Diplomarbeit 2026



Titel: as1UpdateTool
Studiengang: Dipl. Techniker/in HF Applikationsentwicklung
Schule: TEKO Schweizerische Fachschule AG, Luzern
Student: Martin Plüss
Firma: Ammann Schweiz AG
Datum / Semester: 2026
Status: Nicht vertraulich

Inhaltsverzeichnis

- Erklärung zur Nutzung von Künstlicher Intelligenz
- 1. Management Summary
- 2. Kurzer beruflicher Lebenslauf
- 3. Qualifikationsprofil
- 4. Projektinitialisierung
    - 4.1 Pflichtenheft
    - 4.2 Zielscheibe
- 5. Projektplanung
    - 5.1 Vorgehensmodell
    - 5.2 Projektstrukturplan (PSP)
    - 5.3 Projektablaufplanung (Gantt-Diagramm / Soll)
    - 5.4 Kommunikationsplanung
    - 5.5 Risikoanalyse
- 6. Projektrealisierung
    - 6.1 Analyse
    - 6.2 Kreativitätsmethode
    - 6.3 Priorisierungsmethode
    - 6.4 Variantenbildung
    - 6.5 Evaluation der geeignetsten Variante
- 7. Ausarbeitung der präferierten Variante
    - 7.1 Softwareanforderungsprofil
    - 7.2 Phasenplan zur Umsetzung
    - 7.3 SWOT-Analyse
    - 7.4 Risiko-Analyse (Detail/Umsetzung)
    - 7.5 Kosten-Nutzen-Analyse
    - 7.6 Realisierung (Resultat)
- 8. Projektabschluss
    - 8.1 Projektüberwachung (Soll-Ist Abweichungen)
    - 8.2 Evaluation der Zielerreichung
    - 8.3 Reflexion Weg zum Ziel
    - 8.4 Lessons Learned
    - 8.5 Ausblick
- 9. Eigenständigkeitserklärung
- 10. Verzeichnisse
- 11. Anhang

- 11.1 Benutzerdokumentation: As1UpdateTool (CLI-Handbuch)
- 11.2 Testbericht 1: Unit-Tests der Kernlogik
- 11.3 Testbericht 2: Fehlerfall- und Abbruchtests
- 11.4 Testbericht 3: Realversuch auf der as1-Entwicklungsumgebung
- 11.5 Testbericht 4: Abnahmetest der Muss-Erfolgskriterien

Erklärung zur Nutzung von Künstlicher Intelligenz

Bei der Erstellung dieser Diplomarbeit wurden KI-Systeme auf Basis grosser Sprachmodelle unterstützend eingesetzt. Die Nutzung wird hier transparent deklariert.

Umfang der Nutzung:

- **Sprachliche Unterstützung:** Prüfung und Verbesserung von Rechtschreibung, Grammatik und Formulierung unter Einhaltung der Schweizer Orthographie.
- **Recherche-Unterstützung:** Unterstützung bei der Quellensuche sowie bei der Prüfung von Fakten und Verweisen.
- **Struktur- und Prüfunterstützung:** Aufbau und Konsistenz der Dokumentstruktur, Formatierung der Verzeichnisse und Überprüfung rechnerischer Zusammenhänge.
- **Textentwürfe:** Punktuelle Unterstützung bei der Formulierung einzelner Textpassagen auf Grundlage der vom Verfasser selbst erarbeiteten fachlichen Inhalte. Die generierten Texte wurden anschliessend fachlich geprüft und bei Bedarf überarbeitet.

Die fachliche Substanz der Arbeit (Ist-Analyse, Konzeption und Bewertung der Lösungsvarianten, Implementierung des Tools, Durchführung der Tests sowie Reflexion und Bewertung) stammt vollständig vom Verfasser. Sämtliche von KI-Systemen vorgeschlagene Inhalte wurden durch den Verfasser geprüft und in eigener Verantwortung übernommen.

---

1. Management Summary

Basissatz & Ausgangslage

Die Ammann Schweiz AG entwickelt und vertreibt weltweit kundenspezifisch angepasste Asphalt- und Betonmischanlagen. Jede Anlage wird mit der Steuerungssoftware **as1** betrieben, die pro Anlage rund 370 aktive Steuerungselemente, 29'800 Parameter und 5'500 elektrische Signale umfasst. Nach der Inbetriebnahme bestellen Kunden häufig Erweiterungen, Optimierungen oder zusätzliche Funktionen. Diese Anpassungen werden als kundenspezifische Updatepakete auf den bestehenden Anlagen installiert.

Die Erstellung dieser Pakete erfolgt heute grösstenteils manuell. Der Software Engineer ermittelt die Änderungen aus mehreren getrennten Quellen, bereitet sie für die Installation auf und verfasst eine individuelle Installationsanleitung. Dieser Prozess ist zeitaufwendig und fehleranfällig. Die Qualität der Anleitung hängt zudem vom jeweiligen Bearbeiter ab. Das bestehende Installationstool **as1HotfixInstaller** unterstützt bisher nur Standard-Updatepakete, kundenspezifische Pakete werden ohne definierte Richtlinie erstellt.

Vorgehen

Die Arbeit wurde nach einem 4-Phasen-Modell geführt (Initialisierung, Planung, Realisierung, Abschluss). Nach einer Ist-Analyse des bestehenden Prozesses und der Systemumgebung wurden mit einer Kreativitätsmethode drei Lösungsvarianten erarbeitet und in einer Nutzwertanalyse gegen die Nullvariante bewertet. Die Bewertung fiel zugunsten von Variante 2 aus: ein modulares Tool auf Basis von C#/.NET, das die Änderungen automatisch aus dem Vergleich zweier Systemzustände (Snapshots) ableitet.

Die Realisierung folgte der Priorität der Muss-Ziele und war in vier Meilensteine gegliedert. Die Kernlogik ist durch 77 Unit-Tests abgesichert. Ein Realversuch auf einer **as1**-Entwicklungsumgebung validierte die Schnittstellen zu den Fremdsystemen (Datenbank, 7-Zip, **as1HotfixInstaller**) und bestätigte die Installierbarkeit des erzeugten Pakets.

Ergebnisse

Das fertige as1UpdateTool erfüllt sämtliche Muss-Ziele. Es erkennt Änderungen an Parametern und aktiven Steuerungselementen automatisch, ist vollständig über die Kommandozeile bedienbar und erzeugt ohne manuelle Nachbearbeitung ein Paket im Format des **as1HotfixInstaller**. Parameteränderungen werden nicht über SQL-Skripte, sondern über eine ParamTransfer-Importdatei eingespielt, die der Installer nativ unterstützt. Unsichere Parameteränderungen werden erkannt, vom automatischen Import ausgeschlossen und im generierten ReadMe zur manuellen Prüfung aufgeführt. Das ReadMe dokumentiert Installation und Änderungen nachvollziehbar und ersetzt die bisher personengebundene Anleitung durch eine standardisierte Form.

Der Entwicklungsaufwand betrug rund 70 Stunden und entspricht damit ca. CHF 8'750. Die geschätzte jährliche Ersparnis liegt je nach Szenario zwischen CHF 11'250 und CHF 46'875. Die Amortisationszeit beträgt je nach Szenario 9.3 bis 2.2 Monate.

Ausblick

Mit dem As1UpdateTool steht der Ammann Schweiz AG ein Werkzeug zur Verfügung, das die Erstellung kundenspezifischer Updatepakete standardisiert, nachvollziehbar macht und von der Person des Erstellers entkoppelt. Die Schichtenarchitektur erlaubt die Nachrüstung einer grafischen Bedienoberfläche sowie der konzipierten SOLL- und Kann-Funktionen, ohne die Kernlogik anzupassen. Als weitere Entwicklungsschritte sind die Schema-Validierung der erzeugten `install.config`, der direkte Start des Tools aus der **as1** sowie die Auswertung der tatsächlich eingesparten Zeit im Tagesgeschäft vorgesehen.

---

2. Kurzer beruflicher Lebenslauf

Zeitraum	Station / Funktion	Firma / Institution
04.2022 - Heute	**Softwareengineer:** - Konfigurierung und Parametrisierung von Asphalt- und Betonsteuerungen für Kundenprojekte. - Kundenspezifische Anpassungen an den Asphalt- und Betonsteuerungen in C++ und TypeScript. - Unterstützung des Inbetriebnahmepersonal via Remote-Support. - Unterstützung des globalen Kundendiensts.	Ammann Schweiz AG, Langenthal
09.2021 - 02.2022	**Polymechaniker:** - Einrichten und Programmierung von 3 - achsigen Fräsmaschinen. - Herstellung von CAM-Programme für Fräsmaschinen.	KS-Technik, Hallwil
07.2021 - 07.2021	**Polymechaniker:** - Einrichten und Programmierung von 3 - achsigen Fräsmaschinen. - Herstellung von CAM-Programme für Fräsmaschinen.	Hametec AG, Buckten
04.2009 - 06.2021	**Polymechaniker:** - Einrichten und Programmierung von 3 - achsigen Fräsmaschinen. - Einrichten von Drahterodiermaschinen. - Herstellung von CAM-Programme für Fräs- und Erodiermaschinen. - Ausbildung von neuen Mitarbeitern an diversen Maschinen. - Herstellung von Laserunterlagen mit CNC-Lasermaschine. - Ausbildung von Polymechaniker EFZ Lehrlinge.	Unipress AG, Lupfig
04.2008 - 03.2009	**Polymechaniker:** - Einrichten und Programmierung von 3 - achsigen Fräsmaschinen. - Flachschleifen.	Stauffer AG, Fulenbach
11.2007 - 03.2008	Rekrutenschule als Waffenmechaniker.	
08.2007 - 10.2007	**Polymechaniker:** - Einrichten von 5 - achsigen Fräsmaschinen.	Franke Industrietechnik AG, Aarburg
08.2003 - 07.2007	Ausbildung zu Polymechaniker EFZ.	

3. Qualifikationsprofil

Martin Plüss
Wiesenstrasse 25
4900 Langenthal

Unternehmens- und Führungsprozesse gestalten und verantworten (A1)	- Projekte zur Steuerung von Asphaltmischanlagen wurden gemäss den internen Prozessvorgaben selbständig und effizient abgewickelt. - Die Qualität der abgewickelten Projekte wurden mit Hilfe einer internen Checkliste überprüft und sichergestellt. - Durch mein gezieltes Feedback konnte die Checkliste weiter optimiert und somit ein Beitrag zur kontinuierlichen Verbesserung der internen Prozesse geleistet werden.
Kommunikation situationsangepasst und wirkungsvoll gestalten (A2)	- Mitarbeiter und Vorgesetzte wurden regelmässig über den aktuellen Stand von Projekten oder Aufgaben auf dem Laufenden gehalten. - Bei auftretenden Problemen wurden die relevanten Personen proaktiv informiert und konstruktive Lösungsvorschläge präsentiert. Im direkten Kontakt mit Kunden wurden fachliche Fragen zur Software verständlich beantwortet, und es wurden individuelle Lösungen erarbeitet, um den spezifischen Anforderungen gerecht zu werden.
Die persönliche Entwicklung reflektieren und aktiv gestalten. (A3)	- Über den Onlinekurs auf Boot.dev wurde neues Wissen im Thema Backend-Entwicklung erlangt. - Mein erworbenes Wissen teilte ich aktiv mit meinen Kollegen durch informelle Wissensweitergabe oder Unterstützung bei fachlichen Fragen. - Neu erworbenes Wissen setzte ich gezielt in der Praxis ein, um Projekte effizienter zu gestalten und innovative Lösungen zu entwickeln.
Datenschutz und Datensicherheit gewährleisten. (B9)	- Bei meiner Arbeit hielt ich mich strikt an die geltenden Datenschutzrichtlinien, sowohl interne als auch externe, um die sichere Verarbeitung und Speicherung von Daten zu gewährleisten. - Ich stellte sicher, dass sensible Daten ausschliesslich autorisierten Personen zugänglich waren, indem ich die Zugriffsrechte regelmässig überprüfte und durchsetzte.
Softwarearchitektur analysieren und bestimmen (B10)	- Ich studierte die Softwarestruktur und das Design der Asphaltmischanlagensteuerungssoftware der Ammann Schweiz AG, um gezielte Anpassungen basierend auf individuellen Kundenwünschen zu ermöglichen. - Für spezifische Kundenerweiterungen erstellte ich technische Dokumentationen, die die Architektur der Software präzise beschreiben und als Grundlage für die Implementierung dienten.
Applikationen entwickeln, Programme erstellen und testen. (B11)	- Um die Anforderungen des Kunden an die Software zu erfüllen, wurden neue Klassen entwickelt oder bestehende Klassen angepasst. - Dabei arbeitete ich eng mit dem Kunden zusammen, um die gewünschte Lösung effizient und zielgerichtet umzusetzen.

Konzepte und Services entwickeln (B13)	- Die Anforderungen des Kunden an die Asphaltmischanlagensteuerung wurde von mir detailliert analysiert, um ein passendes und effizientes Konzept zu entwickeln. - Die erstellten Konzepte wurden durch gezieltes Feedback vom Kunden, Vorgesetzten und Kollegen kontinuierlich optimiert, um eine hohe Funktionalität und Nutzerzufriendenheit sicherzustellen.
Konzepte und Services umsetzten (B14)	- Basierend auf den erstellten Konzepten wurden neue Klassen und Module entwickelt, sowie Anpassungen an bestehenden Komponenten in der Asphaltmischanlagensteuerung vorgenommen. - Etwaige aufgetretene Fehler wurden systematisch analysiert und behoben, um eine fehlerfreie und zuverlässige Implementierung sicherzustellen.
Nachhaltiges Handeln (B15)	- Durch die Einhaltung der internen Energiesparrichtlinien gestalte ich meine Arbeitsumgebung ressourcenschonend und effizient. - Materialien wie Papier und Karton werden von mir aktiv recycelt, um die nachhaltigen Ziele des Unternehmens zu unterstützen.

4. Projektinitialisierung

4.1 Pflichtenheft

Ausgangslage / Situation

Die Ammann Schweiz AG entwickelt und vertreibt weltweit hochindividualisierbare Asphalt- und Betonmischanlagen. Jede Anlage wird mit der kundenspezifisch angepassten Steuerungssoftware **as1** ausgeliefert, die aus einer in C++ implementierten Steuerungslogik, einer in TypeScript implementierten Prozessvisualisierung und einer lokalen MSSQL-Datenbank zur persistenten Speicherung der Parameter besteht. Die Kommunikation zwischen Logik und Visualisierung erfolgt über ein bidirektionales Message-System. Die Anlagensteuerung wird pro Kunde über drei lokale XML-Konfigurationsdateien definiert: die `Fieldbus.config` (elektrische Module mit digitalen und analogen Ein-/Ausgängen), die `AutomationControl.config` (Logikelemente) und die `ProcessView.config` (Darstellungselemente). Jede Anlage umfasst dabei rund 370 aktive Steuerungselemente, 29'800 Parameter und 5'500 elektrische Signale.

Nach der Inbetriebnahme bestellen Kunden häufig Erweiterungen, Optimierungen oder zusätzliche Funktionen. Diese Anpassungen werden von Software Engineers in den aktuellen Softwarestand integriert und getestet. Anschliessend müssen sie auf bestehenden Kundensystemen installiert werden. Dafür wird pro Projekt ein individuelles Updatepaket erstellt. Ein solches Paket kann SQL-Skripte, XML-Dateien und **as1**-spezifische Standardsoftwareupdates enthalten.

Die Erstellung dieser Updatepakete erfolgt heute grösstenteils manuell. Der Engineer muss Änderungen an Parametern, Konfigurationen, Datenbanktabellen und aktiven Steuerungselementen identifizieren, dokumentieren und für die Installation vorbereiten. Dabei greift er auf mehrere, getrennte Informationsquellen zu: die Änderungshistorie der MSSQL-Datenbank (maximal drei Einträge pro Parameter, nur über die **as1** abrufbar), die XML-Konfigurationsdateien und den manuellen Parameterexport der **as1**. Zusätzlich erstellt er eine individuelle Installationsanleitung, damit das Update auch von anderen Mitarbeitenden installiert werden kann.

Für die Installation existiert mit dem **as1HotfixInstaller** bereits ein internes Werkzeug. Es installiert Pakete, die als selbstentpackendes 7-Zip-Archiv mit dem Präfix `Package-` und einer gültigen `install.config` aufgebaut sind. Es kann jedoch im Moment nur für Standard-Updatepakete verwendet werden. Kundenspezifische Pakete werden manuell und ohne definierte Richtlinie erstellt.

Problemstellung

Die manuelle Erstellung von Updatepaketen ist zeitaufwendig und fehleranfällig. Bei rund 29'800 Parametern und 5'500 elektrischen Signalen pro Anlage ist die Übersicht begrenzt: Einzelne Anpassungen können vergessen oder falsch übernommen werden. Im ungünstigsten Fall können fehlende, sicherheitsrelevante Parameter zu Sachschäden an der Mischanlage führen.

Die Qualität der Installationsanleitung hängt stark vom jeweiligen Bearbeiter ab, da Form, Detailgrad und Verständlichkeit der ReadMe-Datei nicht vorgegeben sind. Die Installation eines Updatepakets erfordert daher häufig Rückfragen an den ursprünglich verantwortlichen Software Engineer und ist personengebunden. Es existiert zwar bereits das interne Updateinstallationstool **as1HotfixInstaller**, dieses kann im Moment jedoch nur für Standard-Updatepakete verwendet werden.

Es besteht Bedarf nach einer Lösung, welche die Ermittlung der relevanten Änderungen automatisiert, die Erstellung von Updatepaketen vereinfacht und die Nachvollziehbarkeit der enthaltenen Änderungen verbessert.

Ziel

Erarbeitung des Konzepts und Entwicklung eines Tools, das die Erstellung von Updatepaketen für bestehenden Kundenanlagen automatisiert.

Deklaration zu Zielen und Erfolgskriterien

Die folgenden Ziele und Erfolgskriterien wurden vor der Durchführung der IST-Analyse (Kapitel 6.1) auf Basis der damals bekannten Informationen erarbeitet. Sie wurden als Rahmenbedingungen zur Erfolgsmessung mit dem Auftraggeber (Ammann Schweiz AG), der Schule (TEKO Luzern) und dem Diplomanden abgestimmt und bleiben unverändert. Die in der IST-Analyse gewonnenen Erkenntnisse (technische Details der **as1**, Funktionsweise des **as1HotfixInstaller**s, Kennzahlen) sind bewusst nicht rückwirkend in die Ziele eingeflossen, um diese Abstimmung nicht nachträglich zu verändern.

Endergebnis

Muss-Ziele

- Reduktion des manuellen Aufwands bei der Erstellung von Updatepaketen durch:
    - automatische Erkennung von Änderungen an Parametern zwischen zwei Softwarezeitpunkten
    - automatische Erkennung von Änderungen an aktiven Steuerungselementen
    - Bereitstellung einer Kommandozeilenschnittstelle zur Ausführung des Tools
    - automatische Erstellung eines für den HotfixInstaller geeigneten Updatepakets
- Reduktion der Fehleranfälligkeit bei der Erstellung und Installation von Updatepaketen durch:
    - standardisierte Erstellung des Updatepakets
    - automatische Erstellung einer Installations- und Änderungsdokumentation (ReadMe)

Kann-Ziele

- Automatische Erkennung von Änderungen an definierten Datenbankobjekten
- Vereinfachung der Bedienung durch automatische Erkennung oder Vorbelegung von Standardpfaden
- Erweiterte Dokumentation der erkannten Änderungen in strukturierter Form

Erfolgskriterien

Muss-Erfolgskriterien

- Das Tool erkennt Änderungen an Parametern zwischen zwei Softwarezeitpunkten automatisch und berücksichtigt diese im Updatepaket.
- Änderungen an aktiven Steuerungselementen werden automatisch erkannt und dokumentiert.
- Die erzeugten Updatepakete können mit dem HotfixInstaller ohne manuelle Nachbearbeitung installiert werden.
- Das automatisch generierte ReadMe dokumentiert die erkannten Änderungen sowie die notwendigen Installationsinformationen nachvollziehbar.

- Die Lösung funktioniert mit den **as1**-Softwareversionen V6.0 und V6.1.
- Das Tool kann vollständig über Kommandozeilenparameter ausgeführt werden.
- Die Lösung wird anhand mindestens eines bestehenden Kundenprojekts erfolgreich getestet und dokumentiert.
- Die technische Dokumentation beschreibt Aufbau, Funktionsweise und Einschränkungen der Lösung nachvollziehbar.

Kann-Erfolgskriterien

- Das Tool erkennt Änderungen an den definierten Datenbankobjekten automatisch und berücksichtigt diese im Updatepaket.
- Die passende Konfigurationsdatei wird automatisch gefunden oder ein geeignetes Standardverzeichnis vorgeschlagen.
- Das ReadMe enthält eine strukturierte Übersicht sämtlicher erkannter Änderungen.

4.2 Zielscheibe

Die Zielscheibe zeigt das Hauptziel im Zentrum, die Muss-Ziele im inneren Ring und die Kann-Ziele im äusseren Ring.

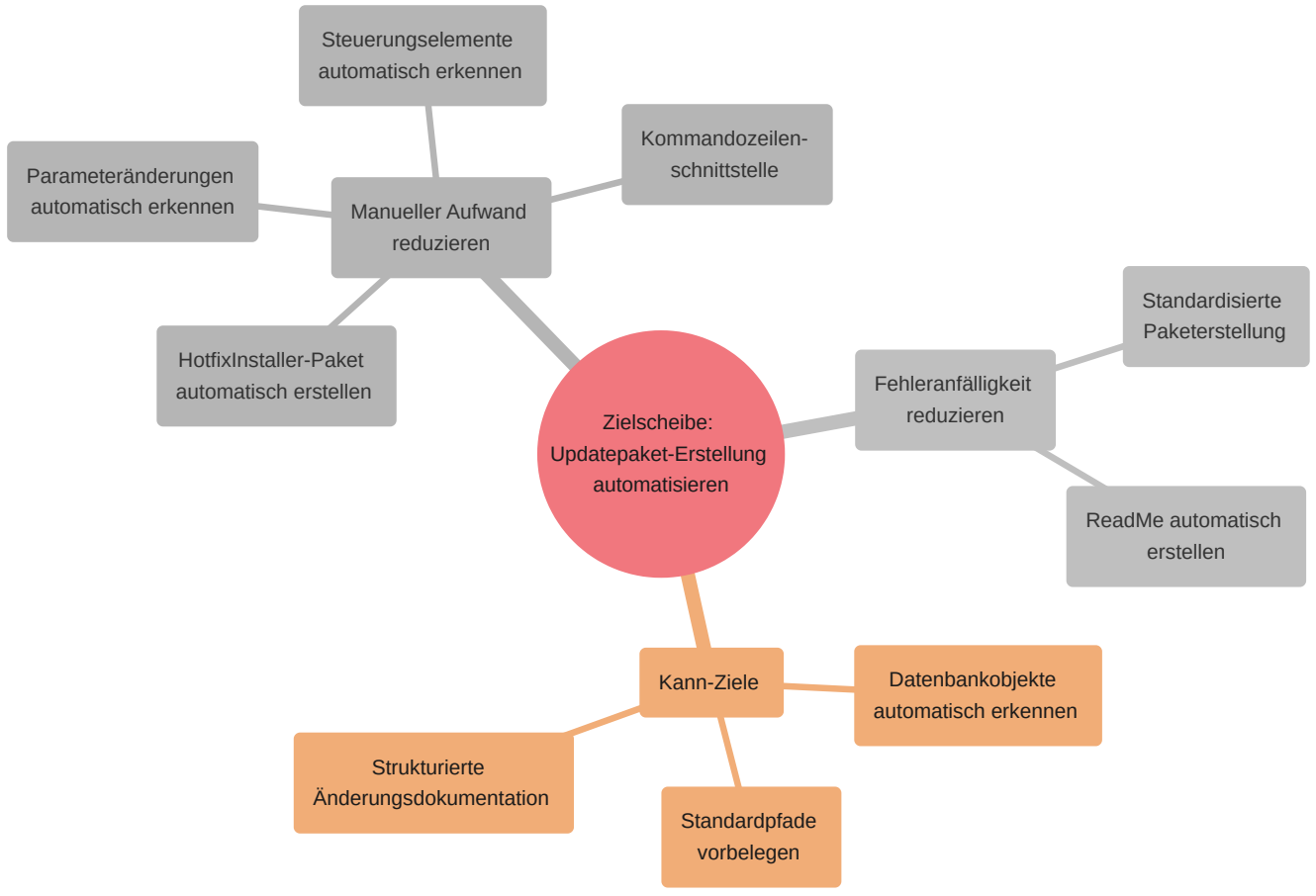

Abbildung 1: Zielscheibe der Diplomarbeit

5. Projektplanung

(Planung: Die folgenden Abschnitte entstanden vor Beginn der Realisierung und beschreiben den Soll-Zustand des Projektverlaufs.)

