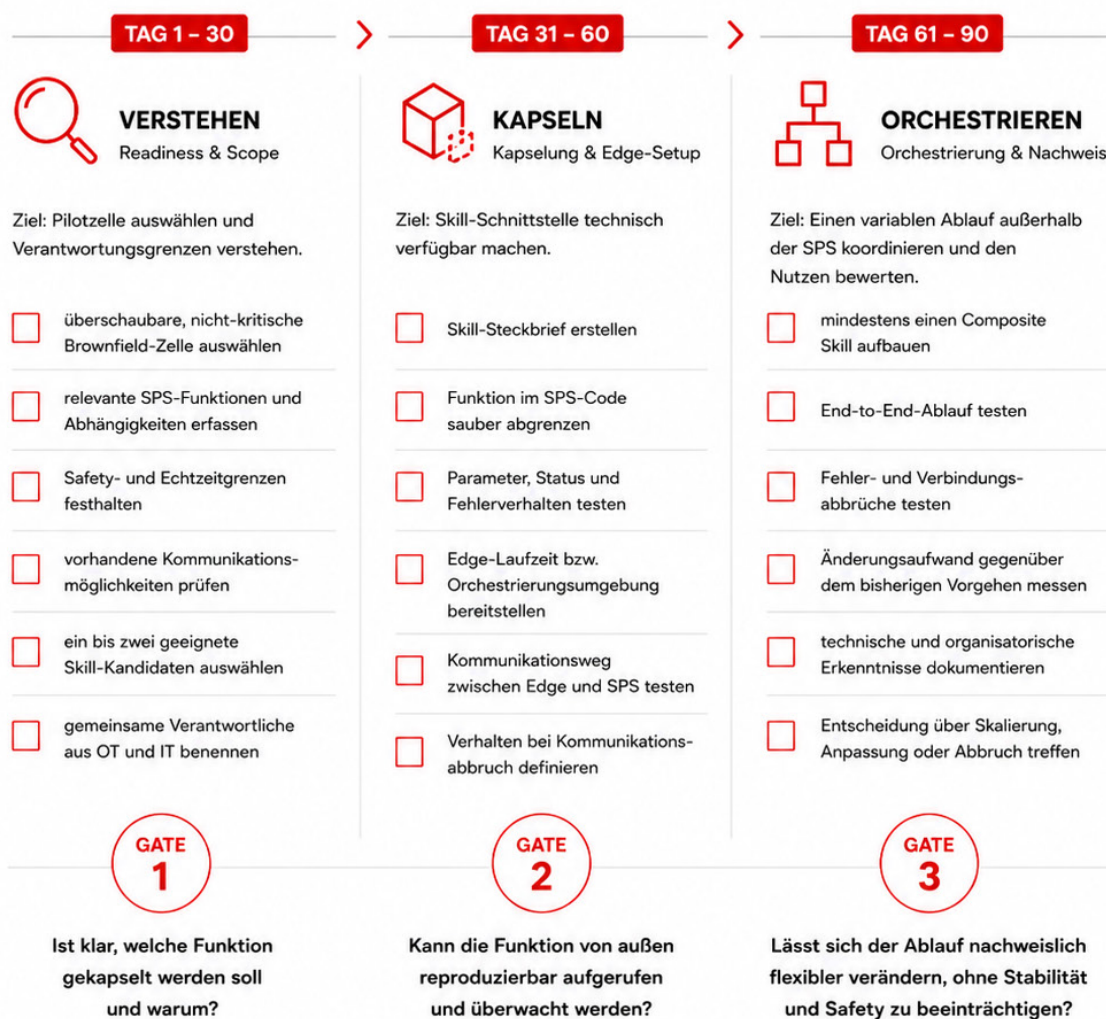
# Pilot-Checkliste

## 90 Tage

Die folgende Checkliste strukturiert die Umsetzung eines ersten Pilotprojekts im Brownfield über einen Zeitraum von drei Monaten.

Ziel ist es, den Nachweis der technischen Machbarkeit mit überschaubarem Risiko zu erbringen.

## #BB



### TOOL 4: PILOT-CHECKLISTE MIT NACHWEIS

| Aktivität                      | Owner | Termin | Nachweis | Status |
|--------------------------------|-------|--------|----------|--------|
| Pilotzelle abgegrenzt          |       |        |          |        |
| Skill definiert                |       |        |          |        |
| SPS-Funktion gekapselt         |       |        |          |        |
| Schnittstelle verfügbar        |       |        |          |        |
| Composite Skill läuft          |       |        |          |        |
| Kommunikationsausfall getestet |       |        |          |        |
| KPI-Vergleich durchgeführt     |       |        |          |        |
| Rollout-Entscheidung getroffen |       |        |          |        |

# Kapitel 6

## Fazit und Ausblick

### Evolution statt Revolution im Brownfield

Die Realisierung der Software-definierten Fabrik scheitert in der Praxis selten an mangelnder Vision.