5.1 Vorgehensmodell

In diesem Projekt wurde nach dem 4-Phasen-Modell gearbeitet [4]. In diesem Modell wird das gesamte Projekt in vier Phasen unterteilt. Jede Phase liefert ein definiertes Ergebnis und wird mit einem Phasenentscheid abgeschlossen. Erst wenn dieses Ergebnis vorliegt, beginnt die nächste Phase. Damit wird sichergestellt, dass die nachfolgende Planung immer auf den Ergebnissen der vorherigen Phase aufbaut und nicht auf Annahmen.

Die erste Phase ist die Projektinitialisierung. In dieser Phase wird das Projekt auf den Weg gebracht. Es werden Fragen geklärt, die Ausgangssituation analysiert, Ziele definiert und am Ende wird der Projektvertrag unterzeichnet. Im vorliegenden Projekt wurde in dieser Phase die manuelle Erstellung von Updatepaketen bei der Ammann Schweiz AG analysiert (vgl. Kapitel 6.1) und daraus der Bedarf für ein automatisiertes Tool abgeleitet. Das Ergebnis dieser Phase ist das Pflichtenheft (Kapitel 4.1).

In der zweiten Phase erfolgt die Projektplanung. Hier werden unter anderem die Ressourcen-, Zeit- und Kommunikationsplanung erstellt. Für das vorliegende Projekt umfasst diese Planung den Projektstrukturplan, die Projektablaufplanung mit den vier Phasen und Meilensteinen des Tools (Kapitel 5.3), die Kommunikationsplanung (Kapitel 5.4) sowie die Risikoanalyse (Kapitel 5.5).

In der dritten Phase, der Realisierung, wird das Projekt umgesetzt. Im vorliegenden Projekt beinhaltet diese Phase die Analyse der bestehenden Prozesse, die Variantenbildung (Kapitel 6.4), die Ausarbeitung der geeignetsten Variante (Kapitel 7) sowie die Implementierung und das Testen des As1UpdateTools.

In der letzten Phase wird das Projekt abgeschlossen und reflektiert, um Fehler in Zukunft zu vermeiden. Dabei wird die Zielerreichung anhand der Erfolgskriterien des Pflichtenhefts ausgewertet (Kapitel 8.2) und der Projektverlauf mit der Planung verglichen (Kapitel 8.1).

Für dieses Projekt eignet sich das 4-Phasen-Modell insbesondere deshalb, weil die Diplomarbeit mit dem Pflichtenheft und den zu definierenden Zielen feste Rahmenbedingungen hat, die in der Initialisierung geklärt werden müssen, und weil der Zeitrahmen der Diplomarbeit nur eine begrenzte Anzahl von Planungs- und Korrekturschleifen zulässt.

5.2 Projektstrukturplan (PSP)

Im Rahmen der Projektstrukturplanung wird das Projekt in Arbeitspakete aufgeteilt, welche in der Summe die Endergebnisse des Projektes ergeben.

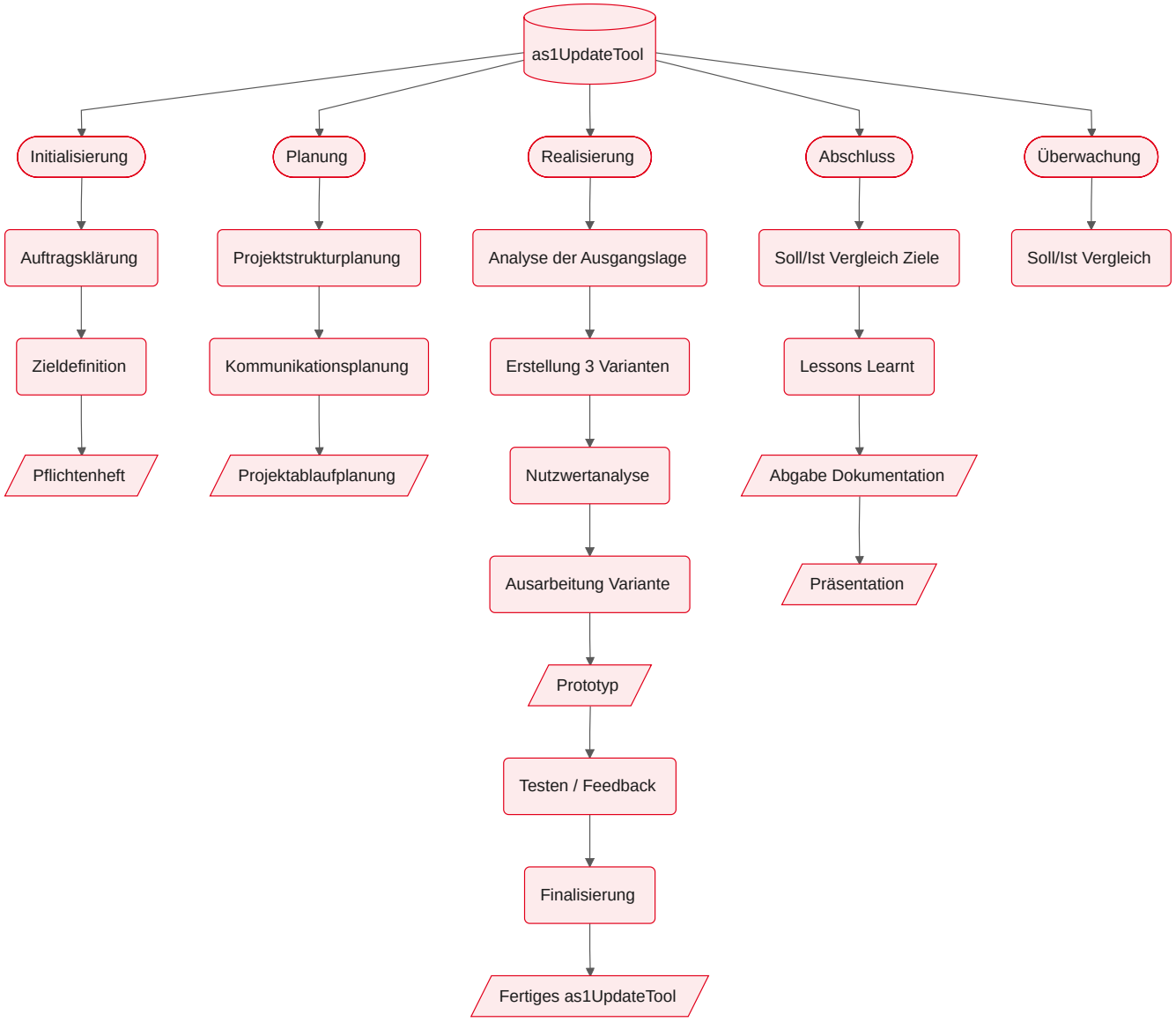

Abbildung 2: Projektstrukturplan (PSP)

5.3 Projektablaufplanung (Gantt-Diagramm / Soll)

Die Ablaufplanung ist als Gantt-Diagramm dargestellt. Sämtliche Diagramme dieser Arbeit wurden mit dem Diagramm-Werkzeug Mermaid [12] erstellt.

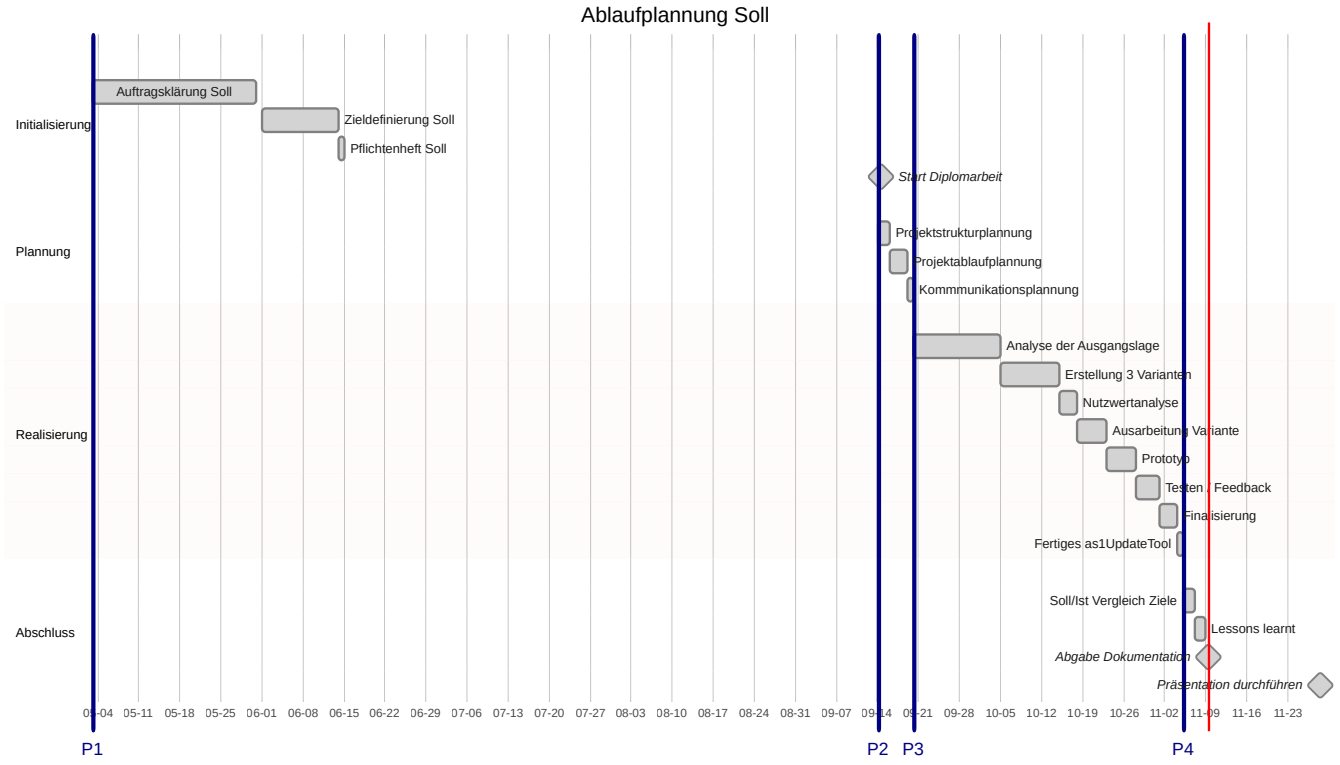

Abbildung 3: Projektablaufplanung (Soll)

5.4 Kommunikationsplanung

Im Rahmen der Diplomarbeit gibt es verschiedene Stakeholder, mit denen kommuniziert werden muss. Nicht alle benötigen die gleichen Informationen oder es werden unterschiedliche Kommunikationskanäle genutzt. Um sicherzustellen, dass alle Stakeholder ausreichend informiert sind um bei Bedarf rechtzeitig Feedback einholen zu können, ist eine gut durchdachte Kommunikationsplanung unabdingbar.

5.4.1 Stakeholder

Tabelle 1: Stakeholder der Kommunikationsplanung

Ansprechgruppe	Informationsbedarf	Inhalt	Periodizität / Termine	Kanal	Verantwortung
Auftragsgeber	Aktueller Projektstand	Aktueller Stand, Probleme, nächste Schritte	Einmal pro Monat	Statusbericht via Teams	Der Diplomand bereitet das Meeting vor und führt es durch.
Diplomlehrer	Aktueller Projektstand	Aktueller Stand, Probleme, nächste Schritte	Zweimal pro Diplomarbeit	Statusbericht via Teams	Der Diplomlehrer lädt den Diplomanden zu einem Termin ein.
Unabhängiger Experte	Kurzübersicht Diplomarbeit	Nutzen, Funktion, Zielerreichung und Demo der Diplomarbeit	Einmal pro Diplomarbeit	Präsentation in Luzern	Der Diplomand bereitet die Präsentation vor und präsentiert diese an dem vorgegebenen Termin.

Ansprechgruppe	Informationsbedarf	Inhalt	Periodizität / Termine	Kanal	Verantwortung
TEKO Luzern Sekretariat	Komplette Dokumentation der Diplomarbeit	Komplette Dokumentation der Diplomarbeit	Am Ende der Diplomarbeit	E-Mail	Der Diplomand ist dafür verantwortlich, dass die Abgabe am definierten Abgabedatum dem Sekretariat zugesendet wird.

Die Tabelle zeigt, dass sich der Informationsbedarf der Stakeholder klar unterscheidet und deshalb nicht für alle die gleiche Kommunikationsform geeignet ist. Der Auftragsgeber und der Diplomlehrer erhalten den aktuellen Projektstand in regelmässigen Abständen, damit sie den Projektverlauf verfolgen und bei Abweichungen frühzeitig Feedback geben können. Der Unterschied liegt dabei in der Periodizität: Der Auftragsgeber wird monatlich informiert, da er als Fachexperte der Ammann Schweiz AG in laufende Fachfragen eingebunden ist, während der Diplomlehrer zu zwei festgelegten Zeitpunkten den Stand beurteilt. Der unabhängige Experte erhält die Informationen einmalig in Form einer Präsentation, da für ihn primär Nutzen, Funktion und Zielerreichung der Arbeit im Vordergrund stehen und nicht der Projektverlauf. Das Sekretariat der TEKO Luzern benötigt ausschliesslich die finale Dokumentation am Abgabedatum, eine Begleitung des Projektverlaufs ist hier nicht vorgesehen. In allen Fällen liegt die Vorbereitung der Kommunikation beim Diplomanden, mit Ausnahme des Termins mit dem Diplomlehrer, der diesen einlädt.

5.5 Risikoanalyse

5.5.1 Vorgehen

Ein Risiko wird als Ursache-Wirkungs-Kette formuliert [6]: Ein Umstand (Ursache) führt zu einem Ereignis, das dem Projekt oder dem Ergebnis schadet (Wirkung). Jedes Risiko wird nach seiner **Eintrittswahrscheinlichkeit** und seinem **Schadensausmass** bewertet. Die Bewertung erfolgt je auf einer dreistufigen Skala (gering / mittel / hoch). Aus beiden Bewertungen ergibt sich die Risikoeinstufung:

Tabelle 2: Bewertungsmatrix Eintrittswahrscheinlichkeit / Schadensausmass

Wahrscheinlichkeit \ Ausmass	gering	mittel	hoch
gering	gering	mittel	mittel
mittel	gering*	mittel	hoch
hoch	mittel	hoch	hoch

() Bei geringer Eintrittswahrscheinlichkeit und geringem Schaden wird das Risiko als gering eingestuft.*

Für jedes Risiko mit Einstufung **mittel** oder **hoch** wird eine Gegenmassnahme definiert. Gegenmassnahmen wirken entweder präventiv (Reduktion der Eintrittswahrscheinlichkeit) oder kurativ (Reduktion des Schadensausmasses).

Die folgende Darstellung zeigt die eingestuften Risiken aus Abschnitt 5.5.2 als Kurzform (R1-R7) in der Bewertungsmatrix. Die Achsen entsprechen der dreistufigen Skala für Eintrittswahrscheinlichkeit und Schadensausmass. Die vier Quadranten zeigen die daraus resultierende Einstufung. Die ausführliche Beschreibung der Risiken steht in der Risikotabelle in Abschnitt 5.5.2.

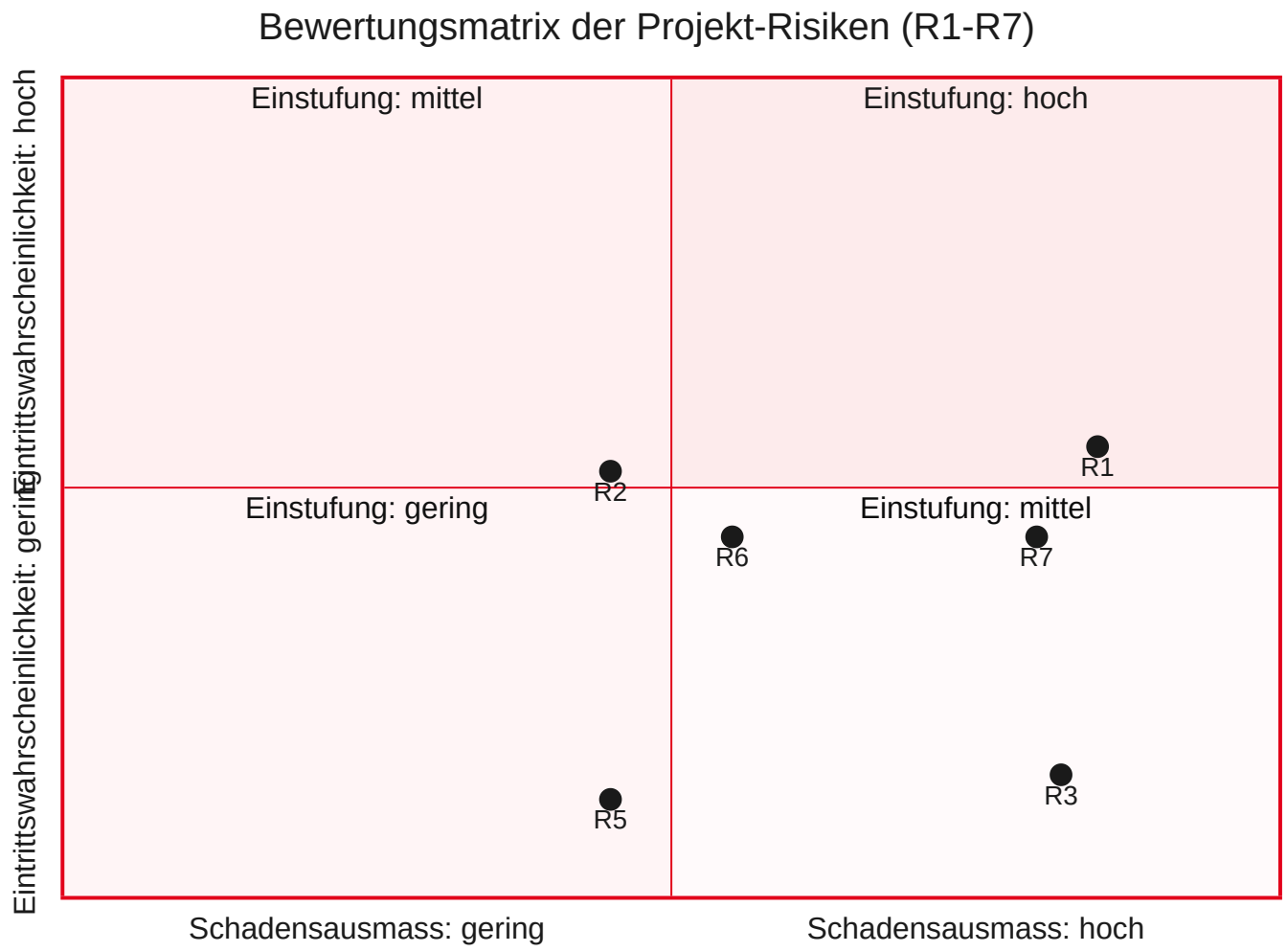

Abbildung 4: Bewertungsmatrix der Projekt-Risiken (R1-R7)

5.5.2 Risikotabelle

Tabelle 3: Risikotabelle der Projekt-Risikoanalyse

Nr.	Risiko (Ursache -> Wirkung)	Wahrscheinlichkeit	Schadensausmass	Einstufung	Gegenmassnahme
R1	Die in der Ist-Analyse erhobenen Erkenntnisse (Änderungshistorie, XML-Aufbau, Funktionsweise des **as1HotfixInstaller**s) weichen von den bisherigen Annahmen ab. -> Das Konzept muss nach der Ist-Analyse angepasst werden, der Zeitplan gerät unter Druck.	mittel	hoch	**hoch**	Zeitpuffer im Projektplan einplanen. Konzeptentscheide bewusst erst nach Abschluss der Ist-Analyse fällen. Gestaffelte Realisierung: Kernfunktionen (Erkennung, Paketierung) vor Komfortfunktionen.
R2	Das benötigte Detailwissen ist bei Ammann intern nur bei wenigen Personen verfügbar. -> Abhängigkeit von einzelnen Ansprechpartnern, Verzögerungen bei Rückfragen.	mittel	mittel	**mittel**	Terminologie und Fachbegriffe dokumentieren. Erkenntnisse aus Rückfragen unmittelbar verschriftlichen, damit das Wissen nicht personengebunden bleibt. Mehrere Ansprechpartner identifizieren.
R3	Kein Testsystem bzw. keine Testumgebung für ein bestehendes Kundenprojekt verfügbar. -> Das Muss-Erfolgskriterium "Test anhand mindestens eines bestehenden Kundenprojekts" kann nicht erfüllt werden.	gering	hoch	**mittel**	Testumgebung und Testprojekt früh im Projekt abklären und reservieren. Alternativ: Test mit einer lokalen Entwicklungs-Instanz der **as1** und anonymisierten Kundendaten vorbereiten.
R5	Die **as1**-Erweiterung für den direkten Tool-Aufruf liegt nicht im Machtbereich der Diplomarbeit. -> Die Lösung kann nicht vollständig wie konzipiert realisiert werden.	gering	mittel	**gering**	Die Systemgrenzen (vgl. Ist-Analyse) definieren die **as1**-Erweiterung als ausserhalb des Projektumfangs. Das Tool ist vollständig über die Kommandozeile bedienbar (Muss-Ziel 1c), ein **as1**-Startzeilenaufruf ist optional und kann nachgeliefert werden.
R6	Der Zugriff auf die Änderungshistorie der MSSQL-Datenbank erfordert Berechtigungen, die noch nicht geklärt sind. -> Die Differenzerkennung der Parameter kann nicht wie vorgesehen umgesetzt werden.	mittel	mittel	**mittel**	Berechtigungen und Zugriffsweg früh mit den zuständigen Stellen von Ammann klären. Nur lesende Zugriffe vorgesehen, gekapselt in einer eigenen Datenschicht.
R7	Der Umfang der Realisierung (GUI, Datenschicht, Tests, Dokumentation) wird unterschätzt. -> Der Zeitrahmen der Diplomarbeit wird gesprengt.	mittel	hoch	**hoch**	Realisierung nach den Muss-Zielen priorisieren. Meilensteine mit Zwischenständen definieren. Bei Zeitknappheit: Ausbau der GUI zurückstufen, da das Tool vollständig kommandozeilenfähig ist und die Muss-Ziele unabhängig von der GUI erreichbar sind.

5.5.3 Interpretation

Die Analyse zeigt zwei Risiken mit hoher Einstufung (R1, R7). Beide betreffen den Zeitrahmen des Projekts und werden mit dem gleichen Grundprinzip behandelt: der Realisierung nach der Priorität der Muss-Ziele. Die drei Risiken mit mittlerer Einstufung (R2, R3, R6) betreffen Abhängigkeiten von Ammann-internen Ressourcen. Diese können durch frühzeitige Abklärungen zu Projektbeginn reduziert werden. Das einzige Risiko mit geringer Einstufung (R5) ist durch die Definition der Systemgrenzen bereits strukturell entschärft.

Keines der identifizierten Risiken gefährdet das Grundziel der Arbeit. Die grössten Risiken sind durch Planung steuerbar. Die wichtigsten Handlungen fallen an den Projektstart an: Testumgebung klären (R3), Datenbankberechtigungen klären (R6) und Ansprechpartner definieren (R2).

---

6. Projektrealisierung

(Dieses Kapitel beschreibt die Analyse- und Konzeptphase der Arbeit: den Ist-Zustand des bestehenden Prozesses (6.1) sowie die daraus erarbeiteten Lösungsvarianten und deren Bewertung (6.2-6.5). Keine der Varianten ist zu diesem Zeitpunkt realisiert.)

6.1 IST-Analyse (Informationen & bestehende Lösungen)

*(Ist-Zustand: die bestehende Software **as1** und der heutige, manuelle Prozess der Update-Erstellung.)*

Dieses Kapitel ist thematisch gegliedert. Kapitel 6.1.1 beschreibt die technischen Grundlagen der Steuerungssoftware **as1**. Kapitel 6.1.2 beschreibt den bestehenden Prozess der Erstellung von Updatepaketen. Kapitel 6.1.3 analysiert Aufwand, Fehlerquellen und Risiken dieses Prozesses, einschliesslich ihrer Ursachen. Kapitel 6.1.4 beschreibt das bestehende Installationstool. Kapitel 6.1.5 fasst die Schwachstellen zusammen und leitet daraus die Schlussfolgerungen für das neue Tool ab. Kapitel 6.1.6 definiert dessen Systemgrenzen.

6.1.1 Ausgangslage und Unternehmenskontext

Unternehmen

Die Ammann Schweiz AG ist ein weltweit führender Anbieter von Asphalt- und Betonmischanlagen, Maschinen und Dienstleistungen für die Bauindustrie mit Kernkompetenz im Strassenbau und der Verkehrsinfrastruktur. Das Unternehmen wurde 1869 in Langenthal gegründet, ist bis heute im Familienbesitz und beschäftigt weltweit rund 3'600 Mitarbeitende. Produktionsstätten bestehen in sieben Ländern: Brasilien, China, Deutschland, Indien, Schweiz und Tschechien.

Die Mischanlage im Überblick

Da die vorliegende Arbeit die Steuerungssoftware einer Mischanlage betrachtet, wird zunächst kurz der Aufbau einer solchen Anlage erläutert. Eine Mischanlage ist eine stationäre Anlage von rund 30 Metern Höhe, in der die Ausgangsstoffe zu Transportbeton (Betonmischanlage) bzw. zu Asphalt (Asphaltmischanlage) vermischt werden. Die wichtigsten Bestandteile sind:

- **Mischturm:** Das Herzstück der Anlage. Hier werden die einzelnen Komponenten (z.B. Kies, Sand, Zement bzw. Bitumen) im Mischer zu dem fertigen Gemisch vermischt.
- **Komponentenzugaben:** Je Komponente eine eigene Einheit, in der die jeweilige Komponente gelagert, dosiert und danach in den Mischer entleert wird.
- **Verladesilos:** Liegen unter dem Mischturm und lagern das fertige Gemisch, damit dieses direkt auf Lastwagen verladen werden kann. Es sind üblicherweise mehrere Verladesilos vorhanden, damit nacheinander unterschiedliche Gemischsorten bereitgestellt werden können.
- **Gesteinstrocknung:** Trocknet den zugeführten Kies, siebt ihn und lagert ihn anschliessend.

Die Software steht dabei in direkter Verbindung mit der physischen Anlage. Jeder Bestandteil der Anlage besitzt elektrische Module (Klappen, Pumpen, Motoren, Waagen, Sensoren), die über die Feldebene von der Steuerungssoftware **as1** angesteuert und überwacht werden [1]. Die Grösse der Anlage und die Anzahl der beteiligten Komponenten erklären die Dimensionen der Software. Mit rund 370 aktiven Steuerungselementen, 29'800 Parametern und 5'500 elektrischen Signalen pro Anlage [3] bildet die **as1** die komplette Anlage einschliesslich aller Komponentenzugaben, der Trocknung und des Misch- und Verladevorgangs ab.

Asphaltmischanlage von aussen

Abbildung 5: Asphaltmischanlage von aussen zur Veranschaulichung der Anlagengrösse

Aufbau und Technologien der Steuerungssoftware **as1**

Für die Steuerung von Mischanlagen setzt die Ammann Schweiz AG die selbst entwickelte, modular aufgebaute Steuerungssoftware **as1** ein. Sie läuft auf dem Windows-Betriebssystem und besteht aus diesen Komponenten.

*Tabelle 4: Komponenten der **as1***

Komponente	Technologie	Funktion
Steuerungslogik	C++	Bildet die mechanischen und elektrischen Funktionen der Anlage ab und kommuniziert in Echtzeit mit den elektrischen Modulen.
Prozessvisualisierung (GUI)	TypeScript	Zeigt dem Benutzer die Anlage an und nimmt seine Eingaben entgegen.
Datenhaltung (lokal)	MSSQL-Datenbank, XML-Konfigurationsdateien	Speichert die Paramterwerte, Zuordnung der elektrischen Ein-/Ausgänge und Produktionsdaten dauerhaft.
Feldebene	Profinet über Ethernet	Verbindet die Steuerung mit den elektrischen Modulen der Anlage.

Internetzugang ist ausschliesslich für die Fernwartung und Softwareupdates erforderlich. Die Software wird an den Standorten Schweiz, Indien, Deutschland und China auf Kundenbestellungen konfiguriert und am Standort Schweiz entwickelt.

Verbindung von der Software zur Feldebene: Die **as1** läuft auf einem separaten Computer mit Windows-Betriebssystem im Steuerungsraum der Anlage. Dessen Ethernet-Anschluss ist über Ethernetkabel mit den elektrischen Modulen (E/A-Baugruppen) der Anlage verbunden. Über diese physische Verbindung setzt das Industrieprotokoll **Profinet** auf, ein auf Ethernet basierender Kommunikationsstandard der Automatisierungstechnik.

Die elektrischen Module verbinden die Software mit der Anlage in beide Richtungen. **Sensoren** melden der Steuerungslogik den Zustand der Anlage (z.B. ob eine Klappe offen oder geschlossen ist, welche Endlage ein Antrieb erreicht hat oder welches Gewicht auf einer Waage liegt). **Aktoren** sind die Stellglieder der Anlage, die die Steuerungslogik aktiv ansteuert (z.B. Motoren, Klappen und Ventile). Die **as1** bildet damit die komplette Anlage ab. Sie liest kontinuierlich die Sensorwerte ein und steuert darauf aufbauend die Aktoren an.

Die folgende Abbildung zeigt den grundsätzlichen Aufbau der **as1**.

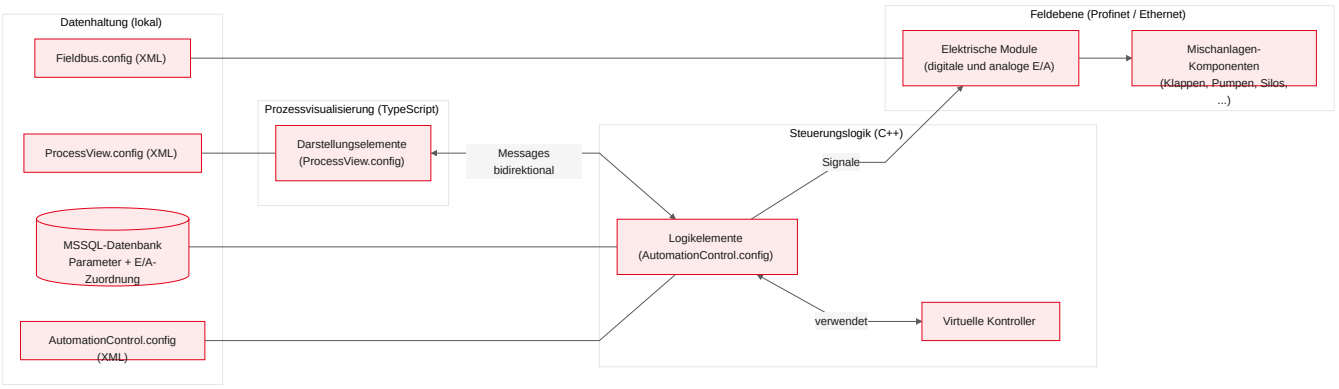

Abbildung 6: Grundsätzlicher Aufbau der as1 mit ihren Schichten

Steuerungselemente

Die **as1** wird für jede Anlage individuell zusammengestellt. Sie besteht aus Standard-Elementen und aus auftragsspezifisch angepassten Elementen. Das Grundbaustein-Modell ist bei allen Elementen gleich:

- **Logikelement (C++):** Bildet ein Bauteil der Mischanlage ab. Zum Beispiel eine Klappe, eine Pumpe, ein Silo oder eine Messstelle. Es verwaltet das Ansteuern des Bauteils, seinen Status und seine elektrische Signale.
- **Virtueller Kontroller (C++):** Bildet kein physisches Bauteil ab, sondern steuert und überwacht das Zusammenspiel mehrerer Logikelemente. Ein Kontroller legt fest, wann beispielsweise eine Klappe offen oder geschlossen sein muss, und teilt dies dem betroffenen Logikelement mit.
- **Darstellungselement (TypeScript):** Zeigt dem Benutzer den Zustand eines Logikelements an und ermöglicht Eingriffe in die Steuerung.

Jedes Logikelement hat genau ein zugehöriges Darstellungselement. Das Paar aus Logik- und Darstellungselement wird als **Steuerungselement** bezeichnet.

Die Kommunikation zwischen Logik und Darstellung läuft über Messages. Eine Seite sendet eine Message, die andere fängt sie ab und verarbeitet sie. Der Messageverkehr funktioniert in beide Richtungen. Statusänderungen der Anlage laufen von der Logik zur Darstellung. Benutzereingriffe von der Darstellung zur Logik.

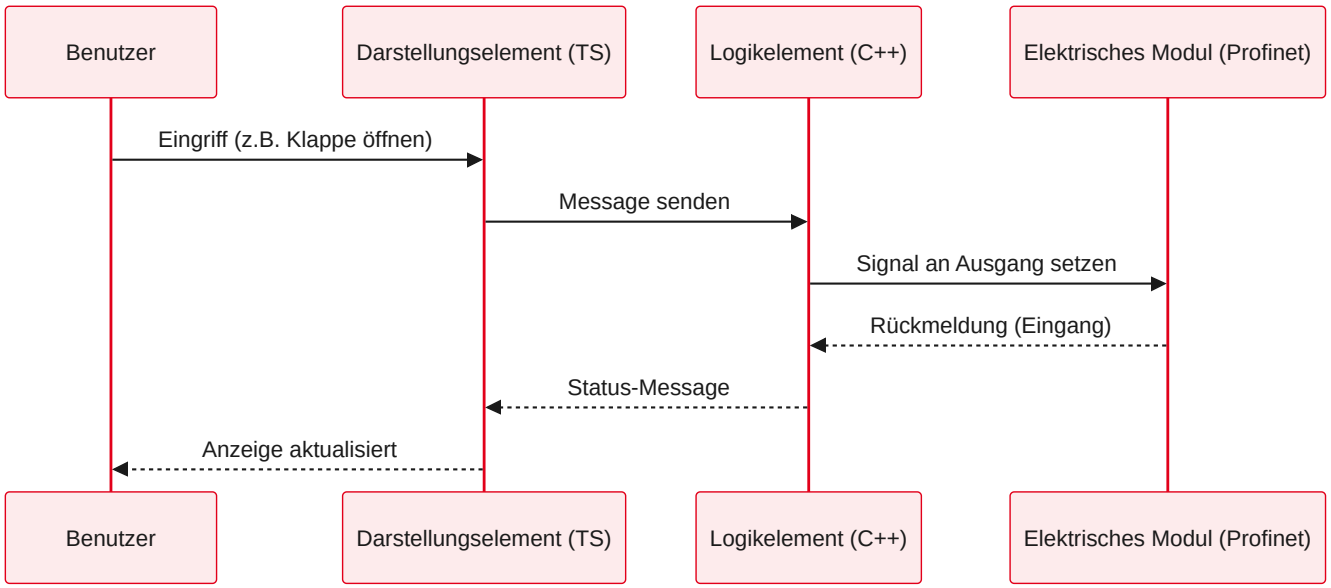

Abbildung 7: Message-Austausch zwischen Bedienung, Darstellung, Logik und Feldebene

Das Zusammenspiel von Logikelementen und virtuellen Kontrollern zeigt das folgende Beispiel aus der Praxis: ein Kontroller, der das Dosieren von Mineral in eine Waage und das anschliessende Entleeren in den Mischer steuert.

Dazu werden im Kontroller die Entleerklappen der verschiedenen Mineralsilos, die Mineralwaage und der Mischer über Parameter miteinander verknüpft. Der Kontroller vergibt Vorgaben an die Logikelemente (z.B. Entleerklappe auf/zu) und erhält von ihnen Statuswerte zurück (z.B. Ist-Gewicht der Waage, Endlage einer Klappe).

Der Kontroller durchläuft dabei eine fest definierte Zustandsfolge: Bereit zum Dosieren, Grobdosieren, Feindosieren, Toleranzkontrolle, Bereit zum Entleeren und Entleeren. In jedem Zustand überprüft der Kontroller alle verknüpften Elemente darauf, ob sie sich im erwarteten Zustand befinden. Beispielsweise erwartet der Kontroller während des Entleerens, dass die Entleerklappe der Waage offen ist. Entspricht ein verknüpftes Element nicht dem erwarteten Zustand, wechselt der Kontroller in den Fehlerzustand. Dieser kann nur durch Quittierung des Bedienpersonals verlassen werden.

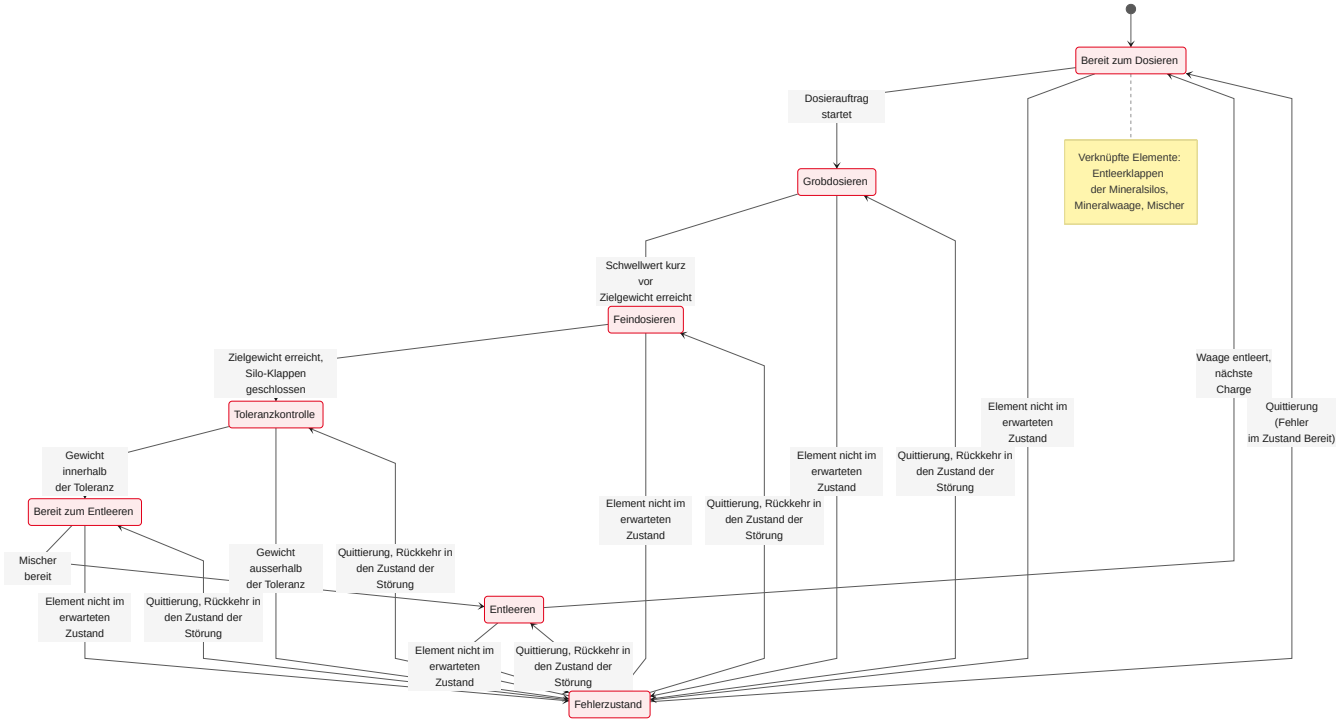

Abbildung 8: Zustandsdiagramm eines virtuellen Kontrollers am Beispiel der Mineral-Dosierung

Parametrisierung

Jedes Steuerelement wird über **Parameter** an seine Funktion angepasst. Das gleiche Logikelement deckt damit unterschiedliche Einsatzszenarien ab. Hier zwei einfache Beispiele:

- **Klappe:** Aktiv angesteuert oder nur überwacht.
- **Pumpe:** Nur eine Drehrichtung, vorwärts und rückwarts, mit oder ohne variable Drehzahl.

Die Parameter und die Zuordnung der elektrischen Ein-/Ausgänge werden in der lokalen MSSQL-Datenbank gespeichert.

Konfigurationsdateien

Weil jede Mischanlage elektrisch und mechanisch nach Kundenwunsch gebaut wird, muss auch die Steuerung pro Anlage angepasst werden. Die Anpassung erfolgt über drei lokale XML-Dateien. Die zu verwendenden Elemente werden anhand von Anlagenschemas sowie den elektrischen Ein- und Ausgängen ermittelt.

*Tabelle 5: XML-Konfigurationsdateien der **as1***

Datei	Inhalt	Abhängigkeit
Fieldbus.config	Elektrische Module	Elektrischer Aufbau der Anlage
AutomationControl.config	Aktive Logikelemente und Kontroller	Mechanischer Aufbau und Prozesslogik der Anlage
ProcessView.config	Darstellungselemente	Logikelemente

Alle drei Dateien sind für jede Anlage einzigartig und müssen pro Anlage erstellt bzw. angepasst werden. Die ProcessView.config hängt von der AutomationControl.config ab. Änderungen an einem Logikelement erfordern in der Regel auch eine Anpassung des Darstellungselement.

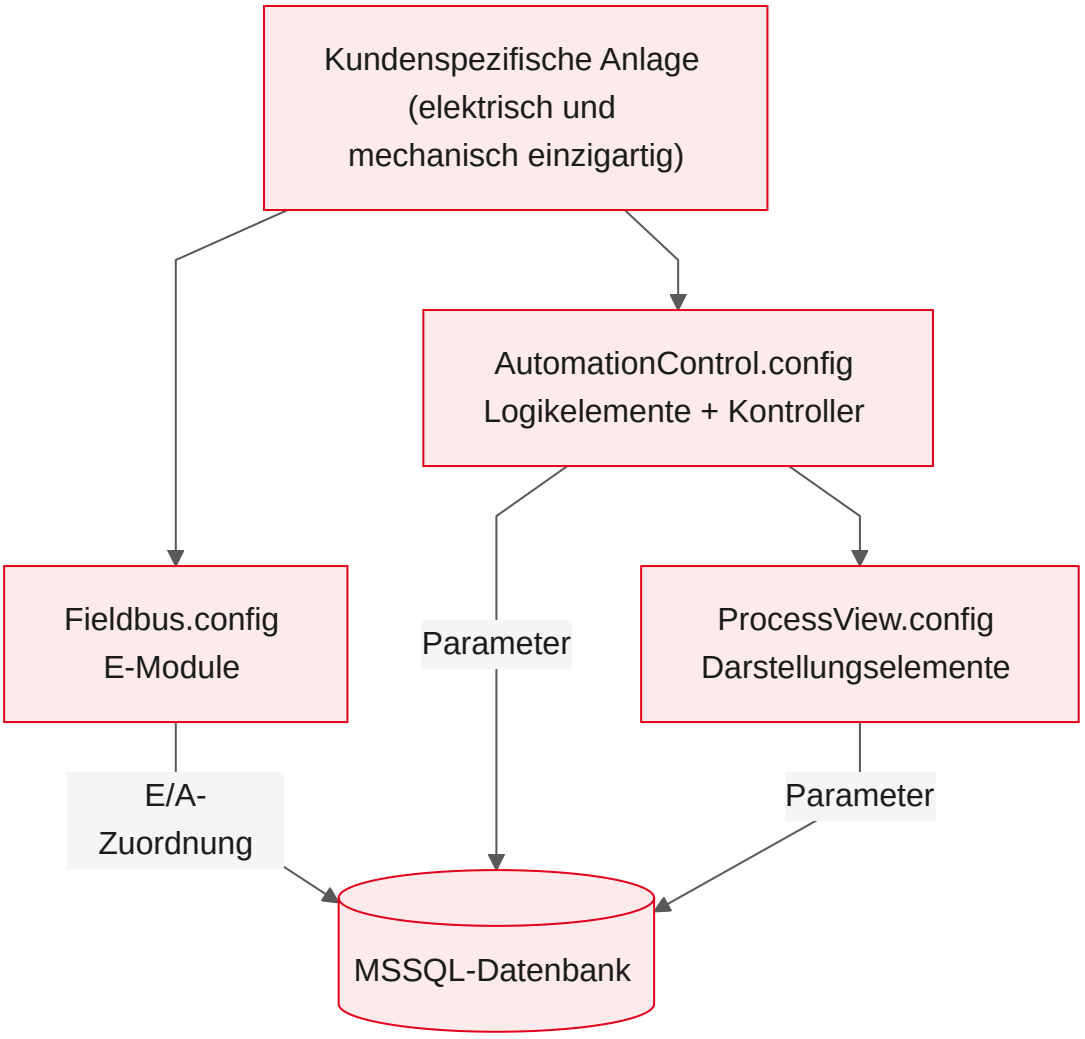

Abbildung 9: Abhängigkeiten der XML-Konfigurationsdateien und der Datenhaltung

Die folgende Tabelle gibt die Grössenordnung einer Anlagensteuerung wieder. Die Werte der Durchschnittsanlage sind der Mittelwert aus vier ausgewerteten Anlagen:

Tabelle 6: Kennzahlen einer Anlagensteuerung

Kennzahl	Wert
Standardelemente	337
Aktive Elemente in Steuerungsvorlage	276
Aktive Elemente pro Durchschnittsanlage	ca. 370
Parameter pro Durchschnittsanlage	ca. 29'800
Elektrische Signale pro Durchschnittsanlage	ca. 5'500

Die folgenden Abbildungen zeigen die Prozessvisualisierung der **as1** und veranschaulichen die Grösse einer Steuerung in der Praxis: Abbildung 10 zeigt die Standardvorlage mit 276 aktiven Elementen, Abbildung 11 die Steuerung eines Projekts durchschnittlicher Grösse. Abbildungen 12 und 13 zeigen eine Grossanlage, deren Steuerung wegen des Umfangs auf zwei Konsolen aufgeteilt wird: Die Konsole Mix steuert den Misch- und Verladebereich, die Konsole Dry den Bereich der Gesteinstrocknung.

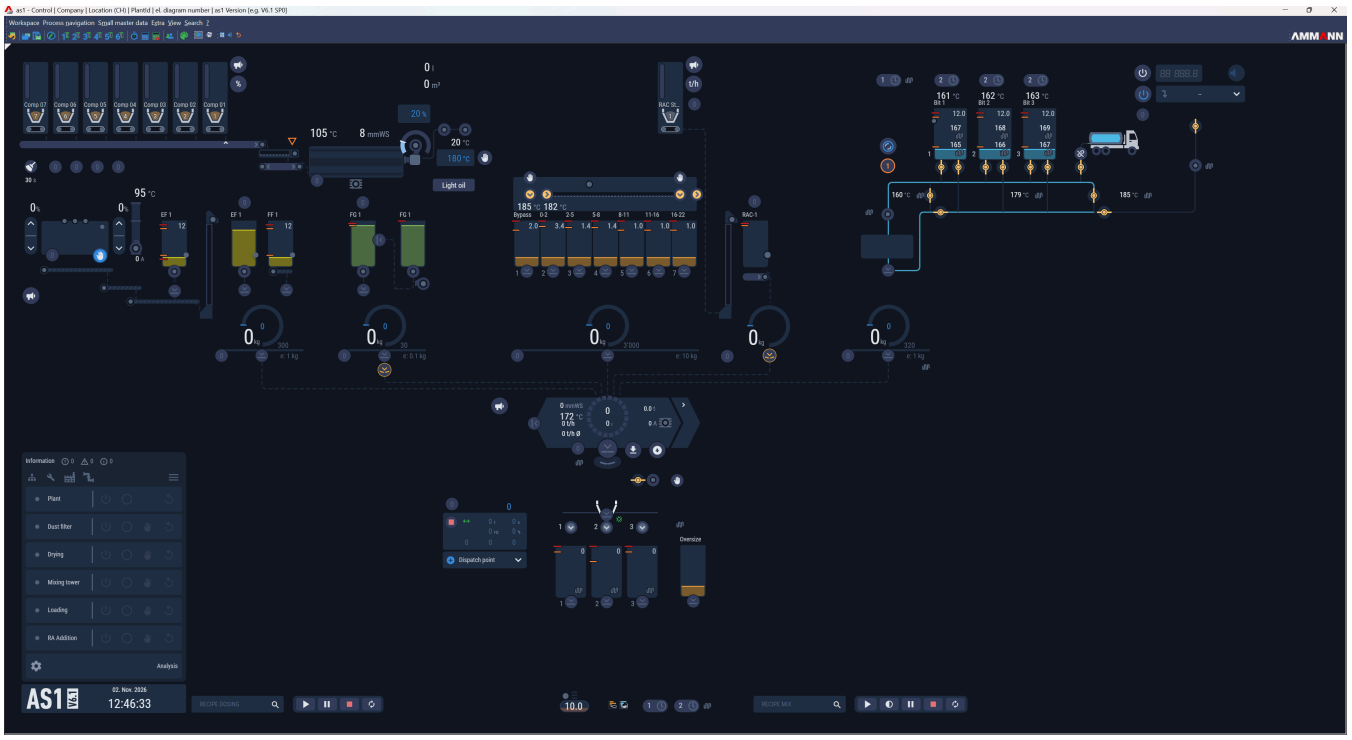

Abbildung 10: as1-Standardvorlage der Steuerung (276 aktive Elemente)