**Eher noch scheitert sie am pragmatischen Umgang mit bestehenden Produktionsanlagen.**

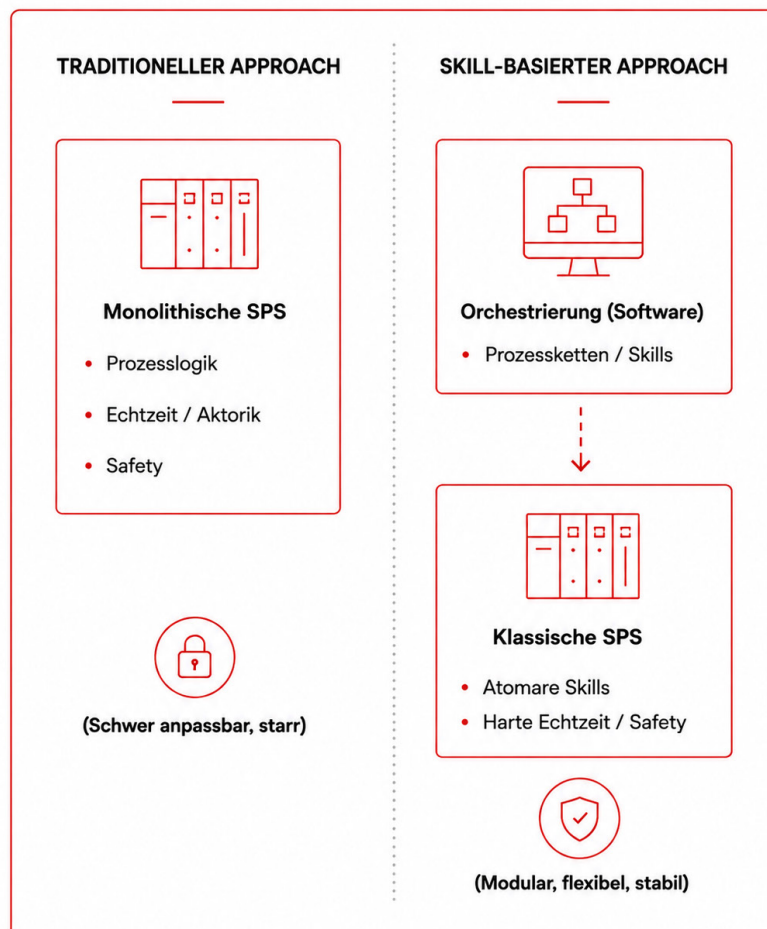
Der Versuch, gewachsene SPS-Strukturen radikal zu *ersetzen*, ignoriert die technischen und regulatorischen Realitäten des Brownfields.

*Harte Echtzeit, funktionale Sicherheit und der wirtschaftliche Bestandschutz bestehender Maschinen lassen sich nicht durch reine Software-Lösungen aufheben.*

Wie dieses Dokument gezeigt hat, liegt der tragfähige Weg nicht in der Ablösung der Steuerungstechnik, sondern in ihrer **funktionalen Neu-Eingliederung**.



### #BB



Die **Kapselung von SPS-Code** in atomare, wiederverwendbare Skills und die Verlagerung der übergeordneten Ablauflogik auf Edge-Plattformen schafft eine **klare Trennung der Verantwortlichkeiten**.

Die **SPS** fokussiert sich auf ihre **Stärken**: Deterministische Echtzeitverarbeitung, Aktorik-Ansteuerung und Safety-Funktionen.

Die **Software-Ebene** übernimmt die Steuerung der Prozessketten, die Anbindung an IT-Systeme und die dynamische Anpassung an neue Produktionsanforderungen.

### Empfohlene Nächste Schritte für Entscheider

Für Industrieunternehmen, die den Wandel zur modularen Produktion im Brownfield einleiten wollen, ergeben sich **drei zentrale Handlungsempfehlungen**:

1. *Standardisierte Schnittstellen definieren:* Investieren Sie frühzeitig in die Festlegung einheitlicher Skill-Interfaces (z. B. auf Basis von OPC UA und PackML), bevor Sie mit der Code-Umsortierung beginnen. Der Nutzen

einer entkoppelten Architektur steht und fällt mit der Qualität ihrer Schnittstellenverträge.

2. *Klein starten, aber architektonisch sauber:* Wählen Sie für ein 90-Tage-Pilotprojekt eine begrenzte, nicht-kritische Fertigungszelle. Ziel muss nicht die vollständige Umstellung des Werks sein, sondern der Nachweis, dass ein Skill-Aufruf von der Edge-Ebene auf die SPS stabil und nachvollziehbar funktioniert.
3. *Interdisziplinäre Teams bilden:* Der Übergang von der SPS-Programmierung zur Skill-Orchestrierung ist ein Kultur- und Methodenwechsel. Erfolgreiche Projekte bringen Automatisierungingenieure (OT) und Softwareentwickler (IT) von Tag 1 an einheitlichen Spezifikations-Tools wie dem Skill-Steckbrief zusammen.

**Die Software-definierte Fabrik beginnt nicht mit dem Abriss der Steuerungsebene. Sie beginnt mit der schrittweisen Kapselung ihrer Funktionen. Damit wird das Brownfield vom Innovationshemmnis zum soliden Fundament einer flexiblen Fertigung.**