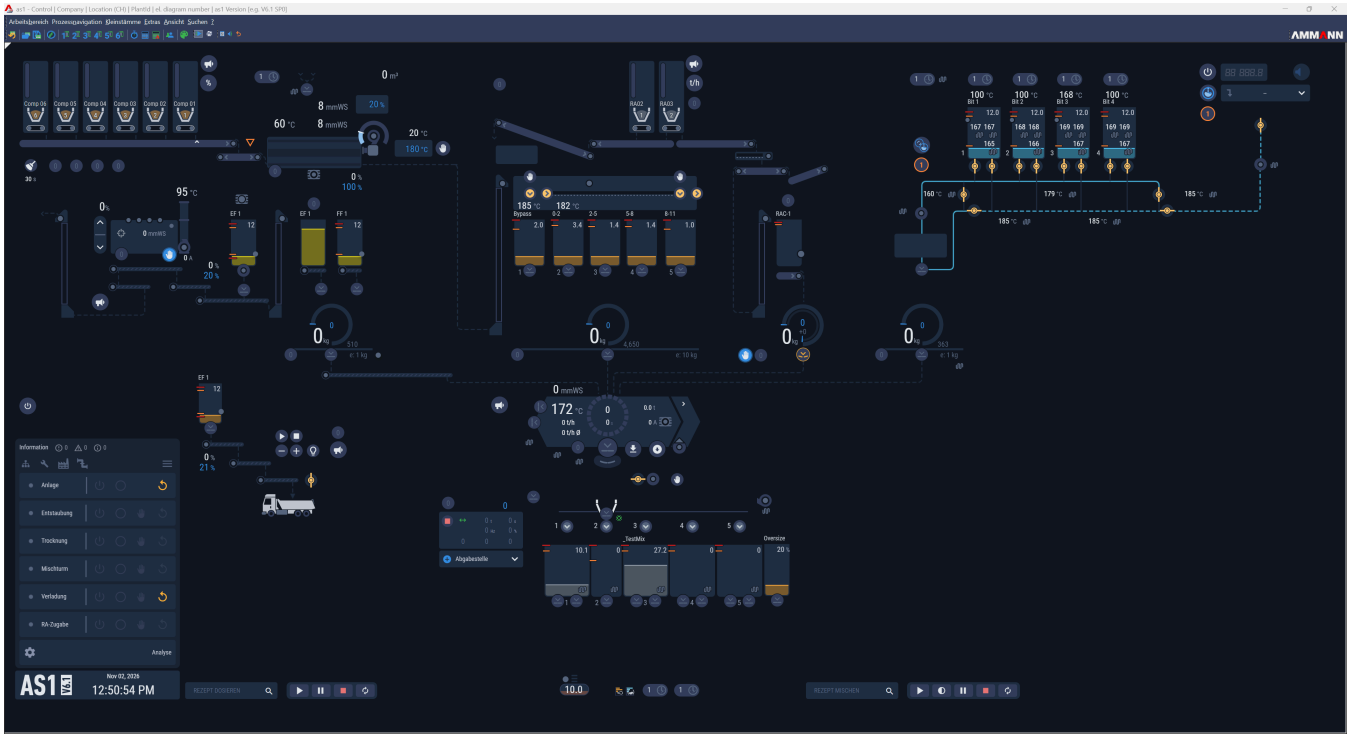

Abbildung 11: as1-Steuerung eines Projekts durchschnittlicher Grösse (ca. 370 aktive Elemente)

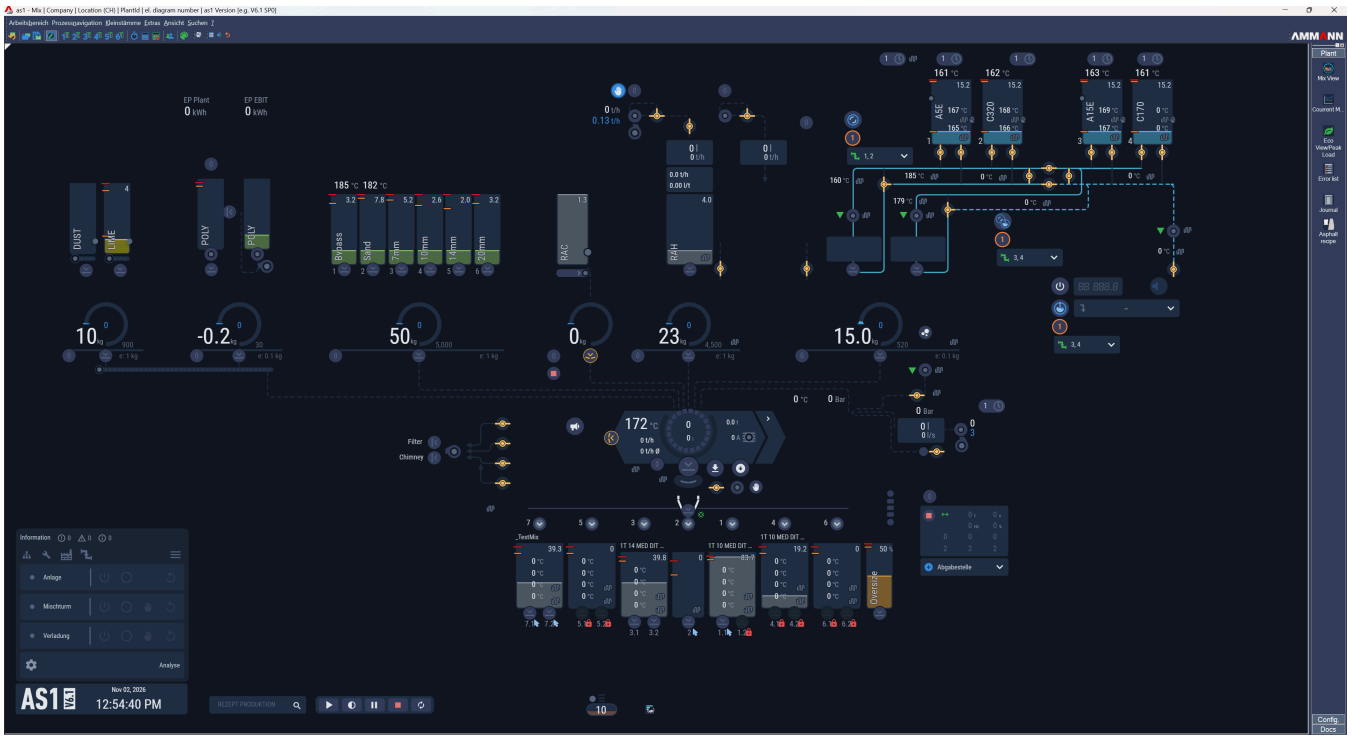

Abbildung 12: Grosse as1-Steuerung, Konsole Mix (Misch- und Verladebereich)

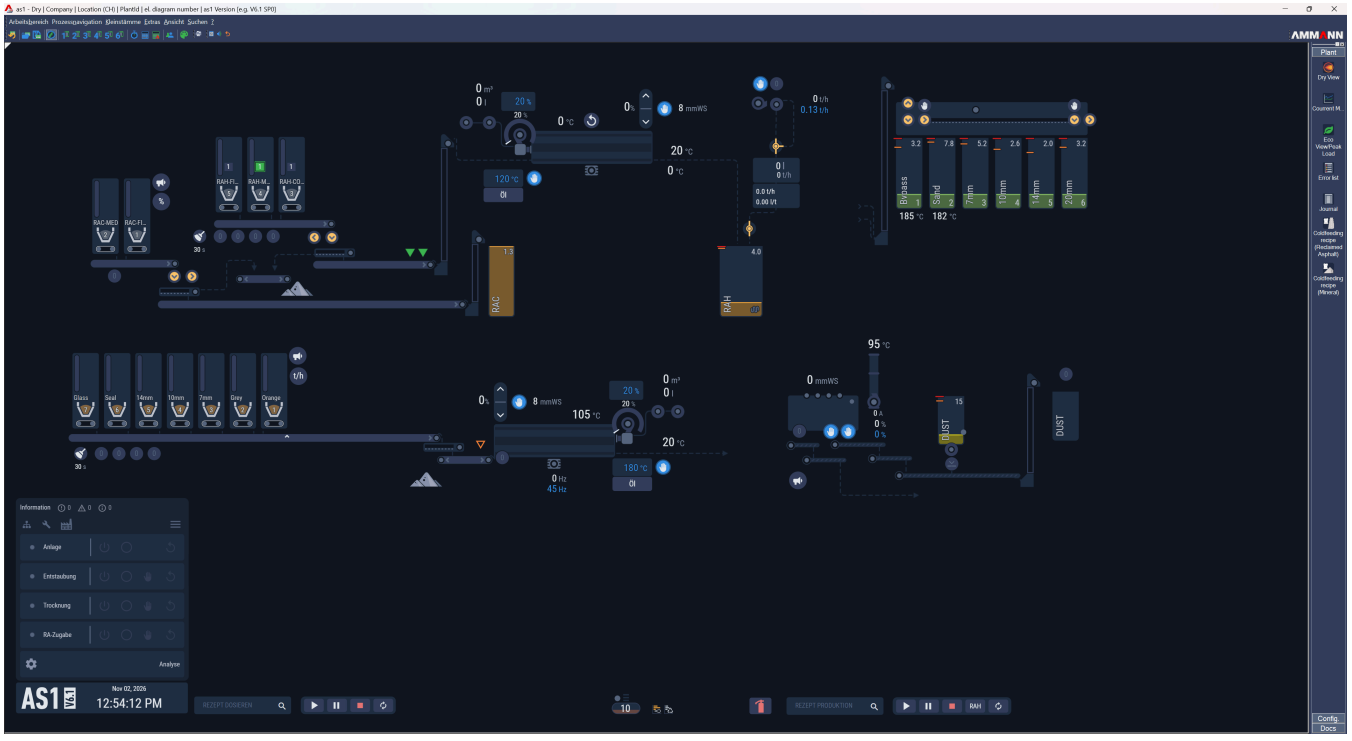

Abbildung 13: Grosse as1-Steuerung, Konsole Dry (Gesteinstrocknung)

Datenhaltung der Parameterstruktur

Die Parameterstruktur und die elektrischen Ein- und Ausgänge werden in der lokalen MSSQL-Datenbank gespeichert und können als XML-Datei exportiert werden. Die Datenbank enthält eine Änderungshistorie, in der pro Parameter die letzten drei Änderungen mit Datum gespeichert werden.

6.1.2 Ist-Prozess der Erstellung von Updatepaketen

Weil jede Mischanlage individualisiert ausgeliefert wird, erfordern spätere Kundenerweiterungen, Optimierungen oder neue Funktionen individuelle Softwareanpassungen. Nach dem internen Konfigurieren, Parametrisieren und Testen der Steuerungssoftware muss der verantwortliche Software Engineer dafür ein Updatepaket für das betroffene Kundensystem erstellen. Das Paket muss so aufgebaut sein, das **alle** Software Engineers in der Lage sind, die Anpassung zu installieren. Auch solche, die die Anlage nicht kennen.

Die Erstellung des Updatepakets erfolgt aktuell weitgehend manuell und gliedert sich in drei Schritte:

Schritt 1 - Manuelles Erstellen der XML-Differenzdateien

Die in der Datenbank gespeicherten Parameter können über eine Exportfunktion der **as1** als XML exportiert werden. Der Engineer wählt dabei in einem Menü ausschliesslich die geänderten Parameter aus. Zusätzlich vergleicht der Engineer die lokal unter Windows gespeicherten XML-Konfigurationsdateien der **as1** und erstellt für jede angepasste Datei manuell eine Datei, welche die Änderungen darstellt.

Schritt 2 - Erstellen der SQL-Skripte

Sind Datenbankänderungen erforderlich, bereitet der Engineer auch diese vor. Vordefinierte SQL-Skripte stehen nur in Ausnahmefällen zur Verfügung. In der Regel passt der Engineer ein bestehendes Skript an, verfasst ein neues Skript oder macht die Änderungen erst während der Installation auf dem Kundenrechner manuell über die **as1**.

Schritt 3 - Manuelles Erstellen der ReadMe

Abschliessend erstellt der Engineer eine individuelle ReadMe-Datei mit einer manuell geschriebenen Installationsanleitung.

Der folgende Ablauf veranschaulicht den Ist-Prozess:

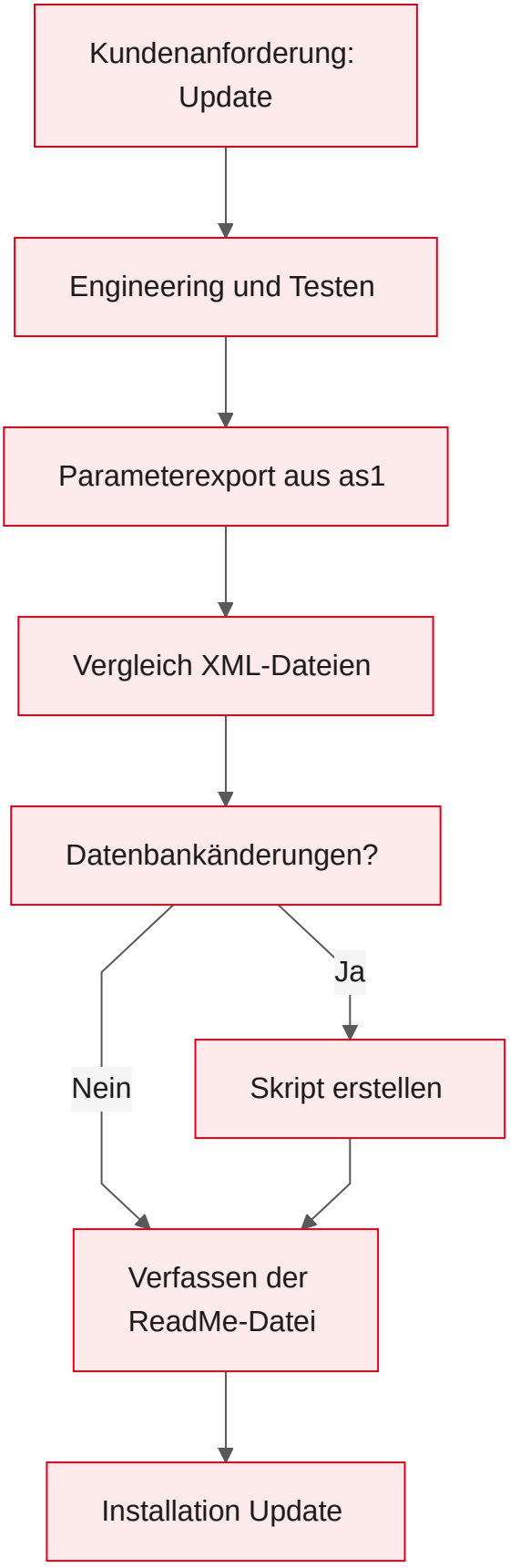

Abbildung 14: Ist-Prozess der Erstellung eines Updatepakets

Rahmenbedingungen im Ist-Prozess

- Die lokal gespeicherten XML-Konfigurationsdateien der **as1** liegen auf allen Kundensystemen am gleichen Ort.
- Für den Prozess sind keine definierten Werkzeuge vorgeschrieben. Gelegentlich kommen Vergleichstools wie Notepad++ [13] oder WinMerge [14] zum Einsatz.

6.1.3 Aufwand und Risiken im Ist-Zustand

Zeitaufwand

Ist-Zustand: Die manuelle Erstellung eines Updatepakets dauert durchschnittlich 30 bis 60 Minuten. Dieser Wert ist eine Expertenschätzung des Verfassers auf Basis von rund fünf Jahren Berufserfahrung als Softwareengineer bei der Ammann Schweiz AG. Ein systematischer Nachweis, zum Beispiel über eine Zeiterfassung pro Paket, liegt nicht vor. Für die anschliessende Installation beim Kunden fällt beim Engineer nochmals ein vergleichbarer Zeitaufwand an.

Fehlerquellen und ihre Ursachen

Die Fehleranfälligkeit des Ist-Prozesses hat drei Ursachen. Sie lässt sich an konkreten Fehlerarten zeigen:

Ursache 1: Die Auswahl der Änderungen erfolgt manuell (Schritt 1 und 2). Der Engineer entscheidet selbst, welche Parameter und XML-Änderungen in das Paket aufgenommen werden. Bei rund 29'800 Parametern und 5'500 elektrischen Signalen pro Durchschnittsanlage (vgl. Kapitel 6.1.1) ist die Übersicht begrenzt. Mögliche Fehler:

- Ein geänderter Parameter wird beim Export übersehen und fehlt im Paket. **Die Folge:** Die installierte Anlage verhält sich anders als die getestete.
- Ein sicherheitsrelevanter Parameter fehlt. **Die Folge:** Im ungünstigsten Fall entsteht Sachschaden an der Mischanlage.
- Umgekehrt wird ein Parameter ins Paket aufgenommen, der nicht geändert werden sollte, und überschreibt einen kundenspezifischen Wert auf dem Zielsystem.

Ursache 2: Die Differenzdarstellung ist abhängig vom Werkzeug und vom Engineer (Schritt 1). Ohne vorgeschriebenes Vergleichswerkzeug entstehen je nach Engineer und verwendetem Werkzeug unterschiedlich vollständige oder unterschiedlich lesbare Differenzdateien. Beim nachträglichen Lesen, z.B. durch einen Stellvertreter, können dadurch Änderungen falsch interpretiert oder übersehen werden.

Ursache 3: Datenbankänderungen werden nicht immer vorgefertigt (Schritt 2). Weil SQL-Skripte meist erst bei der Installation auf dem Kundenrechner entstehen oder manuell ausgeführt werden, können sich Tippfehler einschleichen, Skripte im falschen Kontext laufen oder im ungünstigsten Fall ganz wegfallen. Die Folge sind fehlerhafte oder fehlende Datenbankeinträge.

Qualitätsrisiko Dokumentation

Die ReadMe-Datei wird individuell verfasst. Form, Detailgrad und Verständlichkeit hängen deshalb stark vom jeweiligen Engineer ab. Bei der Installation führt dies häufig zu Rückfragen beim Paket-Ersteller.

Fallbeispiel: Verzögerte Installation eines Updatepakets

Ein Kunde bestellt ein Update für seine Mischanlage X. Der zuständige Engineer bereitet das Update vor und erstellt die Differenzdateien, SQL-Skripte und die ReadMe-Datei. Die Installation des Updates wird nicht zum Liefertermin abgerufen und verzögert sich um ein halbes Jahr. Der Updateauftrag fällt in eine Abwesenheit des Engineers, weshalb sein Stellvertreter die Installation durchführen muss. Die ReadMe-Datei ist für den Stellvertreter nicht verständlich, sodass wichtige Schritte vergessen gehen. Das Update lässt sich nicht erfolgreich installieren. Der Fehler wird erst nach langer Suche gefunden. Statt der üblichen 30 Minuten Installationszeit werden 3 Stunden aufgewendet.

Das Fallbeispiel zeigt, wie sich die drei Ursachen in der Praxis verstärken: Die personelle Abhängigkeit (nur der Ersteller versteht sein Paket) trifft auf ungenügende Dokumentation und auf manuell erstellte, schwer nachvollziehbare Paketinhalte.

6.1.4 Analyse des bestehenden Deployment-Tools (as1HotfixInstaller)

Für die Installation von Updates auf den Kundensystemen existiert bereits das interne Engineering-Tool **as1HotfixInstaller** [2]. Es wird als normaler **as1**-Hotfix ausgeliefert und im Ordner `%COMA%\Tools\HotfixInstaller` installiert. Das Tool kann **as1**-Hotfixes und weitere Pakete installieren, die als selbstentpackendes 7-Zip-Archiv vorliegen. Es unterstützt zwei Betriebsmodi: eine grafische Benutzeroberfläche sowie einen stillen Kommandozeilenmodus (Silent Mode) über das Konsolenargument `-s`. Im Silent Mode werden die Pakete standardmässig aus dem Ordner `C:\Public\Hotfixes` geladen (anpassbar über `-p`).

Grundsätzlicher Installationsablauf des Tools:

1. Entpacken aller Hotfixes und Pakete in temporäre Ordner
2. Prüfen, ob Prozesse Dateien sperren, die ersetzt oder verändert werden müssen
3. Backup aller Dateien, die ersetzt oder verändert werden, nach `%COMA%\Updates`
4. Installation gemäss den Anweisungen der `install.config`
5. Löschen der Paketarchive (nur auf Anforderung)

Alle Statusmeldungen werden in Logdateien im Ordner `C:\ProgramData\Ammann\ComaData\Logs\HotfixInstaller` protokolliert. Die Datei `Updates.log` dokumentiert pro **as1**-System, welche Hotfixes und Pakete erfolgreich installiert wurden. Pro Installation entsteht zudem eine detaillierte Logdatei `Updates_YYYYMMdd_aagXYZ.log` mit jedem einzelnen Installationsschritt.

Paketformat: Pakete sind selbstentpackende 7-Zip-Archive [11] und müssen zwingend das Namenspräfix `Package-` tragen (z.B. `Package-Kundenspezifisch-V1.0.exe`). Versionsangaben im Namen sind bei Paketen nicht erforderlich. Pakete ohne dieses Präfix werden wie Hotfixes behandelt. Für jedes Paket, das kein Hotfix ist, ist eine `install.config` zwingend erforderlich.

Paketstruktur: Auf der obersten Ebene des Archivs liegt die `install.config`. Dateien, die eine Zieldatei auf dem **as1**-System ersetzen, müssen den exakten Namen der Zieldatei tragen und in derselben Ordnerstruktur abgelegt sein (beginnend bei `Coma` / `ComaData` bzw. `Program Files (x86)` / `ProgramData`). Beispiel: Die Datei `AutomationControl.config` mit Ziel `C:\ProgramData\Ammann\ComaData\Pcs\AutomationControl.config` liegt im Paket unter `ComaData\Pcs\AutomationControl.config`. Dateien ohne Zieldatei, hauptsächlich

Parameterexporte und SQL-Skripte, können frei abgelegt werden, da ihr Pfad in der `install.config` definiert wird. Als Konvention liegt beides im Ordner `Coma\Database`.

Aufbau der install.config: Die Steuerdatei besteht aus fünf optionalen Hauptelementen innerhalb des Wurzelelements `InstallInstructions` (mit optionalem Attribut `MinimalInstallerVersion` zur Versionsprüfung des Installers):

Tabelle 7: Hauptelemente der install.config

Element	Funktion
`Prerequisites`	Installationsvoraussetzungen (derzeit nicht implementiert)
`InstallationParameters`	Vom Benutzer vor der Installation zu definierende Parameter (derzeit nicht implementiert)
`PreInstallation`	Schritte vor der Hauptinstallation
`Installation`	Hauptschritte (nur Pakete, bei Hotfixes ignoriert)
`PostInstallation`	Schritte nach der Hauptinstallation

Verfügbare Installationsschritte (in `PreInstallation`, `Installation` und `PostInstallation`):

*Tabelle 8: Installationsschritte des **as1HotfixInstaller**s*

Schritt	Funktion
`ShowWarning`	Zeigt dem Benutzer vor der Installation eine Warnung mit Fortsetzen/Abbrechen
`CreateBackup`	Erstellt ein Backup der Datenbank, der Parameter oder einer Datei
`ModifyXml`	Verändert eine XML-Zieldatei anhand einer XML-Modifikationsdatei im Paket
`ExecuteSql`	Führt ein einzelnes SQL-Skript in der **as1**-Datenbank aus
`Execute`	Führt einen Befehl (cmd.exe) oder eine Datei aus
`ImportParameters`	Importiert einen **as1**-Parameterexport
`CopyFile`	Kopiert eine Datei aus dem Paket ins Zielsystem (mit Option `OverwriteExisting`)
`DeleteFile` / `DeleteFolder`	Sichert und löscht Dateien (auch mit Wildcard-Muster) bzw. einen Ordner
`ImportRegistry`	Importiert eine .reg-Datei in die Windows-Registrierung
`DownloadAndExecute`	Lädt ein Programm aus dem Internet und führt es aus (überspringbar, wenn Datei/Ordner existiert)

Schritt	Funktion
`ShowHtml`	Öffnet nach der Installation eine HTML-Datei (z.B. ergänzende Installationsanleitung) im Browser

XML-Modifikationsdateien: Für den Schritt `ModifyXml` definiert das Paket eine XML-Modifikationsdatei mit strukturierten Änderungsanweisungen. Sie enthält die drei Schritte `Add`, `Remove` und `Replace`, die über XPath-Ausdrücke bzw. Kommentar-Anker (`AboveXPath`, `BelowXPath`, `AboveComment`, `BelowComment`) die Position im XML-Zieldokument referenzieren. Beispiel:

```
<XmlConfiguration>
  <Add BelowComment="Haftmittel" BelowXPath="//ControlObject[@Caption='Bitumen']">
    <ControlObject Type="CAucScVolHaftmittel" Caption="Haftmittel"/>
  </Add>
  <Replace XPath="//ControlObject[@Caption='Filter']">
    <ControlObject Type="CAucFilterKunde" Caption="Filter"/>
  </Replace>
  <Remove XPath="//ControlObject[@Caption='ElementToRemove1']"/>
</XmlConfiguration>
```

Für die Updatepaketerstellung ist dies zentral: Die XML-Differenzdateien müssen nicht als freies Diff-Format, sondern als solche XML-Modifikationsdateien vorliegen, um vom **as1HotfixInstaller** automatisch angewendet werden zu können.

6.1.5 Schwachstellenanalyse und Systemgrenze

Die Gegenüberstellung des Ist-Prozesses (Kapitel 6.1.2) und der Funktionsweise des **as1HotfixInstaller** (Kapitel 6.1.4) legt die folgenden Schwachstellen im Ist-Zustand offen.

Tabelle 9: Schwachstellen des Ist-Prozesses

Nr.	Schwachstelle	Ursache	Auswirkung
1	XML-Differenzdateien werden vollständig manuell erstellt.	Manuelle Auswahl der Änderungen bei hoher Parameterzahl.	Zeitaufwand, übersehende oder falsch übernommene Änderungen.
2	SQL-Skripte werden manuell erstellt oder ganz weggelassen.	Keine vordefinierten Skripte, Änderungen teils erst auf dem Kundensystem.	Fehlerhafte oder fehlende Datenbankeinträge.
3	Die ReadMe wird individuell verfasst.	Keine Vorgabe zu Form und Detailgrad.	Qualitätsschwankungen, Rückfragen, Fehlinstallationen.
4	Es existiert keine Richtlinie für den Paketaufbau.	Fehlender Paketstandard.	Pakete unterscheiden sich je nach Engineer markant.

Nr.	Schwachstelle	Ursache	Auswirkung
5	Es existiert kein Werkzeug für die Paketerstellung.	Fehlende Werkzeugunterstützung.	Der ganze Prozess beruht auf manueller Arbeit.

Schlussfolgerung: Die Schwachstellen lassen sich auf zwei grundsätzliche Punkte zurückführen. Erstens fehlt ein definierter Standard, wie ein Updatepaket aufgebaut sein muss. Zweitens fehlt ein Werkzeug, das die Schritte 1 bis 3 des Ist-Prozesses unterstützt oder automatisiert. Der **as1HotfixInstaller** als Installationstool kann diese Lücke nicht schliessen, da er erst greift, nachdem das Paket manuell erstellt wurde. Daraus ergibt sich die Notwendigkeit eines neuen Tools für die Paketerstellung, das die Erkennung der Änderungen, die Erzeugung der Differenzdateien und SQL-Skripte sowie die Dokumentation standardisiert.

6.1.6 Systemgrenzen des neuen Tools

Das neu zu entwickelnde Tool setzt an der Stelle an, an der der Engineer das Update in der **as1** fertig konfiguriert und parametrisiert hat. Innerhalb der Systemgrenzen liegen:

- Die automatische Erkennung der Anpassungen an der Mischanlagensteuerung.
- Die Erzeugung der XML-Differenzdateien und SQL-Skripte.
- Die Erzeugung der `install.config`.
- Die standardisierte ReadMe-Erstellung.
- Die Paketierung in ein vom **as1HotfixInstaller** verarbeitbares Format.

Ausserhalb der Systemgrenze bleibt die Installation auf dem Kundensystem. Sie bleibt Aufgabe des **as1HotfixInstaller**.

6.2 Kreativitätsmethode

(Konzeptphase: Die Methode diente der Suche nach Lösungsansätzen für das geplante Tool. Es wird noch keine Variante favorisiert.)

Als Kreativitätsmethode wird der morphologische Kasten genutzt. Bei dieser Methode wird die Problemstellung möglichst simpel aber präzise formuliert. Für die Problemstellung werden danach verschiedene Merkmale definiert. In einer Tabelle werden dann verschiedene Ausprägungen für die einzelnen Merkmale festgehalten. Anhand dieser Liste werden dann verschiedene Versionen einer Lösung definiert.

6.2.1 Fragestellung

Wie kann die Erstellung von Updatepaketen für die **as1** automatisiert werden, sodass Änderungen aus den XML Dateien automatisch erkannt, die Update Dateien in der für den **as1HotfixInstaller** vorgeschriebenen XML Struktur generiert und als selbst entpackendes 7-Zip Archiv paketiert werden?

Abgrenzung der Fragestellung: Die Vorgaben `install.config` und selbstentpackendes 7-Zip-Archiv sind bewusst Teil der Fragestellung, da es sich um feste Ist-Restriktionen des bestehenden Deployment Tools handelt (vgl. Kapitel 6.1.4). Diese Elemente werden nicht variiert, sondern bilden den fixen Rahmen, innerhalb dessen die Varianten gebildet werden. Die übrigen Aspekte (Differenzerkennung, SQL-Generierung, ReadMe-Erstellung und Paketierung) sind frei gestaltbar und werden im morphologischen Kasten variiert.

6.2.2 Morphologischer Kasten

Tabelle 10: Morphologischer Kasten

Nr.	M1: Differenz XML	M2: Differenz SQL	M3: Technologie	M4: Generierung 'install.config'	M5: Readme	M6: Paketierung
1	Manuell	Manuell	C++	Manuell	Manuell	Manuell mit 7-Zip
2	Datenvergleich mit 3rd Party Tool	Schemavergleich mit Skript (alt/neu)	C# / .NET	Eingabe im GUI	Aus Metadaten generiert	Halbautomatisch über Skript
3	Versionsverwaltungsbasiert	Generiertes SQL Skript	TypeScript / Node.js	Automatisch aus Ordnerstruktur	Automatisch aus der 'install.config' / Paketinhalt generiert	Vollautomatisch in einem Tool integriert
4	Neues Tool das Vergleich macht	Vergleich über Vorlage	Powershell	Automatisch aus Metadateneingabedatei	-	-

Die Merkmale M5 und M6 verfügen über drei Ausprägungen, alle übrigen über vier. Damit ergeben sich **2'304** mögliche Kombinationen.

Verweise auf den Kasten: Abgekürzt werden die Merkmale des Kastens mit **M**, ihre Ausprägungen mit **A**. Die Bezeichnung *M1-A2* bezeichnet somit die zweite Ausprägung des ersten Merkmals, also "Datenvergleich mit 3rd-Party-Tool" für die Differenzerkennung der XML-Dateien. Die Nummer der Ausprägung entspricht der Zeilennummer in der obigen Tabelle (A1 = erste Zeile, A4 = vierte Zeile), die Nummer des Merkmals der Spalte (M1 = erste Merkmalsspalte). Auf diese Schreibweise wird in den folgenden Kapiteln zur Priorisierung und Variantenbildung Bezug genommen.

6.3 Priorisierungsmethode

(Konzeptphase: Die Priorisierung grenzt die Lösungsansätze anhand der Projektziele ein.)

Für die Variantenbildung mit dem morphologischen Kasten muss zuerst die Anzahl der möglichen Kombinationen reduziert werden. Ohne Reduktion wären mit den gefundenen Ausprägungen 2'304 verschiedene Varianten möglich. Die Reduktion erfolgt in zwei Schritten: Zuerst werden die dem Automatisierungsziel widersprechenden Ausprägungen ausgeschlossen, anschliessend werden die untereinander unverträglichen Ausprägungen ermittelt.

Schritt 1: Ausschluss manueller Ausprägungen

Sämtliche rein manuelle Ausprägungen (M1-A1, M2-A1, M4-A1, M5-A1, M6-A1) widersprechen dem in der Fragestellung formulierten Automatisierungsziel und adressieren keine der Schwachstellen aus Kapitel 6.1.5. Sie werden vollständig ausgeschlossen. Ebenfalls ausgeschlossen wird M1-A3 (versionsverwaltungsbasierte Differenzerkennung), da für die **as1**-Konfigurationen aktuell keine Versionsverwaltung eingesetzt wird. Ein Aufbau eines solchen Systems würde den Rahmen dieser Arbeit sprengen. Damit reduziert sich die Anzahl der Kombinationen auf **192**.

Schritt 2: Verträglichkeitsprüfung

Die verbleibenden Ausprägungen werden paarweise auf Verträglichkeit geprüft. Nicht oder nur bedingt verträgliche Kombinationen sind:

Tabelle 11: Verträglichkeitsprüfung der Ausprägungen

Ausprägungen	Verträglich	Begründung
M4-A2 <-> M3-A4	Bedingt	Ein GUI ist in PowerShell über Windows Forms oder WPF grundsätzlich realisierbar, jedoch nicht zweckmässig. GUI-Code und Ablauflogik vermischen sich in einem Skript, was die Wartbarkeit bei wachsender Komplexität stark einschränkt.
M6-A3 <-> M3-A4	Bedingt	Ein Kommandozeilenaufruf von 7-Zip aus PowerShell ist möglich, eine echte Integration in ein Tool liegt jedoch nicht vor.
M1-A2 <-> M2-A3	Bedingt	Wenn die SQL-Skripte generiert werden, wäre die XML-Differenzierung über ein Fremdwerkzeug inkonsistent. Die Differenzerkennung sollte in beiden Merkmalen nach demselben Prinzip erfolgen.
M5-A2 <-> M4-A3	Bedingt	Bei automatischer Generierung der 'install.config' aus der Ordnerstruktur müssen die Metadaten anderweitig erhoben werden, da die Ordnerstruktur alleine keine semantischen Informationen (z.B. Beschreibungstexte) enthält.

Schritt 3: Bildung von Blickwinkeln zur Variantenauswahl

Nach dem Ausschluss und der Verträglichkeitsprüfung verbleiben 192 mögliche Kombinationen. Eine Bewertung all dieser Kombinationen wäre weder sinnvoll noch durchführbar. Viele Kombinationen unterscheiden sich nur unwesentlich voneinander, und einzelne Ausprägungen treten in mehreren Kombinationen auf, ohne dass sich diese wesentlich unterscheiden. Aus der Menge der möglichen Kombinationen müssen daher einige wenige, charakteristische Varianten ausgewählt werden, die das Spektrum des Lösungsraums abdecken.

Für diese Auswahl werden **Blickwinkel** gebildet. Ein Blickwinkel ist ein Optimierungskriterium, das eine Variante bewusst in den Vordergrund stellt. Er beantwortet die Frage: "Wie sähe eine Lösung aus, wenn dieses Kriterium das wichtigste wäre?" Da die Fragestellung (Kapitel 6.2.1) die Automatisierung vorgibt, aber nicht, in welchem Grad und mit welcher Technologie sie umzusetzen ist, liegen die wesentlichen Unterschiede zwischen denkbaren Lösungen im Automatisierungsgrad, im Aufwand und in der Art der Änderungserfassung.

Es werden drei Blickwinkel gebildet, die bewusst gegensätzlich gewählt sind, damit die ausgewählten Varianten das Spektrum des Lösungsraums möglichst weit abdecken:

1. **Schnellste Umsetzung:** Die Lösung mit dem geringsten Aufwand, die die Fragestellung bereits teilweise erfüllt. Sie zeigt, was mit minimalen Mitteln erreichbar wäre.
2. **Vollständige Automatisierung im bestehenden Engineering-Prozess:** Eine Lösung, die den gesamten Paketerstellungsprozess automatisiert, ohne dass der Engineer seinen gewohnten Ablauf ändert. Die Änderungen werden vom Tool aus den vorhandenen Quellen (Snapshots, Änderungshistorie) selbst erhoben.
3. **Maximale Automatisierung und Reproduzierbarkeit:** Die weitestgehende Lösung, bei der die Änderung nicht vom Tool erkannt, sondern vom Engineer deklarativ beschrieben wird und daraus sämtliche Paketbestandteile generiert werden.

Jeder Blickwinkel führt zu einer konkreten Kombination von Ausprägungen aus dem morphologischen Kasten. Zu jedem Blickwinkel werden die Ausprägungen ausgewählt, die ihn am besten umsetzen. Die daraus entstandenen drei Varianten werden in Kapitel 6.4 beschrieben. Die Blickwinkel selbst nehmen keine Bewertung vor. Sie strukturieren lediglich die Auswahl. Welcher Blickwinkel der Diplomarbeit am besten dient, entscheidet ausschliesslich die Nutzwertanalyse in Kapitel 6.5.

6.4 Variantenbildung

(Konzeptphase: Die folgenden Varianten sind ausgearbeitete Entwürfe. Keine der Varianten ist zu diesem Zeitpunkt realisiert.)

Nach der Auswahl der Blickwinkel (Kapitel 6.3, Schritt 3) werden die daraus entstandenen Varianten ausgearbeitet. Allen Varianten ist der zeitliche Ablauf gemeinsam. Der Snapshot des Ausgangszustands wird **vor Beginn** der Anpassungen an der Software erstellt, die Differenzerkennung erfolgt **nach Abschluss** der Anpassungen. Der Vergleich spannt sich damit über die gesamte Engineering-Phase auf. Jede Variante kombiniert konkrete Ausprägungen aus dem morphologischen Kasten:

Tabelle 12: Variantenübersicht (Merkmale je Variante)

Variante	M1: Differenz XML	M2: Differenz SQL	M3: Technologie	M4: Generierung 'install.config'	M5: Readme	M6: Paketierung
1. Skriptbasiert (PowerShell)	A4	A2	A4	A3	A1*	A2
2. Integrales GUI-Tool (C#/.NET)	A4	A3	A2	A2	A2	A3
3. Metadaten-getrieben	A4	A3	A3	A4	A2	A3

** Die ReadMe-Datei (M5-A1) wird in Variante 1 bewusst als Kompromiss beibehalten. PowerShell eignet sich für die Ablaufsteuerung, aber kaum für die automatisierte Erzeugung semantisch vollständiger Dokumentation. Die vollständige Automatisierung der ReadMe ist damit Teil des Differenzierungsmerkmals der Varianten 2 und 3.*

6.4.1 Variante 1: Skriptbasierte Automatisierung (PowerShell)

Blickwinkel: Schnellste Umsetzung.

Begründung: Diese Variante ist die kleinstmögliche Kombination, die die Fragestellung teilweise erfüllt.

Beschreibung: Ein einzelnes PowerShell-Skript erweitert den bestehenden Prozess um automatisierte Schritte. Die Bedienung erfolgt ausschliesslich über Kommandozeilenparameter, auf eine grafische Oberfläche wird bewusst verzichtet. Für den späteren direkten Aufruf aus der **as1** heraus muss die **as1** um eine entsprechende Startfunktion (Aufruf des Tools mit Startzeilenargumenten) erweitert werden. Diese Erweiterung ist nicht Teil des Skripts selbst. Zwar wären einfache GUI-Elemente in PowerShell über Windows Forms oder WPF realisierbar, widerspricht dies jedoch dem Blickwinkel "schnellste Umsetzung" und der Zielsetzung eines wartbaren, schlanken Skripts (vgl. Kapitel 6.3, Schritt 2).

1. **Snapshot:** Vor Beginn der Anpassungen an der Software kopiert das Skript die lokal gespeicherten XML-Konfigurationsdateien der **as1** in einen mit Datum benannten Snapshot-Ordner und sichert damit den Ausgangszustand. Das Datum des Referenz-Snapshots wird als Parameter übergeben.
2. **XML-Differenzierung:** Nach Abschluss der Anpassungen vergleicht das Skript den Referenz-Snapshot mit dem aktuellen Zustand der XML-Dateien. Da die Dateien auf allen Kundensystemen am gleichen Ort liegen (Kapitel 6.1.2), sind die Pfadangaben fest im Skript hinterlegt. Für jede geänderte Datei wird eine Differenzdatei erzeugt, die der bestehenden manuellen Praxis entspricht, aber reproduzierbar erstellt wird.
3. **SQL-Vorbereitung:** Für die Datenbankänderungen wird auf ein Skript-Template zurückgegriffen, das der Engineer vor der Paketierung inhaltlich ergänzt. Da die Änderungsinformationen pro Parameter in der MSSQL-Datenbank hinterlegt sind und ohne ein zusätzliches SQL-Skript nur über die **as1** abrufbar sind (vgl. Kapitel 6.1.1), dient das Template als Sammelpunkt: Der Engineer überträgt die in der **as1** sichtbaren Änderungen in das Template.
4. **install.config und ReadMe:** Die `install.config` wird automatisch aus der Ordnerstruktur des Paketinhalts generiert. Kopiervorgänge und SQL-Ausführungen werden den gefundenen Dateien zugeordnet. Die ReadMe wird aus einer Vorlage gefüllt, in die der Engineer Beschreibung und Installationsschritte manuell einträgt.
5. **Paketierung:** Das Skript ruft die 7-Zip-Kommandozeile auf und erzeugt das selbstentpackende Archiv mit dem vorgeschriebenen Namenspräfix `Package-`.

- **Vorteile:** Geringer Entwicklungsaufwand, schnell einführbar. Keine zusätzliche Laufzeitumgebung, da PowerShell und 7-Zip auf Windows-Systemen vorhanden sind. Automatisiert die fehleranfälligsten manuellen Schritte (Schwachstellen 1 und 2 aus Kapitel 6.1.5).
- **Nachteile:** Keine grafische Oberfläche. Bedienung über Parameter ist eingeschränkt und fehleranfällig bei falscher Reihenfolge oder fehlenden Argumenten. Schlechte Wartbarkeit bei wachsender Skriptlänge. Keine Validierung der Benutzereingaben. Die Schwachstellen 3 (Dokumentationsqualität) und 4 (fehlender Paketstandard) aus Kapitel 6.1.5 werden nicht behoben. Für den direkten Aufruf aus der **as1** ist eine **as1**-Erweiterung erforderlich.

Technische Details:

- **XML-Vergleich:** Das Skript vergleicht die Snapshots dateiweise (Hash-Vergleich zur Vorfilterung, danach zeilenweiser Abgleich geänderter Dateien). Für die Differenzdateien stehen zwei vom **as1HotfixInstaller** unterstützte Wege zur Verfügung (vgl. Kapitel 6.1.4): die vollständige geänderte Datei, die über den Schritt `CopyFile` installiert wird, oder eine echte Differenz in Form einer XML-

Modifikationsdatei (`XmlConfiguration` mit `Add`-, `Remove`- und `Replace`-Anweisungen), die über den Schritt `ModifyXml` installiert wird. Das Skript erzeugt beide Formate, da die installierende Person je nach Fall beide benötigt: Das vollständige Abbild zeigt den Zielzustand, die Modifikationsdatei zeigt, was sich ändert und warum.

- **SQL-Zugriff:** Das Skript selbst greift nicht direkt auf die MSSQL-Datenbank zu. Die Datenbankänderungen stammen aus dem manuell ergänzten Template (siehe Schritt 3), was den Entwicklungsaufwand klein hält, den SQL-Schritt aber manuell belässt.
- **install.config-Schema:** Die Struktur der `install.config` ist vollständig dokumentiert (vgl. Kapitel 6.1.4): Wurzelelement `InstallInstructions` mit den fünf Hauptelementen `Prerequisites`, `InstallationParameters`, `PreInstallation`, `Installation` und `PostInstallation` sowie zwölf definierten Installationsschritten (u.a. `CopyFile`, `ModifyXml`, `ExecuteSql`, `ImportParameters`). Das Skript generiert die `install.config` aus der Ordnerstruktur. Ersetzte Dateien werden als `CopyFile`-Schritte, geänderte XML-Dateien wahlweise als `ModifyXml`-Schritte und SQL-Skripte als `ExecuteSql`-Schritte eingetragen.
- **Paketierung:** Die Erzeugung des selbstentpackenden Archivs erfolgt über den 7-Zip-Kommandozeilenaufruf mit SFX-Modul, mit dem vorgeschriebenen Namenspräfix `Package-`. Die Paketstruktur folgt den Vorgaben des **as1HotfixInstaller**s (vgl. Kapitel 6.1.4): `install.config` auf oberster Ebene, Zieldateien in der Ordnerstruktur der Ziele (beginnend bei `ComaData` bzw. `ProgramData`), SQL-Skripte und Parameterexporte gemäss Konvention im Ordner `Coma\Database`.

6.4.2 Variante 2: Modulare CLI Anwendung (C#/.Net)

Blickwinkel: Vollständige Automatisierung im bestehenden Engineering-Prozess.

Begründung: Die Paketerstellung wird vollständig automatisiert, ohne dass der Engineer seinen gewohnten Ablauf ändern muss. Die benötigten Informationen entstehen als Nebenprodukt der normalen Arbeit (Snapshots, Änderungshistorie) und werden vom Tool automatisch erhoben. Die verwendeten Technologien sind bei der Ammann Schweiz AG bereits vorhanden.

Beschreibung: Eine eigenständige Windows-Anwendung unterstützt den gesamten Prozess. Das Tool ist dabei auf zwei Arten bedienbar: über eine grafische Oberfläche, die den Engineer durch die Schritte führt, und über die Kommandozeile, über die der vollständige Ablauf ohne Interaktion ausgeführt werden kann. Beide Bedienformen nutzen dieselbe Ablaufsteuerung. Die Anwendung gliedert sich in folgende funktionale Bausteine:

1. **Snapshot-Verwaltung:** Vor Beginn der Anpassungen an der Software erstellt das Tool auf Knopfdruck einen Snapshot der XML-Konfigurationsdateien der **as1** und legt diesen in einem Ordner ab. Jedem Snapshot wird eine Metadaten-Datei (`snapshot.json`) beigelegt, die mindestens folgende Attribute enthält: Erstellungsdatum, Anlage, erstellender Engineer und ein frei wählbarer Kommentar. Das Erstellungsdatum definiert zugleich den Beginn des Änderungszeitraums, über den die Differenzerkennung später abgleicht.
2. **Differenzerkennung XML:** Nach Abschluss der Anpassungen wählt der Engineer den Referenz-Snapshot aus. Das Tool vergleicht den Snapshot mit dem aktuellen Zustand der XML-Dateien und erkennt die geänderten Dateien (z.B. AutomationControl.config, ProcessView.config). Aus den erkannten Änderungen werden die XML-Differenzdateien automatisch generiert.
3. **Differenzerkennung Datenbank:** Das Erstellungsdatum des gewählten Referenz-Snapshots definiert den Zeitpunkt, ab dem die Änderungen aus der Änderungshistorie der MSSQL-Datenbank (Kapitel 6.1.1) ausgewertet werden. Das Tool generiert daraus ein SQL-Skript, das die festgestellten Parameteränderungen reproduzierbar auf dem Kundensystem ausführt.
4. **Paketgenerierung:** Aus den erzeugten Dateien generiert das Tool die `install.config` (Kopiervorgänge und SQL-Ausführung in den drei Phasen Pre-Installation, Installation und Post-Installation, vgl. Kapitel 6.1.4), erzeugt die ReadMe-Datei standardisiert aus den Metadaten und paketiert alles vollautomatisch als selbstentpackendes 7-Zip-Archiv mit dem Namenspräfix `Package-`.
5. **Bedienung (GUI und CLI):** Über die grafische Oberfläche wird der Engineer durch die einzelnen Schritte geführt. Eingaben werden validiert, und das Tool verhindert unvollständige Pakete. Über die Kommandozeile sind dieselben Schritte ohne Interaktion ausführbar. Dies ermöglicht automatisierte Tests, die Integration in bestehende Abläufe und den späteren direkten Aufruf aus der **as1** (Muss-Ziel 1c).

Gegenüber der Ableitung des Snapshot-Datums aus dem Ordnernamen bietet die Metadaten-Datei zwei Vorteile: Der Ordnername bleibt frei wählbar und lesbar, und es können zusätzliche, für die ReadMe-Generierung benötigte Informationen (Anlage, Engineer, Kommentar) mitgeführt werden.

- **Vorteile:** Behebt sämtliche Schwachstellen aus Kapitel 6.1.5: automatische Differenzerkennung (Schwachstelle 1), generierte SQL-Skripte (Schwachstelle 2), standardisierte, aus den Paketdaten erzeugte ReadMe (Schwachstelle 3), einheitlicher Paketaufbau durch das geführte Tool (Schwachstelle 4) und eine definierte Werkzeugunterstützung (Schwachstelle 5). Die geführte Bedienung ermöglicht allen Software Engineers, unabhängig vom Erfahrungsstand und unabhängig davon, wer das Update

entwickelt hat, eine korrekte Installation und deckt damit direkt das Fallbeispiel aus Kapitel 6.1.3 ab. Gute Wartbarkeit und Erweiterbarkeit. .NET ist auf Windows-Systemen verfügbar.

- **Nachteile:** Das Tool muss gepflegt und dokumentiert werden. Mehr Entwicklungsaufwand als Variante 1. Der Zugriff auf die MSSQL-Datenbank bzw. deren Änderungshistorie muss technisch sauber gekapselt werden. Für den direkten Aufruf aus der **as1** ist eine **as1**-Erweiterung erforderlich.

Technische Details:

- **Datenbankzugriff:** Da die Änderungsinformationen pro Parameter in der MSSQL-Datenbank hinterlegt sind und ohne ein zusätzliches SQL-Skript nur über die **as1** abrufbar sind, liest das Tool die Änderungshistorie über eine eigene, nur lesende Datenbankverbindung aus. Die Zugriffslogik wird in einer eigenen Datenschicht gekapselt, damit das Tool keine schreibenden Zugriffe auf die Kundendatenbank ausführt und die **as1** nicht umgangen wird.
- **XML-Vergleich:** Der Vergleich läuft wie in Variante 1 zweistufig (Hash-Vorfilterung, danach zeilenweiser Abgleich). Das Tool erzeugt automatisch beide vom **as1HotfixInstaller** unterstützten Formate: die vollständige geänderte Datei (Installation via `CopyFile`) sowie die XML-Modifikationsdatei mit `Add`-, `Remove`- und `Replace`-Anweisungen und XPath-Referenzen (Installation via `ModifyXml`).
- **Snapshot-Format:** Die Metadaten-Datei `snapshot.json` enthält mindestens: Erstellungsdatum, Anlage, erstellender Engineer und Kommentar. Das Erstellungsdatum ist das technische Verknüpfungsattribut: Es definiert den Auswertungszeitpunkt der Änderungshistorie (Baustein 3) und wird zusätzlich für die ReadMe-Generierung verwendet.
- **install.config-Schema:** Das Tool generiert die install.config gemäss der dokumentierten Struktur (vgl. Kapitel 6.1.4) und validiert sie vor der Paketerstellung: Wurzelelement `InstallInstructions`, Schrittzuordnung zu den Phasen `PreInstallation`, `Installation` und `PostInstallation`, nur die zwölf definierten Installationsschritte, konsistente Pfadangaben gemäss Paketstruktur (Zieldateien ab `ComaData` / `ProgramData`, SQL-Skripte und Parameterexporte in `Coma\Database`). Die ReadMe-Datei wird als HTML-Datei erzeugt und über den Schritt `ShowHtml` in der PostInstallation automatisch nach der Installation geöffnet. Die Dokumentation ist damit Teil des Pakets und wird dem Installateur aktiv angezeigt statt nur beigelegt.
- **GUI-Architektur:** Die Anwendung trennt GUI, Ablaufsteuerung und Datenzugriff in separate Schichten. Dadurch bleibt die Vergleichs- und Generierungslogik auch ohne GUI über Kommandozeilenargumente ausführbar, was automatisierte Tests und die spätere Erweiterbarkeit ermöglicht.
- **Validierungskonzept:** Das Tool validiert an drei Stellen: die Eingaben in der grafischen Oberfläche, die erzeugte `install.config` gegen das dokumentierte Schema vor der Paketierung und das erzeugte Archiv nach der Paketierung (Existenz und Integrität). Was das Tool nicht prüfen kann, ist die inhaltliche Korrektheit der erkannten Parameteränderungen. Diese Prüfung bleibt Aufgabe des Engineers und wird durch die erzeugte ReadMe mit der tabellarischen Übersicht aller Änderungen unterstützt.
- **Paketierung:** Die Erzeugung des selbstentpackenden 7-Zip-Archivs mit dem Namenspräfix `Package-` erfolgt vollständig integriert über die 7-Zip-Kommandozeile, angestossen und auf Erfolg geprüft durch das Tool.

6.4.3 Variante 3: Metadaten-getriebene Vollautomatisierung

Blickwinkel: Maximale Automatisierung.

Begründung: Maximaler Ausbau der Möglichkeiten, die aus dem morphologischen Kasten hervorgehen.

Metadaten-Konzept: Metadaten sind in dieser Variante eine strukturierte, maschinenlesbare Beschreibung der Änderung, typischerweise eine JSON- oder YAML-Datei, die mindestens die betroffenen Parameter bzw. Konfigurationsdateien, die auszuführenden SQL-Statements, eine inhaltliche Beschreibung der Änderung sowie die für die ReadMe vorgesehenen Anweisungstexte enthält.

Beschreibung: Der Engineer beschreibt die Änderung deklarativ in der Metadaten-Datei, sobald die Anpassung in der **as1** fertig konfiguriert und getestet ist. Das Tool, basierend auf TypeScript und Node.js, verarbeitet diese Datei in einem einzigen Durchlauf:

1. **Validierung:** Die Metadaten-Datei wird gegen ein fest definiertes Schema geprüft (vollständige Pflichtangaben, gültige Parameter- und Dateireferenzen, syntaktisch korrekte SQL-Statements). Ungültige Eingaben werden unmittelbar mit konkreten Fehlermeldungen zurückgewiesen.
2. **XML-Differenzdateien:** Das Tool erzeugt aus den referenzierten Konfigurationsdateien die Differenzdateien. Die Änderung wird also nicht durch Vergleich zweier Zustände erkannt, sondern aus der Beschreibung abgeleitet.
3. **SQL-Skripte:** Die in den Metadaten enthaltenen SQL-Statements werden in das Skript für die Installationsphase der `install.config` überführt.
4. **install.config und ReadMe:** Sämtliche Installationsschritte werden aus den Metadaten generiert und den drei Phasen (Pre-Installation, Installation, Post-Installation) zugeordnet. Die ReadMe entsteht vollständig aus den Beschreibungs- und Anweisungstexten der Metadaten, sie ist damit nie optional, sondern zwingender Bestandteil des Prozesses.
5. **Paketierung:** Das Tool erzeugt das selbstentpackende 7-Zip-Archiv mit dem Namenspräfix `Package-`.

Der grundlegende Unterschied zu Variante 2 liegt im Prinzip der Änderungserfassung: Bei Variante 2 **erkennt** das Tool die Differenzen selbst, indem es den aktuellen Zustand mit einem gespeicherten Snapshot und der Änderungshistorie abgleicht (Pull-Prinzip). Bei Variante 3 **beschreibt** der Engineer die Änderung vorab deklarativ, und das Tool generiert daraus sämtliche Artefakte (Push-Prinzip). Der Vorteil des Push-Prinzips ist die vollständige Reproduzierbarkeit: Jedes Updatepaket lässt sich jederzeit aus seiner Metadaten-Datei identisch erneut erzeugen.

- **Vorteile:** Höchster Automatisierungsgrad und damit minimale manuelle Fehlerquellen. Vollständig standardisierte und reproduzierbare Updatepakete. Die Validierung verhindert unvollständige oder fehlerhafte Pakete bereits bei der Erstellung.
- **Nachteile:** Höchster Entwicklungsaufwand (Schema-Definition, Generator, Validierung). Erfordert disziplinierte Metadaten-Pflege während des Engineerings und teilweise Anpassungen im Engineering-Prozess. TypeScript/Node.js muss als zusätzliche Laufzeitumgebung auf den Engineering-Arbeitsplätzen bereitgestellt und gepflegt werden. Für den direkten Aufruf aus der **as1** ist ebenfalls eine **as1**-Erweiterung erforderlich.

Technische Details:

- **Metadaten-Schema:** Das Schema der Metadaten-Datei wird formal definiert (z.B. als JSON-Schema). Es definiert Pflicht- und Optionalfelder, erlaubte Wertebereiche und Referenzen auf Parameter bzw. Konfigurationsdateien. Gegen dieses Schema validiert das Tool jede Eingabedatei, bevor irgendein

Artefakt generiert wird. Ungültige Metadaten brechen die Paketerstellung mit konkreter Fehlermeldung ab.

- **SQL-Handling:** Da die Änderungsinformationen pro Parameter in der MSSQL-Datenbank hinterlegt sind, beschreibt die Metadaten-Datei die SQL-Statements vollständig deklarativ. Das Tool führt die Validierung der Statements syntaktisch gegen die Schema-Definition durch (Pflichtfelder, zulässige Anweisungen), ohne selbst eine Datenbankverbindung aufzubauen. Die Prüfung auf inhaltliche Korrektheit bleibt Aufgabe des Engineers.
- **Validierung als Prozessschritt:** Die Validierung ist ausdrücklich der erste Verarbeitungsschritt (vgl. Beschreibung, Punkt 1). Fehlerhafte, unvollständige oder widersprüchliche Metadaten verhindern die Paketerstellung vollständig, im Gegensatz zu Variante 1 und 2, wo unvollständige Pakete möglich sind und erst die geführte Bedienung (Variante 2) bzw. der Engineer (Variante 1) davor schützt.
- **install.config-Schema:** Die aus den Metadaten generierte `install.config` folgt der dokumentierten Struktur (vgl. Kapitel 6.1.4) und wird vor der Paketerstellung gegen das Schema validiert: erlaubte Elemente (`InstallInstructions` mit `Prerequisites`, `InstallationParameters`, `PreInstallation`, `Installation`, `PostInstallation`), nur die zwölf definierten Installationsschritte und konsistente Pfadangaben gemäss Paketstruktur. Die Zuordnung der Schritte zu den drei Phasen erfolgt über feste Zuordnungsfelder in der Metadaten-Datei. Die ReadMe wird analog zu Variante 2 als HTML-Datei erzeugt und über den Schritt `ShowHtml` nach der Installation automatisch angezeigt.
- **Bedienung:** Das Tool wird über die Kommandozeile aufgerufen und erhält die Metadaten-Datei als Argument. Eine grafische Oberfläche ist nicht vorgesehen, da der Prozess nach dem Erfassen der Metadaten ohne weitere Interaktion abläuft.
- **Paketierung:** Das Tool erzeugt das selbstentpackende 7-Zip-Archiv über die 7-Zip-Kommandozeile mit dem Namenspräfix `Package-`. Der abschliessende Schritt beinhaltet die Prüfung, dass das erzeugte Archiv dem vom **as1HotfixInstaller** geforderten Format entspricht (vgl. Kapitel 6.1.4).
- **Reproduzierbarkeit:** Da alle Artefakte ausschliesslich aus der Metadaten-Datei generiert werden, genügt die Aufbewahrung der Metadaten-Datei, um ein Paket zu einem späteren Zeitpunkt identisch neu zu erzeugen.

6.5 Evaluation der geeignetsten Variante

(Konzeptphase: Die Bewertung entscheidet, welche der konzipierten Varianten realisiert werden soll.)

6.5.1 Vorgehen und Bewertungskriterien

Zur Ermittlung der geeignetsten Variante wird eine Nutzwertanalyse durchgeführt [5]. Bewertet werden die Nullvariante (Beibehaltung des Ist-Zustands) sowie die drei Varianten aus Kapitel 6.4. Die Bewertungskriterien werden aus den Muss- und Kann-Zielen des Pflichtenhefts (Kapitel 4.1) abgeleitet. Die Gewichtung der Kriterien erfolgt auf einer Skala von 1 (niedrige Priorität) bis 10 (hohe Priorität).

Tabelle 13: Bewertungskriterien der Nutzwertanalyse

Nr.	Kriterium	Herleitung	Gewichtung
K1	Automatische Erkennung von Parameteränderungen zwischen zwei Softwarezeitpunkten	Muss-Ziel 1a	10
K2	Automatische Differenzerkennung aktiver Steuerungselemente zwischen zwei Softwarezeitpunkten	Muss-Ziel 1b	10
K3	Standardisierte Paketerstellung, **as1HotfixInstaller**-fähig ohne manuelle Nachbearbeitung	Muss-Ziel 1d, Erfolgskriterium	10
K4	Automatische ReadMe (Installations- und Änderungsdokumentation, nachvollziehbar)	Muss-Ziel 2b	7
K5	Fehlerreduktion durch Validierung bei der Paketerstellung	Muss-Ziel 2a	7
K6	Automatische Erkennung von Änderungen an definierten Datenbankobjekten	Kann-Ziel 1	4
K7	Bedienung: automatische Pfad-Erkennung oder Vorbelegung von Standardpfaden	Kann-Ziel 2	3
K8	Strukturierte Übersicht sämtlicher erkannter Änderungen	Kann-Ziel 3	2
K9	Entwicklungsaufwand im Rahmen der Diplomarbeit	Rahmenbedingung	8
K10	Wartbarkeit und Erweiterbarkeit des Tools	allgemein	5

Begründung der Gewichtung: Die Kriterien K1 bis K3 erhalten die maximale Gewichtung, da sie die Kern-Muss-Ziele der Arbeit betreffen: die automatische Erkennung der Änderungen (K1, K2) und die Installierbarkeit ohne Nachbearbeitung (K3). K3 wird gleich hoch gewichtet wie K1 und K2, weil das Erfolgskriterium "ohne manuelle Nachbearbeitung" die eigentliche Vertragsschnittstelle zum bestehenden **as1HotfixInstaller** ist. Ein Paket, das am Ende nicht installierbar ist, macht die vorherige Erkennung wertlos. K4 und K5 (Muss-Ziele 2a/2b) werden mit 7 gewichtet, da die ReadMe qualitativ nachvollziehbar sein muss, aber nicht jede Information zwingend vollautomatisch entstehen muss. Die Kann-Ziele K6 bis K8 werden mit 4 bzw. 3 bzw. 2 gewichtet, entsprechend ihrer Priorisierung im Pflichtenheft. Der Entwicklungsaufwand (K9) wird mit 8

gewichtet: Er ist eine ernstzunehmende Rahmenbedingung der Diplomarbeit, aber eine Nebenbedingung und nicht das Ziel der Lösung. Die Wartbarkeit (K10) erhält eine mittlere Gewichtung von 5.

Nicht differenzierende Kriterien: Die Kommandozeilenfähigkeit (Muss-Ziel 1c), die Kompatibilität mit den **as1**-Versionen V6.0 und V6.1 sowie die erforderliche **as1**-Erweiterung für den Startzeilenaufruf (vgl. Kapitel 6.4) sind für alle drei Varianten identisch erfüllbar bzw. erforderlich und werden daher nicht als Bewertungskriterien aufgenommen.

6.5.2 Bewertungsmatrix

Die Bewertung jedes Kriteriums erfolgt je Variante auf einer Skala von 0 bis 5. Eine feinere Skala (z.B. 0 bis 10) wurde bewusst verworfen: Bei nur vier Varianten und zehn Kriterien würde sie eine Scheingenauigkeit erzeugen, die sich begründungstechnisch nicht halten lässt. Die fünfstufige Skala erzwingt zudem die saubere Trennung zwischen der Wichtigkeit eines Kriteriums (abgebildet in der Gewichtung) und der Güte der Erfüllung durch eine Variante (abgebildet in der Bewertung).

Tabelle 14: Bewertungsskala der Nutzwertanalyse

Stufe	Bedeutung
0	nicht erfüllt
1	kaum erfüllt
2	teilweise erfüllt
3	überwiegend erfüllt
4	gut erfüllt
5	vollständig erfüllt

Die Wertigkeit ergibt sich aus dem Produkt der Kriteriumsbewertung und der Gewichtung. Der maximale Nutzwert einer Variante beträgt damit 330 Punkte (Summe der Gewichtungen 66 × Faktor 5). Die Nullvariante dient als Referenz: Sie zeigt den Nutzen jeder Lösungsvariante gegenüber dem heutigen, manuellen Vorgehen.

Bewertung der Varianten je Kriterium:

Tabelle 15: Bewertung der Varianten je Kriterium

Nr.	Gew.	Kriterium	Nullvariante	Variante 1	Variante 2	Variante 3
K1	10	Erkennung Parameteränderungen	0	4	5	0
K2	10	Differenzerkennung Steuerungselemente	0	5	5	0
K3	10	Paket HotfixInstaller-fähig ohne Nachbearbeitung	0	4	5	3
K4	7	Automatische ReadMe	0	1	5	5
K5	7	Fehlerreduktion durch Validierung	0	1	4	5
K6	4	Erkennung Datenbankobjekt-Änderungen	0	1	3	1
K7	3	Pfad-Erkennung / Vorbelegung	0	2	4	1
K8	2	Strukturierte Änderungsübersicht	0	1	4	3
K9	8	Entwicklungsaufwand (hohe Bewertung = geringer Aufwand)	5	4	2	1
K10	5	Wartbarkeit / Erweiterbarkeit	0	2	4	3

Begründung der Bewertungen (je Zelle):

- **K1/K2:** Die Nullvariante erkennt nichts automatisch (0). Bei K2 erkennen die Varianten 1 und 2 beide vollständig automatisch (je 5): Die Differenzerkennung der aktiven Steuerungselemente läuft vollständig in den XML-Dateien, der Snapshot-Vergleich deckt sie ohne manuellen Anteil ab. Bei K1 erhält Variante 1 die Bewertung 4, weil der Parameter-Teil (Auswertung der Änderungshistorie) über ein manuell ergänztes Template läuft. Variante 2 wertet die Änderungshistorie der MSSQL-Datenbank automatisch aus (5). Variante 3 erhält bei beiden Kriterien 0, da sie keine Änderungen erkennt, sondern auf die deklarative Beschreibung durch den Engineer setzt. Sie erfüllt damit das Muss-Ziel der automatischen Erkennung strukturell nicht.
- **K3:** Variante 1 und 2 erzeugen beide die `install.config` gemäss Schema und paketieren automatisch. Variante 1 erhält 4, weil das SQL-Template vor der Paketierung manuell vervollständigt werden muss, was streng genommen eine Nachbearbeitung darstellt. Variante 3 erzeugt das Paket zwar vollständig aus den Metadaten, das Ergebnis hängt aber vollständig von der Qualität der Eingabedatei ab. Ohne Prüfung der tatsächlichen Systemzustände bleibt eine Restunsicherheit, ob das Paket den realen Änderungen entspricht (3).
- **K4:** Variante 1 füllt nur eine ReadMe-Vorlage manuell (1) und behebt damit die Schwachstelle der personengebundenen Dokumentation nicht. Die Varianten 2 und 3 erzeugen die ReadMe automatisch aus den Metadaten bzw. Paketdaten und binden sie über den ShowHtml-Schritt aktiv in die Installation ein (je 5).
- **K5:** Variante 1 validiert weder Eingaben noch Inhalte (1). Variante 2 verhindert durch die geführte Bedienung mit Eingabevalidierung unvollständige Pakete weitgehend (4), erhält aber nicht die

Höchststufe, weil die Vollständigkeit der erkannten Änderungen weiterhin von der Pflege der Snapshots abhängt. Variante 3 verhindert unvollständige Pakete bereits durch die Schema-Validierung der Metadaten vollständig (5), allerdings kann die Validierung nicht prüfen, ob die Beschreibung der tatsächlichen Anpassung entspricht.

- **K6:** Keine Variante enthält die Erkennung von Datenbankobjekt-Änderungen im geforderten Umfang. Variante 2 erhält 3, da die Auswertung der Änderungshistorie mit den erweiterten Auswertungen (das schemabasierte Auswerten weiterer Datenbankobjekte) am nächsten liegt. Variante 1 (1) und Variante 3 (1) müssten hierfür manuell erweitert werden.
- **K7:** Variante 1 hat die Pfade fest im Skript hinterlegt, was im Standardfall funktioniert, bei Abweichungen aber eine Skriptanpassung erfordert (2). Variante 2 kann Pfade automatisch erkennen oder im GUI vorbelegen und gleichzeitig Überschreiben erlauben (4). Variante 3 deklariert alle Pfade manuell in der Metadaten-Datei (1).
- **K8:** Variante 2 kann aus den erkannten Änderungen automatisch eine strukturierte Übersicht erzeugen (4). Variante 3 kann die strukturierte Übersicht aus den Metadaten ableiten (3). Variante 1 erzeugt keine strukturierte Übersicht (1).
- **K9:** Die Nullvariante verursacht keinen Entwicklungsaufwand (5). Variante 1 ist mit einem einzelnen PowerShell-Skript am günstigsten (4, nicht 5, weil auch ein Skript getestet und dokumentiert werden muss). Variante 2 erfordert eine vollständige Anwendung mit GUI, Datenschicht und Tests (2). Variante 3 erfordert zusätzlich Schema-Definition, Generator und die Node.js-Laufzeitumgebung (1).
- **K10:** Variante 1 fällt bei wachsender Skriptlänge in der Wartbarkeit ab (2). Variante 2 bietet durch die Schichtentrennung die beste Wartbarkeit (4). Variante 3 ist gut erweiterbar, aber an die Laufzeitumgebung und die Schema-Pflege gebunden (3).

Nutzwerte (Summe der Wertigkeiten):

Tabelle 16: Nutzwerte der Varianten

Rang	Variante	Nutzwert	Anteil vom Maximum (330)
1	Variante 2: Integrales GUI-Tool (C#/.NET)	**281**	85 %
2	Variante 1: Skriptbasierte Automatisierung (PowerShell)	198	60 %
3	Variante 3: Metadaten-getriebene Vollautomatisierung	136	41 %
4	Nullvariante (Referenz)	40	12 %

Interpretation: Variante 2 erreicht mit deutlichem Abstand den höchsten Nutzwert. Sie erfüllt die hoch gewichteten Muss-Kriterien K1 bis K3 vollständig und behandelt mit der automatischen ReadMe (K4) und der Validierung (K5) auch die Qualitätsschwachstellen des Ist-Prozesses. Variante 1 erreicht rund 70 % des Nutzwerts von Variante 2 bei deutlich geringerem Entwicklungsaufwand. Sie erkennt die XML-Differenzen zwar vollständig automatisch (K2: 5), bleibt aber bei der Parameter-Erkennung (K1), der Dokumentation (K4) und der Fehlervermeidung (K5) hinter den Muss-Zielen zurück. Variante 3 scheitert strukturell an den Muss-Zielen der automatischen Erkennung (K1/K2 mit Bewertung 0) und verzeichnet zudem den höchsten Entwicklungsaufwand. Ihr Nutzwert liegt trotz Vollautomatisierung der ReadMe und Validierung unter dem von

Variante 1. Die Nullvariante bestätigt schliesslich den Handlungsbedarf: Ohne Tool bleiben die Muss-Ziele unerfüllt.

6.5.3 Sensitivitätsanalyse

Die Sensitivitätsanalyse prüft, wie stabil das Ergebnis der Nutzwertanalyse gegenüber Änderungen der Gewichtung ist. Dazu werden zwei Szenarien gebildet, die jeweils eine bewusst einseitige Verschiebung der Prioritäten darstellen. Die Szenarien sind bewusst einseitig gewählt, weil nicht erwartet wird, dass diese Gewichtungen der tatsächlichen Priorisierung entsprechen, sondern weil sie die Grenzen der Entscheidung aufzeigen. Gewinnt bereits eine andere Variante unter einer einseitigen Gewichtung, ist das Gesamtergebnis nicht belastbar.

Szenario A: Maximale Gewichtung des Entwicklungsaufwands (K9 von 8 auf 10). Dieses Szenario testet die kritischste Rahmenbedingung der Diplomarbeit. Was wäre, wenn der Realisierungsaufwand das wichtigste Kriterium wäre? Es begünstigt die Nullvariante und Variante 1 am stärksten.

Szenario B: Maximale Gewichtung der Qualität (K4 und K5 von 7 auf 10). Dieses Szenario testet die Gegenrichtung: Was wäre, wenn die Dokumentationsqualität und die Fehlervermeidung gleich wichtig wären wie die Erkennung selbst? Es begünstigt Variante 3 am stärksten, da sie bei beiden Kriterien die Höchstbewertung erhält.

Tabelle 17: Szenarien der SensitivitÃ¤tsanalyse

Variante	Basis	Szenario A (K9=10)	Szenario B (K4=K5=10)
Variante 2 (C#/.NET)	**281**	**285**	**308**
Variante 1 (PowerShell)	198	206	204
Variante 3 (Metadaten)	136	138	166
Nullvariante	40	50	40

Ergebnis der Sensitivitätsanalyse: Variante 2 erreicht in allen drei Gewichtungen den höchsten Nutzwert. Das Ergebnis ist damit als stabil zu bewerten:

- **Szenario A** (Entwicklungsaufwand maximal): Variante 2 gewinnt mit 285 Punkten, obwohl sie beim aufgewerteten Kriterium K9 nur die Bewertung 2 erhält. Der Vorsprung schmilzt gegenüber dem Basis-Szenario nicht, sondern wächst sogar leicht (285 zu 206 statt 281 zu 198), weil die Nullvariante als einzige den Kriterien-Katalog vollständig ignoriert und Variante 1 beim Aufwand nur bedingt punktet. Selbst eine maximale Aufwertung des Entwicklungsaufwands ändert die Rangfolge nicht.
- **Szenario B** (Qualität maximal): Variante 3 kann ihren einzigen strukturellen Vorteil (vollautomatische ReadMe und Validierung) voll ausspielen und legt von 136 auf 166 Punkte zu. Der Abstand zu Variante 2 bleibt mit 142 Punkten (308 zu 166) zwar weiterhin deutlich, verringert sich gegenüber der Basisanalyse aber leicht von 145 auf 142 Punkte, da Variante 3 bei den beiden aufgewerteten Kriterien die Höchstbewertung erhält und Variante 2 dort nur 4 bzw. 5 erreicht. Der Vorsprung von Variante 2 bleibt auf den Muss-Kriterien K1 bis K3 begründet: Variante 3 kann die Lücke aus K1/K2 (Bewertung 0 bei Gewicht 10) durch keine Gewichtung schliessen, solange die automatische Erkennung als Muss-Ziel erhalten bleibt.

Schlussfolgerung: Die Wahl von Variante 2 ist unter allen geprüften Gewichtungen stabil. Der Entscheid hängt einzig an der Prämisse, dass die automatische Erkennung der Änderungen (K1/K2) ein Muss-Ziel bleibt, wie es das Pflichtenheft definiert. Erst wenn diese Prämisse aufgegeben würde, könnte eine andere Variante konkurrenzfähig werden. Da Variante 3 ohne die automatische Erkennung aber genau die Kernanforderung der Arbeit verliert, bestätigt die Sensitivitätsanalyse die Entscheidung für Variante 2 zusätzlich.

7. Ausarbeitung der präferierten Variante

Dieses Kapitel folgt der Gliederung "Planung -> Resultat mit Feedback zur Planung": Die Abschnitte 7.1 bis 7.5 beschreiben die Planung der Umsetzung (Anforderungsprofil, Phasenplan, SWOT, Risiko-Analyse, Kosten-Nutzen). Abschnitt 7.6 dokumentiert anschliessend das realisierte Resultat und begründet die Abweichungen, die während der Realisierung bewusst entschieden wurden.

7.1 Softwareanforderungsprofil

(Planung (Soll): Die Anforderungen entstanden vor der Realisierung und blieben während der Umsetzung unverändert. Abweichungen sind in 7.6.6 begründet.)

Vorgehen und Notation

Die Anforderungen werden aus den Muss- und Kann-Zielen des Pflichtenhefts (Kapitel 4.1) und den Erkenntnissen der Ist-Analyse (Kapitel 6.1) abgeleitet. Sie sind nach funktionalen (was das Tool tut) und nicht-funktionalen Anforderungen (wie es sich verhalten muss) gegliedert. Jede Anforderung hat eine eindeutige Nummer, einen Verweis auf ihre Herkunft und ein Testkriterium, anhand dessen die Erfüllung überprüfbar ist.

Die Priorisierung folgt dem Schema **MUSS / SOLL / KANN**:

- **MUSS:** Anforderungen, die aus den Muss-Zielen des Pflichtenhefts resultieren. Ohne sie ist die Arbeit in ihrem Kernziel nicht erfolgreich.
- **SOLL:** Anforderungen mit hohem Nutzen, die bei knappen Ressourcen zurückgestellt werden können, ohne die Muss-Ziele zu gefährden.
- **KANN:** Anforderungen aus den Kann-Zielen des Pflichtenhefts. Ihre Erfüllung verbessert die Lösung, ihr Fehlen schmälert den Erfolg nicht.

Der Realisierungsumfang der vorliegenden Arbeit ist bewusst auf die MUSS- und ausgewählte SOLL-Anforderungen begrenzt. Dies ist eine bewusste Priorisierungsentscheidung auf Basis des verfügbaren Zeitbudgets (vgl. Phasenplan) und wird in der SWOT-Analyse und im Ausblick reflektiert.

7.1.1 Funktionale Anforderungen

Snapshot-Verwaltung

Tabelle 18: Anforderungen: Snapshot-Verwaltung

Nr.	Anforderung	Priorität	Herkunft	Testkriterium
F-01	Das Tool erstellt auf Anweisung einen Snapshot der lokalen XML-Konfigurationsdateien der **as1** (`Fieldbus.config`, `AutomationControl.config`, `ProcessView.config`). Snapshots werden pro Anlage gruppiert (Ordner benannt nach der Anlage, Anlagenname aus der Datenbank) und innerhalb der Anlage in zeitlich benannten Unterordnern abgelegt.	MUSS	Ist-Analyse 6.1.2 (Informationsquellen), Variante 2, Baustein 1	Snapshot-Aufruf erzeugt Anlagenordner mit zeitlich benanntem Unterordner und den drei Dateien. Die Dateien sind mit den Quelldateien identisch (Hash-Vergleich).
F-02	Jedem Snapshot wird eine Metadaten-Datei (`snapshot.json`) beigelegt mit mindestens: Erstellungsdatum, Anlage, erstellender Engineer, Kommentar.	MUSS	Variante 2, Baustein 1	`snapshot.json` existiert, ist gültiges JSON und enthält alle Pflichtfelder.
F-03	Das Tool listet vorhandene Snapshots auf und erlaubt die Auswahl eines Referenz-Snapshots für den Vergleich.	MUSS	Variante 2, Baustein 2	CLI-Aufruf listet Snapshots mit Datum, Anlage und Kommentar.

Differenzerkennung XML (aktive Steuerungselemente)

Tabelle 19: Anforderungen: Differenzerkennung XML

Nr.	Anforderung	Priorität	Herkunft	Testkriterium
F-04	Das Tool erkennt automatisch, welche XML-Konfigurationsdateien sich zwischen dem Referenz-Snapshot und dem aktuellen Zustand geändert haben (Hash-Vergleich zur Vorfilterung).	MUSS	Muss-Ziel 1b, Ist-Analyse 6.1.1	Bei Testdaten mit einer bekannten Änderung wird genau diese Datei erkannt.
F-05	Für jede geänderte Datei erzeugt das Tool die vollständige geänderte Datei als `CopyFile`-Payload (Installation via `CopyFile`).	MUSS	Ist-Analyse 6.1.4 (install.config-Schema)	Erzeugte Datei ist identisch mit der geänderten Quelldatei.

Nr.	Anforderung	Priorität	Herkunft	Testkriterium
F-06	Für jede geänderte Datei erzeugt das Tool eine XML-Modifikationsdatei mit `Add`-, `Remove`- und `Replace`-Anweisungen inklusive XPath-Referenzen (Installation via `ModifyXml`).	SOLL	Ist-Analyse 6.1.4	Modifikationsdatei ist wohlgeformtes XML. Bei Anwendung auf den Referenz-Snapshot entsteht die geänderte Datei (Round-Trip-Test).

Differenzerkennung Parameter (Datenbank)

Tabelle 20: Anforderungen: Differenzerkennung Parameter

Nr.	Anforderung	Priorität	Herkunft	Testkriterium
F-07	Das Tool verbindet sich über eine eigene, nur lesende Verbindung mit der MSSQL-Datenbank und liest die Änderungshistorie ab dem Erstellungsdatum des Referenz-Snapshots aus.	MUSS	Muss-Ziel 1a, Ist-Analyse 6.1.1 (Änderungshistorie), Variante 2, Baustein 3	Historie-Einträge ab Snapshot-Datum werden vollständig angezeigt.
F-08	Das Tool generiert aus den erkannten Parameteränderungen ein SQL-Skript, das die Änderungen auf dem Kundensystem reproduzierbar ausführt.	MUSS	Muss-Ziel 1a	Generiertes SQL-Skript ist syntaktisch gültig und umfasst alle erkannten Änderungen (Abgleich mit Testdaten).
F-09	Das Tool erlaubt den manuellen Abgleich mit dem Parameterexport der **as1**, um die erkannten Änderungen zu verifizieren.	KANN	Ist-Analyse 6.1.2	Parameterexport kann geladen und angezeigt werden.

Paketgenerierung

Tabelle 21: Anforderungen: Paketgenerierung

Nr.	Anforderung	Priorität	Herkunft	Testkriterium
F-10	Das Tool generiert die `install.config` gemäss dem dokumentierten Schema (`InstallInstructions` mit `Prerequisites`, `InstallationParameters`, `PreInstallation`, `Installation`, `PostInstallation`, nur die definierten Installationsschritte).	MUSS	Muss-Ziel 1d, Ist-Analyse 6.1.4	install.config ist wohlgeformt, schemakonform und enthält alle erkannten Änderungen als Schritte.
F-11	Das Tool validiert die erzeugte `install.config` vor der Paketierung: erlaubte Elemente, gültige Schritt-Typen, konsistente Pfadangaben gemäss Paketstruktur.	SOLL	Ist-Analyse 6.1.4, Variante 2, Baustein 4	Ungültige Konfigurationen werden mit konkreter Fehlermeldung abgewiesen.
F-12	Das Tool erzeugt eine ReadMe als HTML-Datei aus den Metadaten und Paketdaten (erkannte Änderungen, Installationsinformationen) und bindet sie über den Schritt `ShowHtml` in die PostInstallation ein.	MUSS	Muss-Ziel 2b, Ist-Analyse 6.1.4 (ShowHtml)	ReadMe wird nach der Installation automatisch angezeigt und enthält die erkannten Änderungen.
F-13	Das Tool paketiert sämtliche Dateien als selbstentpackendes 7-Zip-Archiv mit dem Namenspräfix `Package-` gemäss der vorgegebenen Paketstruktur.	MUSS	Muss-Ziel 1d, Ist-Analyse 6.1.4	Entpacktes Archiv entspricht der Strukturvorgabe. Der **as1HotfixInstaller** akzeptiert das Paket.
F-14	Das Tool prüft den Erfolg der Paketierung (Existenz und Integrität des Archivs) und meldet das Resultat.	SOLL	Variante 2, Baustein 4	Fehlerhafte Paketierung wird mit Fehlermeldung abgebrochen.

Bedienung

Tabelle 22: Anforderungen: Bedienung

Nr.	Anforderung	Priorität	Herkunft	Testkriterium
F-15	Das Tool ist vollständig über Kommandozeilenparameter bedienbar (Snapshot erstellen, vergleichen, Paket erstellen) ohne Interaktion.	MUSS	Muss-Ziel 1c	Kompletter Durchlauf (Snapshot bis Paket) ist ohne Benutzereingabe möglich.
F-16	Das Tool unterstützt die automatische Erkennung oder Vorbelegung von Standardpfaden (XML-Konfigurationspfad, Datenbankverbindung) mit der Möglichkeit zum Überschreiben.	SOLL	Kann-Ziel 2	Standardfall läuft ohne Pfadangaben. Abweichende Pfade sind per Parameter möglich.
F-17	Das Tool bietet eine geführte grafische Oberfläche für den kompletten Ablauf (Snapshot-Auswahl, Validierung, Fortschritt).	KANN	Variante 2, Baustein 5 (konzipiert, in dieser Arbeit nicht realisiert)	Entfällt für diese Arbeit. Konzeptionell in der Architektur vorgesehen (Schichtentrennung).

Erweiterte Funktionen

Tabelle 23: Anforderungen: Erweiterte Funktionen

Nr.	Anforderung	Priorität	Herkunft	Testkriterium
F-18	Das Tool erkennt Änderungen an definierten Datenbankobjekten (Tabellen, Sichten, Prozeduren) und berücksichtigt sie im Paket.	KANN	Kann-Ziel 1	Definitionsliste der DB-Objekte ist konfigurierbar. Testobjekt wird erkannt.
F-19	Das Tool erzeugt eine strukturierte Übersicht aller erkannten Änderungen (als Tabelle im ReadMe).	KANN	Kann-Ziel 3	ReadMe enthält tabellarische Übersicht.

7.1.2 Nicht-funktionale Anforderungen

Tabelle 24: Nicht-funktionale Anforderungen

Nr.	Anforderung	Priorität	Herkunft	Testkriterium
NF-01	Das Tool läuft unter Windows mit den **as1**-Softwareversionen V6.0 und V6.1.	MUSS	Muss-Erfolgskriterium	Test auf beiden Versionen erfolgreich (bzw. auf verfügbarer Version mit Kompatibilitätsnachweis).
NF-02	Das Tool nimmt ausschliesslich lesende Zugriffe auf die MSSQL-Datenbank vor. Schreibende Zugriffe erfolgen nicht.	MUSS	Variante 2 (Datenschutz), Ist-Analyse 6.1.1	Code-Review der Datenschicht. SQL-Server-Protokollierung während des Testlaufs zeigt nur lesende Zugriffe.
NF-03	Das Tool verändert die Quelldateien (XML-Konfigurationen) nicht und löscht keine Daten.	MUSS	Sicherheitsanforderung aus Ist-Analyse 6.1.3 (Sachschadenrisiko)	Quelldateien sind nach dem Durchlauf unverändert (Hash-Vergleich).
NF-04	Fehler (fehlender Snapshot, keine DB-Verbindung, ungültige Pfade) werden mit verständlichen Meldungen quittiert und brechen den Vorgang kontrolliert ab.	SOLL	Muss-Ziel 2a (Fehlerreduktion)	Herbeigeführte Fehlerfälle erzeugen eine klare Meldung, keine unbehandelten Ausnahmen.
NF-05	Die Codebasis ist schichtengetrennt (Bedienung, Ablaufsteuerung, Datenzugriff), sodass die GUI nachrüstbar ist, ohne die Kernlogik zu ändern.	SOLL	Variante 2 (GUI-Architektur), K10 Wartbarkeit	Kernlogik ist ohne GUI ausführbar (belegt durch CLI-Nutzung).
NF-06	Das Tool ist mit vertretbarem Aufwand um die Kann-Funktionen (F-16 bis F-19) und die GUI (F-17) erweiterbar.	SOLL	K10 Erweiterbarkeit	Architektur-Doku zeigt Erweiterungspunkte.

7.1.3 Umfangsentscheidung und Abgrenzung

Für die Realisierung innerhalb des Zeitbudgets der Diplomarbeit gilt folgende Umfangsentscheidung:

Tabelle 25: Umfangsentscheidung und Abgrenzung

Umfang	Anforderungen
Wird realisiert	F-01 bis F-05, F-07, F-08, F-10, F-12, F-13, F-15, NF-01 bis NF-05
Wird realisiert, wenn Zeit bleibt	F-06, F-11, F-14, F-16
Konzipiert, nicht realisiert	F-09, F-17, F-18, F-19, NF-06

Die Entscheidung basiert auf den Prioritäten des Pflichtenhefts (Muss-Ziele zuerst) und dem verfügbaren Zeitbudget. Die nicht realisierten Anforderungen sind in der Architektur vorgesehen (vgl. NF-05, NF-06) und werden im Ausblick der Arbeit als Weiterentwicklung beschrieben. Damit bleibt der Erfolg der Arbeit an den Muss-Erfolgskriterien messbar, unabhängig vom Umfang der KANN-Anteile. Die Abweichungen, die sich während der Realisierung ergaben, sind in Abschnitt 7.6.6 dokumentiert.

7.2 Phasenplan zur Umsetzung

(Planung (Soll): Der Phasenplan beschreibt den geplanten Verlauf. Der tatsächliche Verlauf ist in Abschnitt 7.2.5 beschrieben.)

7.2.1 Vorgehen

Die Realisierung ist in vier Phasen mit je einem Meilenstein gegliedert. Jede Phase liefert ein überprüfbares Zwischenergebnis, das anhand des Softwareanforderungsprofils getestet wird. Die Phasen sind bewusst so geschnitten, dass nach jeder Phase ein funktionsfähiger Zwischenstand existiert ("inkrementeller Aufbau"): Scheint die Zeit knapp zu werden, kann nach Phase 2 jederzeit ein funktionierendes, wenn auch unvollständiges, Ergebnis abgegeben werden. Die Reihenfolge folgt der Datenflusskette des Tools: Erst die Daten erzeugen (Snapshot), dann erkennen (Differenzen), dann ausgeben (Paket).

7.2.2 Übersicht

Tabelle 26: Übersicht der Realisierungsphasen

Phase	Inhalt	Dauer	Meilenstein
P1: Fundament	Projekt-Setup, Architektur, CLI-Gerüst, Snapshot-Verwaltung	2 Tage	M1: Snapshot erstellen und listen per CLI funktioniert (F-01, F-02, F-03)
P2: Erkennung	XML-Differenzerkennung, DB-Historie, SQL-Generierung	3 Tage	M2: Alle Änderungen zwischen Snapshot und Ist werden erkannt (F-04, F-05, F-07, F-08)
P3: Paket	install.config, ReadMe, Paketierung	3 Tage	M3: Vollständiges, HotfixInstaller-fähiges Paket wird erzeugt (F-10, F-12, F-13)
P4: Integration & Test	Komplettdurchlauf, Tests, Fehlerbehebung, Doku	2 Tage	M4: Muss-Erfolgskriterien erfüllt, dokumentiert (NF-01 bis NF-03, Testprotokoll)

7.2.3 Detailplanung

Phase 1: Fundament (2 Tage)

Aufbau der Solution mit Schichtenarchitektur (vgl. NF-05):

```
As1UpdateTool.sln
├── As1UpdateTool.Core        (Kernlogik: Snapshot, Differenz, Paket)
├── As1UpdateTool.Data        (DB-Zugriff, nur lesend)
├── As1UpdateTool.Cli         (Kommandozeilen-Frontend)
└── As1UpdateTool.Tests       (Unit-Tests der Kernlogik)
```

- Tag 1: Solution und Projekte anlegen, NuGet-Pakete einbinden (CommandLine, SevenZip, ggf. Moq/NUnit für Tests), CLI-Argumentparsing mit Subcommands (`snapshot`, `diff`, `package`, `list`), erste Version der Konfiguration (Pfade, DB-Verbindungsstring)
- Tag 2: Snapshot-Verwaltung vollständig: Anlagenordner (Anlagenname aus der Datenbank) mit zeitlich benanntem Unterordner anlegen, drei XML-Konfigurationsdateien kopieren, `snapshot.json` schreiben und lesen, Snapshot-Liste ausgeben (gruppiert nach Anlage)

M1-Kontrolle (Ende Tag 2): `As1UpdateTool.Cli snapshot create` erzeugt gültigen Snapshot. `As1UpdateTool.Cli snapshot list` zeigt ihn. Damit sind F-01 bis F-03 erfüllt und testbar.

Phase 2: Erkennung (3 Tage)

- Tag 3: Hash-Vergleich der XML-Dateien (Snapshot vs. aktuell), geänderte Dateien als `CopyFile`-Payload bereitstellen (F-04, F-05). Unit-Test mit künstlichen Testdaten: eine Datei geändert, eine unverändert, eine neu. Erwartungsbild definiert das Testkriterium.
- Tag 4: DB-Zugriff: SqlConnection (nur lesend, vgl. NF-02), Auslesen der Änderungshistorie ab Snapshot-Datum, Datenmodell für Parameteränderungen
- Tag 5: SQL-Skript-Generierung aus den erkannten Parameteränderungen (F-08). Verifikation gegen einen manuellen Parameterexport als Gegenprobe.

M2-Kontrolle (Ende Tag 5): `As1UpdateTool.Cli diff --snapshot <name>` liefert die vollständige Liste der Änderungen: geänderte XML-Dateien, Parameteränderungen aus der DB-Historie, generiertes SQL-Skript.

Phase 3: Paket (3 Tage)

- Tag 6: install.config-Generierung aus den Ergebnissen der Phase 2: Phasen-Zuordnung (PreInstallation/Installation/PostInstallation), Schritte für SQL-Ausführung (RunSql) und Dateikopieren (CopyFile), ReadMe-Einbindung (ShowHtml)
- Tag 7: ReadMe-Generierung als HTML aus den Metadaten: erkannte Änderungen strukturiert aufbereitet, Installationsinformationen, ShowHtml-Verweis
- Tag 8: Paketierung: Ordnerstruktur gemäss Ist-Analyse 6.1.4, Aufruf der 7-Zip-SFX-Erstellung, Namenspräfix `Package-`, Kontrolle des Ergebnisses

M3-Kontrolle (Ende Tag 8): `As1UpdateTool.Cli package --snapshot <name>` erzeugt ein `Package-*.exe`-Archiv, das der **as1HotfixInstaller** ohne Nachbearbeitung akzeptiert.

Phase 4: Integration & Test (2 Tage)

- Tag 9: Komplettdurchlauf auf echten/nachgebauten Projektdaten: Snapshot-> manuelle Änderung ->

Diff -> Paket -> Installation mit **as1HotfixInstaller** auf Testsystem. Fehlerbehebung. Ggf. SOLL-Anforderungen (F-06, F-11) nachrüsten, falls Zeit.

- Tag 10: Testprotokoll anhand des Anforderungsprofils (jede realisierte Anforderung mit Resultat), Sicherheits-Verifikation NF-02/NF-03, Doku-Kapitel ergänzen (Realisierung, Tests), Screenshots für die Arbeit

M4-Kontrolle (Ende Tag 10): Alle MUSS-Anforderungen erfüllt und im Testprotokoll dokumentiert. Das Erfolgskriterium "Test anhand eines bestehenden Kundenprojekts" ist erfüllt bzw. mit der verfügbaren Umgebung umgesetzt.

7.2.4 Puffer und Risikobezug

Der Plan enthält bewusst keine expliziten Puffertage: Der inkrementelle Aufbau ist der Puffer. Kann Phase 4 nicht wie geplant abgeschlossen werden, existiert mit M3 bereits ein vollständig funktionierendes Tool für den Kernprozess. Die SOLL-Anforderungen (F-06, F-11, F-14, F-16) sind in den Phasen 2-3 vorgesehen, fallen aber als erste weg, ohne ein MUSS zu gefährden, entsprechend der Gegenmassnahme zu R7 der Risikoanalyse.

Der Plan deckt die identifizierten Projektrisiken direkt ab: Der frühe DB-Zugriff (Tag 4) entspricht der Gegenmassnahme zu R6 (Berechtigungen früh klären). Der Test mit dem **as1HotfixInstaller** (Tag 9) entspricht R3 (Testsystem früh abklären, spätestens zu Beginn von Phase 4 muss die Testumgebung verfügbar sein).

7.2.5 Ist-Verlauf

Die Realisierung verlief im Wesentlichen gemäss Phasenplan. Alle vier Meilensteine wurden erreicht.

- **Phase 1 (M1)** wurde termingerecht abgeschlossen. Eine Ergänzung gegenüber der Detailplanung erwies sich als notwendig: Die NuGet-Pakete CommandLine und SevenZip kamen nicht zum Einsatz (vgl. Abschnitt 7.6.6).
- **Phase 2 (M2)** wurde vollständig einschliesslich der Datenbank-Spur abgeschlossen. Die CTE-basierte Abfrage der Änderungshistorie inklusive Auflösung der Parameterpfade konnte innerhalb der geplanten Zeit realisiert werden. In dieser Phase fiel auch der Entscheid zur Abweichung F-08 (ParamTransfer-Import statt generiertem SQL-Skript), der in Abschnitt 7.6.6 begründet ist.
- **Phase 3 (M3)** verlief zügiger als geplant, sodass die beiden SOLL-Anforderungen F-06 (ModifyXml-Operationen) und F-14 (Paketprüfung) bereits innerhalb der Phase realisiert und getestet werden konnten.
- **Phase 4 (M4)** bestätigte im Realversuch die Architektur-Annahmen (SFX-Archivstruktur, Verhalten des **as1HotfixInstaller**) und deckte eine Integrations-Erkenntnis auf, die von Unit-Tests prinzipbedingt nicht sichtbar war: einen Typ-Mismatch zwischen SQL-Spalte (Guid) und Lesezugriff, der behoben wurde. Die Erkenntnisse aus dem Realversuch sind in Abschnitt 7.6.5 festgehalten.

7.3 SWOT-Analyse

(Planung (Soll): Die Analyse bewertet die geplante Lösung vor der Realisierung.)

Die SWOT-Analyse bewertet die gewählte Lösungsvariante 2 (C#/.NET-Tool mit Snapshot-Vergleich, vgl. Kapitel 6.4.2) vor Beginn der Realisierung. Sie gliedert die Bewertung in interne Faktoren (Stärken und Schwächen der Variante selbst) und externe Faktoren (Chancen und Risiken aus dem Umfeld der Arbeit).

Die folgende Übersicht zeigt die Vier-Felder-Struktur der SWOT-Analyse mit den wichtigsten Punkten je Quadrant. Die ausführliche Begründung der einzelnen Punkte folgt in den Abschnitten 7.3.1 und 7.3.2.

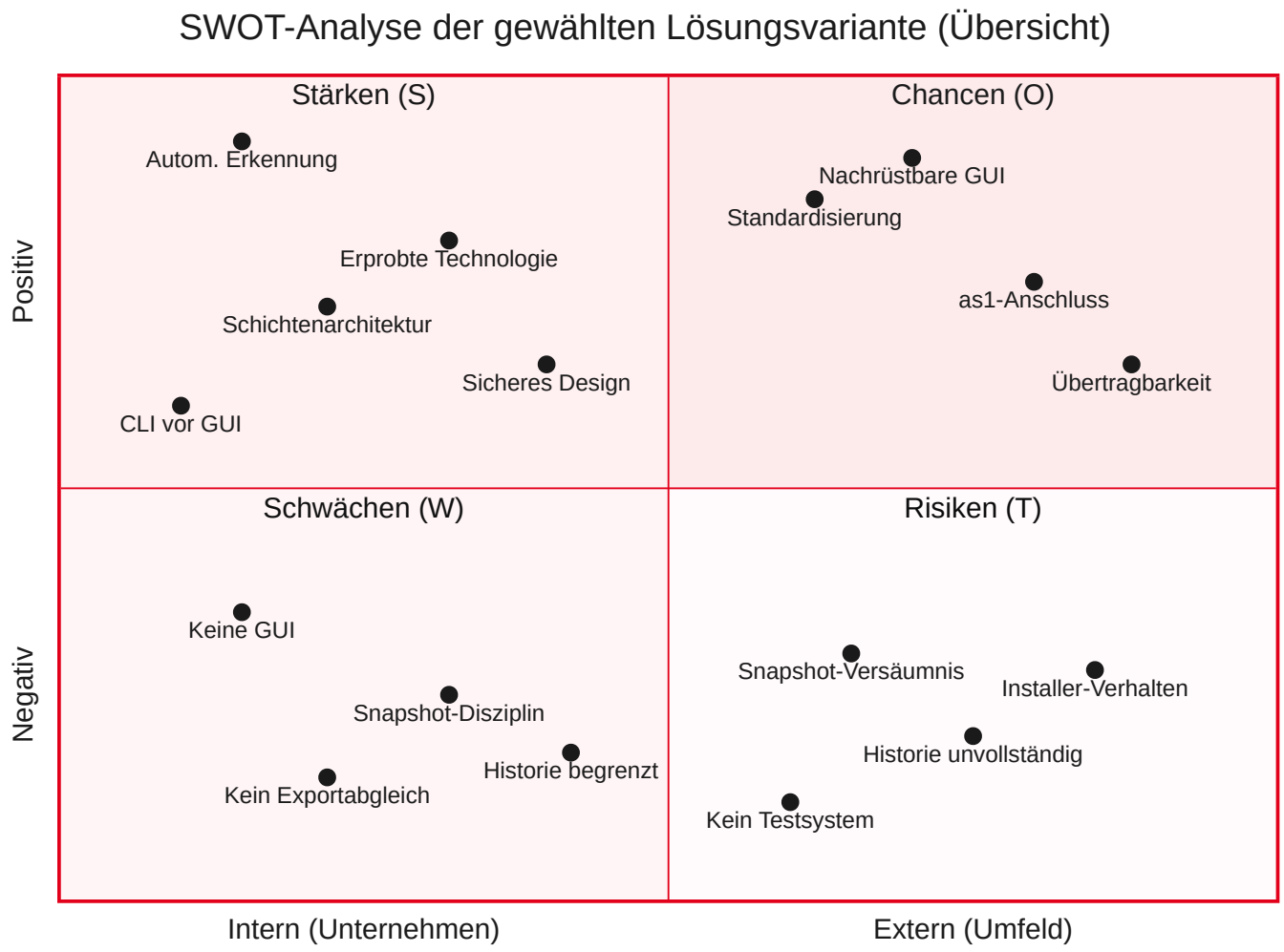

Abbildung 15: SWOT-Analyse der gewählten Lösungsvariante (Übersicht)

7.3.1 Interne Faktoren

Stärken

- **Vollautomatische Erkennung:** Die Differenzerkennung für Parameter (Datenbankhistorie) und aktive Steuerungselemente (XML-Vergleich) läuft ohne manuelle Anteile. Damit erfüllt die Variante die Muss-Kriterien K1 und K2 vollständig, der grösste Hebel gegenüber dem Ist-Prozess.
- **Erprobte Technologie:** C#/.NET Framework ist im Ammann-Umfeld etabliert, die Entwicklung erfolgt auf derselben Plattform, auf der die **as1** läuft. Keine neuen Abhängigkeiten von Drittanbietern.
- **Schichtenarchitektur als Basis:** Die Trennung von Bedienung (CLI), Ablaufsteuerung und Datenzugriff (NF-05) macht die Kernlogik unabhängig von der Bedienoberfläche. GUI und weitere Funktionen sind nachrüstbar, ohne den Kern anzufassen.
- **Sicherheitsorientiertes Design:** Nur lesender Datenbankzugriff (NF-02), keine Veränderung von Quelldateien (NF-03). Das Tool kann den Ist-Zustand per Konstruktion nicht beschädigen, ein zentraler Vertrauensfaktor bei kundensystemnahem Einsatz.
- **CLI vor GUI:** Der Kommandozeilenbetrieb ist bereits der Realisierungszustand der Arbeit. Damit ist die Bedienung durch den Engineer in der Entwicklungsumgebung als auch automatisiert (Skripte, **as1**-Integration) möglich.

Schwächen

- **Keine grafische Oberfläche:** Die geführte Bedienung (Baustein 5 der Variante) ist konzipiert, wird aber in dieser Arbeit nicht realisiert. Die Bedienung erfordert Kommandozeilen-Kenntnisse und ist weniger intuitiv als der ursprüngliche Varianten-Umfang.
- **Abhängigkeit vom Snapshot-Disziplin:** Die Erkennungsqualität steht und fällt mit der Erstellung des Snapshots **vor** Beginn der Anpassungen. Wird der Snapshot vergessen oder zu spät erstellt, sind die früheren Änderungen nicht im Paket enthalten. Das Tool erkennt diesen Fall nicht automatisch.
- **Beschränkung auf die lokale Datenbankhistorie:** Die Änderungshistorie der MSSQL-Datenbank speichert maximal drei Einträge pro Parameter (vgl. Ist-Analyse 6.1.1). Bei häufig geänderten Parametern kann die Historie die tatsächlichen Änderungen nicht vollständig auflösen.
- **Kein Abgleich mit dem Parameterexport:** Die Verifikation der erkannten Parameteränderungen gegen den manuellen Export der **as1** (F-09) ist als KANN-Anforderung nicht realisiert. Eine zweite, unabhängige Quelle fehlt damit als Gegenprobe.
- **Einplatz-Charakter der Entwicklung:** Das Tool wird innerhalb von zwei Wochen als Einzelprojekt entwickelt. Code-Reviews durch Dritte, Pair-Programming oder vergleichbare Qualitätssicherungs-Massnahmen finden nicht statt.

7.3.2 Externe Faktoren

Chancen

- **Standardisierungspotenzial bei Ammann:** Das Tool erzwingt über die install.config-Generierung und die ReadMe-Erstellung einen einheitlichen Paketstandard. Wird es übernommen, löst es direkt die in der Ist-Analyse identifizierte Schwachstelle "kein definierter Standard" ab.
- **Anschlussfähigkeit an as1:** Die CLI-Schnittstelle ist so gestaltet, dass die spätere **as1**-Integration (Startfunktion, vgl. Baustein 1 der Variante) nachrüstbar ist, ohne dass Ammann die Kernlogik anpassen muss. Das senkt die Hürde für eine spätere Produktübernahme.
- **Nachrüstbare GUI als Entwicklungspfad:** Die Schichtenarchitektur erlaubt Ammann, die geführte Bedienung nachzurüsten, ohne dass die Diplomarbeit darauf warten muss. Damit ist die Arbeit ein konsistentes Zwischenprodukt mit klarem Weg zum Endausbau.
- **Übertragbarkeit auf weitere as1-Varianten:** Die Konzeption (Snapshot-Vergleich, Historie-Auswertung) ist nicht auf V6.0/V6.1 beschränkt. Ähnliche Bedingungen bei anderen Versionen könnten mit vertretbarem Aufwand erschlossen werden.

Risiken (extern)

(Die folgenden Risiken werden in der nachfolgenden Risiko-Analyse (Soll) mit Eintrittswahrscheinlichkeit und Schadensausmass bewertet und mit Gegenmassnahmen hinterlegt.)

- Snapshot-Versäumnis bei der Nutzung im Alltag (Disziplinrisiko, vgl. Schwäche)
- Datenbankhistorie unvollständig oder geleert (Datenrisiko)
- Testsystem nicht wie geplant verfügbar (Umgebungsrisiko, vgl. R3 der Projekt-Risikoanalyse)
- **as1HotfixInstaller** verhält sich in der Testumgebung anders als dokumentiert (Integrationsrisiko)

7.4 Risiko-Analyse (Soll)

(Planung (Soll): Die Analyse bewertet die Risiken der geplanten Realisierung. Der Stand der Umsetzung der Gegenmassnahmen ist in 7.4.3 beschrieben.)

7.4.1 Vorgehen

Die Risiko-Analyse (Soll) untersucht die Risiken der **gewählten Lösung** im Betrieb, ergänzend zur Projekt-Risikoanalyse, die die Risiken des Vorhabens selbst betrachtet. Die Bewertung folgt derselben Methode wie in der Projekt-Risikoanalyse: Ursache-Wirkungs-Kette, dreistufige Skala für Eintrittswahrscheinlichkeit und Schadensausmass, Einstufung über die Bewertungsmatrix, Gegenmassnahmen für mittlere und hohe Risiken.

7.4.2 Risikotabelle

Tabelle 27: Risikotabelle der Risiko-Analyse (Detail/Umsetzung)

Nr.	Risiko (Ursache → Wirkung)	Wahrscheinlichkeit	Schadensausmass	Einstufung	Gegenmassnahme
S1	Der Engineer erstellt den Snapshot versehentlich erst nach den ersten Anpassungen -> Frühere Änderungen fehlen im Updatepaket, die Anlage verhält sich beim Kunden fehlerhaft	mittel	hoch	**hoch**	ReadMe-Hinweis und Schulung: Snapshot ist Schritt 1 im Ablauf. Das Tool protokolliert das Snapshot-Datum im ReadMe, sodass die Lücke bei der Installation erkennbar wird. Optional (Zeitrest): Warnung beim Diff, wenn zwischen Snapshot-Datum und heutigem Datum eine ungewöhnlich lange Zeitspanne liegt
S2	Die Änderungshistorie der Datenbank ist unvollständig (maximal drei Einträge pro Parameter, ältere Einträge überlagert) -> Das Tool erkennt nicht alle Parameteränderungen, das Paket ist unvollständig	mittel	mittel	**mittel**	Das Tool listet im ReadMe die Parameter mit mehreren Historieneinträgen separat auf ("mehrfach geändert, manuell prüfen"). Der Engineer wird damit auf unsichere Fälle aufmerksam, statt sie unbemerkt zu übergehen
S3	Das Testsystem für den HotfixInstaller-Test steht nicht wie geplant zur Verfügung -> Das Muss-Erfolgskriterium "Installation ohne Nachbearbeitung" kann nicht am realen System verifiziert werden	gering	hoch	**mittel**	Testumgebung früh abklären und reservieren (vgl. R3 der Projekt-Risikoanalyse). Alternative: Installationstest auf lokaler Entwicklungsinstanz der **as1**, dokumentierter Verweis auf die Einschränkung
S4	Der **as1HotfixInstaller** verhält sich anders als in der Doku beschrieben (z.B. abweichende Behandlung von CopyFile oder ShowHtml) -> Das generierte Paket wird abgelehnt oder installiert fehlerhaft	gering	hoch	**mittel**	Früher Test mit einem minimalen Testpaket bereits in Phase 3 (nicht erst am Ende), damit Abweichungen früh erkannt werden. Rückgrat ist die dokumentierte install.config-Schema der Ist-Analyse. Bei Abweichungen: Schema-Anpassung mit Ammann-Rückfrage
S5	Datenbankberechtigungen für den lesenden Historienzugriff fehlen auf Kundensystemen -> Die Parameter-Erkennung (F-07) schlägt fehl, das Paket wäre unvollständig	gering	mittel	**gering**	Berechtigungen früh klären (vgl. R6 der Projekt-Risikoanalyse). Tool bricht bei fehlender Berechtigung mit klarer Meldung ab, statt ein unvollständiges Paket zu erzeugen (NF-04)

Nr.	Risiko (Ursache → Wirkung)	Wahrscheinlichkeit	Schadensausmass	Einstufung	Gegenmassnahme
S6	Das Tool wird im Alltag nicht angenommen, der manuelle Prozess bleibt parallel bestehen -> Der Nutzen der Arbeit bleibt aus, es existieren zwei parallele Prozesse	mittel	mittel	**mittel**	Konsistente, niederschwellige Bedienung über wenige CLI-Befehle. ReadMe-Qualität als sichtbares Alleinstellungsmerkmal gegenüber dem manuellen Prozess. Demonstration an realem Projekt im Rahmen der Arbeit

7.4.3 Interpretation

Die Analyse weist ein Risiko mit hoher Einstufung aus (S1: Snapshot-Versäumnis). Es ist das strukturierende Risiko der gewählten Variante, da die Erkennungsqualität direkt von der Einhaltung des Ablaufs abhängt. Die Gegenmassnahme kombiniert Prozess (Schulung, dokumentierter Ablauf) und Tool-Unterstützung (Protokollierung des Snapshot-Datums im ReadMe), vollständig automatisiert lässt es sich nicht eliminieren, da das Tool den Zustand vor seinem eigenen Einsatz nicht kennen kann.

Die vier Risiken mit mittlerer Einstufung (S2, S3, S4, S6) verteilen sich auf Datenqualität (S2), Umgebung (S3, S4) und Akzeptanz (S6). Ihnen gemeinsam ist, dass sie durch frühzeitige Abstimmung (S3, S4), Transparenz im ReadMe (S2) und niederschwellige Bedienung (S6) behandelt werden. Das gering eingestufte Risiko S5 ist durch die Gegenmassnahmen der Projekt-Risikoanalyse (R6) bereits vorgesteuert.

Im Ergebnis zeigt die Risiko-Analyse (Soll): Die Lösung ersetzt das grösste Risikofeld des Ist-Prozesses, die manuelle, personengebundene und fehleranfällige Paketerstellung, durch einen automatisierten, nachvollziehbaren Prozess. Die verbleibenden Risiken betreffen vor allem die Einführung und den Betrieb, nicht die Technik selbst, und sind durch die definierten Gegenmassnahmen beherrschbar.

Stand der Umsetzung: Die Gegenmassnahmen zu S1 und S2 sind in der realisierten Lösung vollständig umgesetzt: Das Snapshot-Datum wird im ReadMe protokolliert (S1), und mehrfach geänderte Parameter werden als unsichere Fälle erkannt, vom automatischen Import ausgeschlossen und im ReadMe in einem eigenen Warnblock mit konkreter Handanweisung aufgeführt (S2). Beide Umsetzungen sind durch Unit-Tests gesichert (vgl. Abschnitt 7.6.5).

7.5 Kosten-Nutzen-Analyse

(Planung (Soll) mit Ist-Werten: Die Annahmen entstanden vor der Realisierung, der Entwicklungsaufwand wurde nach Abschluss durch den tatsächlich angefallenen Aufwand ersetzt.)

7.5.1 Vorgehen

Die Kosten-Nutzen-Analyse stellt die Kosten der Lösung (Entwicklungsaufwand) dem Nutzen (eingesparter manueller Aufwand, vermiedene Fehlerkosten) gegenüber. Da das Tool zum Zeitpunkt der Analyse noch nicht im Einsatz ist, basiert die Rechnung auf Schätzungen und Annahmen. Jede Annahme ist mit Herkunft und Bandbreite ausgewiesen. Die Rechnung wird in drei Szenarien durchgeführt (pessimistisch, realistisch, optimistisch), damit die Aussagekraft der Annahmen sichtbar wird. Der Entwicklungsaufwand wurde nach Abschluss der Realisierung durch den tatsächlich angefallenen Aufwand ersetzt (vgl. Abschnitt 7.2.5).

Quantifizierbarer Nutzen (in CHF berechenbar) und nicht quantifizierbarer Nutzen (qualitativ, nicht bezifferbar) werden getrennt ausgewiesen. Nur der quantifizierbare Nutzen geht in die Amortisationsrechnung ein.

7.5.2 Annahmen

Kosten

Tabelle 28: Kostenannahmen der Kosten-Nutzen-Analyse

Parameter	Wert	Herkunft / Begründung
Entwicklungsaufwand Tool	ca. 70 h	Tatsächlich angefallener Aufwand (vgl. Abschnitt 7.2.5). Ursprünglich geplant waren 10 Arbeitstage à 8 h
Stundenansatz Engineer (interner Verrechnungssatz)	CHF 125/h	Interner Verrechnungssatz der Ammann Schweiz AG für Software Engineers (CHF 1'000 pro 8-Stunden-Tag)
Entwicklungskosten gesamt	CHF 8'750	70 h × CHF 125

Nutzen (Ist-Aufwand pro Updatepaket)

Tabelle 29: Nutzenannahmen der Kosten-Nutzen-Analyse

Parameter	Wert	Herkunft / Begründung
Manueller Aufwand pro Paket (Ist)	4-8 h	Annahme aus Ist-Analyse 6.1.3 (Differenzdateien mühsam erstellen, ReadMe schreiben, Paket manuell schnüren). Bandbreite mit Engineers verifizieren
Aufwand pro Paket mit Tool (Soll)	0.5-1 h	Restaufwand: Snapshot erstellen, Diff-Ergebnis prüfen, ReadMe finalisieren, Paket auslösen
Eingesparter Aufwand pro Paket	3-7.5 h	Differenz Ist minus Soll
Anzahl kundenspezifischer Pakete pro Jahr	30-50	Expertenschätzung: 30-50 Projekte pro Engineer und Jahr. Wert wurde mit der Fachexpertin von Ammann abgestützt und für die Berechnung als Fixwert angenommen

7.5.3 Szenarien

Tabelle 30: Szenarien der jährlichen Ersparnis

	Pessimistisch	Realistisch	Optimistisch
Eingesparte Zeit pro Paket	3 h	5 h	7.5 h
Pakete pro Jahr	30	40	50
Eingesparte Stunden pro Jahr	90 h	200 h	375 h
Eingesparte Kosten pro Jahr (à CHF 125)	CHF 11'250	CHF 25'000	CHF 46'875

Das folgende Diagramm stellt die jährliche Ersparnis der drei Szenarien den einmaligen Entwicklungskosten von CHF 8'750 gegenüber:

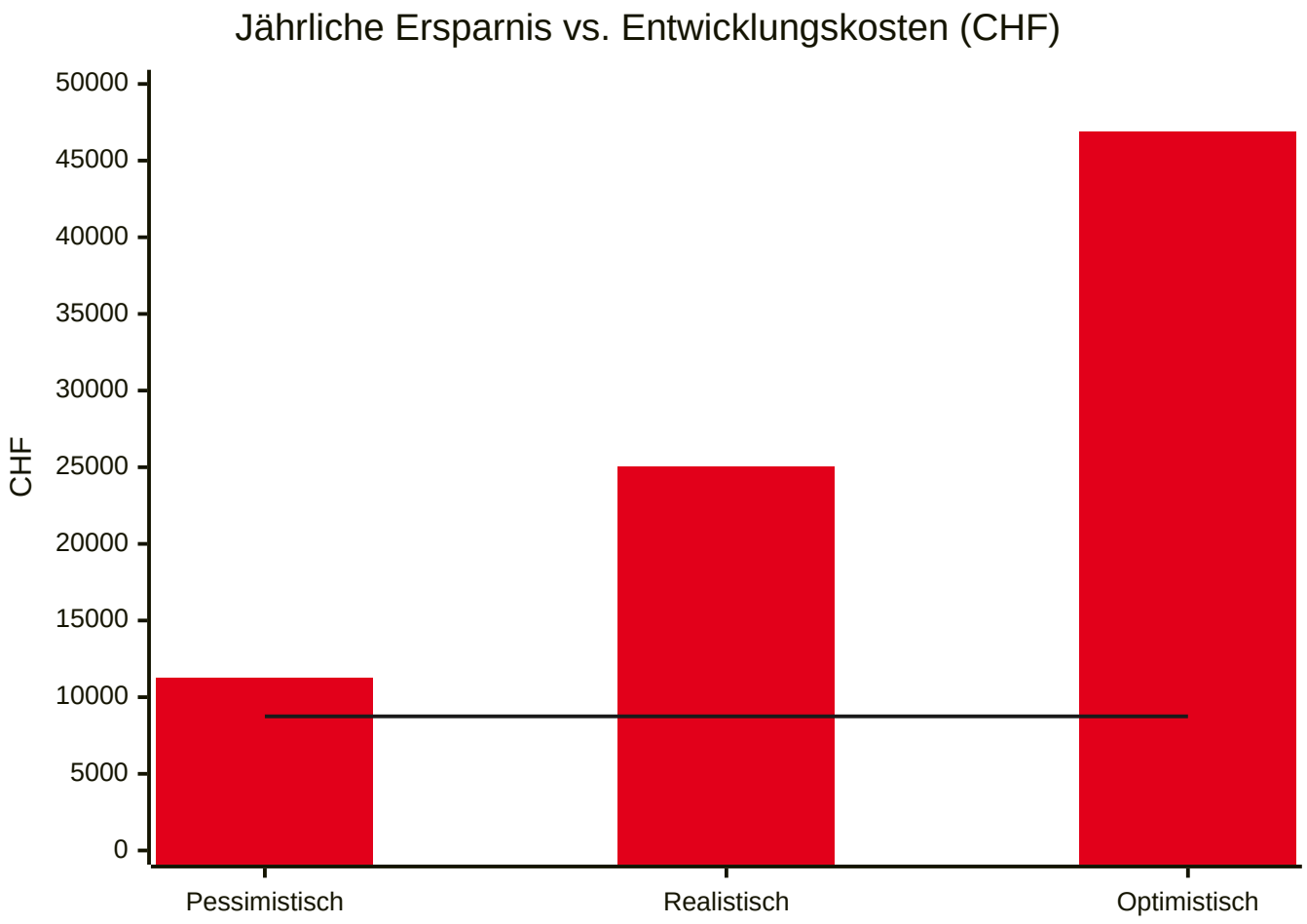

Abbildung 16: Jährliche Ersparnis der Szenarien im Vergleich zu den Entwicklungskosten

7.5.4 Amortisationsrechnung

Tabelle 31: Amortisationsrechnung

	Pessimistisch	**Realistisch**	**Optimistisch**
Entwicklungskosten	CHF 8'750	CHF 8'750	CHF 8'750
Ersparnis pro Jahr	CHF 11'250	CHF 25'000	CHF 46'875
Amortisationsdauer	ca. 9.3 Monate	ca. 4.2 Monate	ca. 2.2 Monate

In allen drei Szenarien amortisiert sich das Tool innerhalb des ersten Betriebsjahres. Selbst im pessimistischen Szenario (niedrigste Zeitersparnis, wenigste Pakete) übersteigt die jährliche Ersparnis die Entwicklungskosten.

7.5.5 Nicht quantifizierbarer Nutzen

Die folgenden Nutzenbeiträge sind real, lassen sich aber nicht seriös in CHF beziffern. Sie werden deshalb qualitativ ausgewiesen und gehen nicht in die Amortisationsrechnung ein:

- **Vermeidung von Sachschaden und Anlagenstillstand:** Ein fehlerhaftes Updatepaket kann auf der Kundenseite einen Anlagenstillstand verursachen (vgl. Ist-Analyse 6.1.3, Fallbeispiel). Die automatisierte

Erkennung reduziert das Risiko übersehener Änderungen deutlich. Ein einziger vermiedener Stillstand übersteigt die Entwicklungskosten um ein Vielfaches, ein monetärer Nachweis ist im Voraus jedoch nicht möglich.

- **Aufhebung der Personenabhängigkeit:** Der Ist-Prozess ist an einzelne Engineers gebunden (personengebundenes Wissen, vgl. Ist-Analyse 6.1.3). Das Tool mit strukturierter ReadMe macht die Paketerstellung breiter durchführbar.
- **Standardisierung:** Das Tool erzwingt einen einheitlichen Paketaufbau (install.config-Schema, ReadMe-Struktur) und behebt damit direkt die identifizierte Ist-Schwäche "kein definierter Standard".
- **Qualität der Dokumentation:** Die automatisch generierte ReadMe ist bei jedem Paket vorhanden, aktuell und nachvollziehbar. Im Ist-Prozess war sie je nach Engineer und Zeitdruck unterschiedlich gut oder fehlend.
- **Nachrüstbarkeit als Investition:** Die Schichtenarchitektur (NF-05) erlaubt Ammann, GUI und Kann-Funktionen nachzurüsten, ohne die Kernlogik anzufassen. Der Entwicklungsaufwand dieser Arbeit bleibt damit auch bei zukünftigen Ausbaustufen nutzbar.

7.5.6 Interpretation

Die Kosten-Nutzen-Analyse zeigt, dass sich das Tool bereits unter vorsichtigen Annahmen innerhalb eines Jahres amortisiert (realistisches Szenario: rund vier Monate). Diese Aussage ist robust: Selbst wenn die Annahmen zum manuellen Aufwand um die Hälfte zu hoch gegriffen sein sollten, bleibt die Amortisationsdauer unter einem Jahr.

Der nicht quantifizierbare Nutzen ist dabei mindestens so gewichtig wie die reine Zeitersparnis. Die Verminderung des Stillstandsrisikos behandelt das grösste Sachrisiko des Ist-Prozesses. Die jährliche Ersparnis von rund CHF 25'000 (realistisches Szenario) entspricht rund 200 Arbeitsstunden, also etwa einem Zehntel einer Jahresvollzeitstelle, und ist für Ammann damit operativ relevant.

7.6 Realisierung (Resultat)

(Ist-Zustand nach abgeschlossener Realisierung. Abweichungen von der Planung sind in Abschnitt 7.6.6 begründet.)

7.6.1 Aufbau und Entwurfsprinzipien

Die Lösung ist als Kommandozeilen-Tool in C#/.NET Framework realisiert [7] und in vier Projekte mit klarer Schichtentrennung gegliedert:

Tabelle 32: Projektstruktur des As1UpdateTools

Projekt	Schicht	Verantwortung
`As1UpdateTool.Cli`	Bedienung	Argument-Verarbeitung, Befehlsdispatch, Ausgaben und Exit-Codes
`As1UpdateTool.Core`	Ablaufsteuerung / Fachlogik	Snapshots, Differenzerkennung, Paketgenerierung
`As1UpdateTool.Data`	Datenzugriff	Lesender Zugriff auf die MSSQL-Datenbank und die **as1**-Systemkonfiguration
`As1UpdateTool.Tests`	Qualitätssicherung	Unit- und Zusammenspiel-Tests (77 Tests)

Drei Entwurfsprinzipien durchziehen die gesamte Codebasis:

- **Interfaces an Systemgrenzen:** Jede Abhängigkeit zu einem System, das im Test nicht verfügbar ist, liegt hinter einem Interface (`IAs1DbAccess` für die Datenbank, `IPackageBuilder` für die Paketerstellung inkl. 7-Zip-Aufruf). Dadurch sind sämtliche Unit-Tests ohne **as1**-Installation, ohne Datenbank und ohne Netzwerk lauffähig.
- **Zentrale Konfigurationsstelle:** Alle Standardpfade sind an genau einer Stelle (`ToolSettings.CreateDefault()`) definiert. Eine spätere Konfigurierbarkeit über eine Konfigurationsdatei erfordert ausschliesslich die Änderung dieser einen Methode.
- **Fachliche Zustände statt Fehlercodes:** Rückgabewerte steuern den Ablauf (Exit-Code 0 = Erfolg, 1 = Aufforderung an den Engineer, z.B. "zuerst Snapshot erstellen"), unbehandelte Umweltprobleme werden zentral abgefangen und mit Exit-Code 2 quittiert.

Das folgende Zustandsdiagramm zeigt die fachlichen Zustände des Paketbaus (`package`) mit den zugehörigen Exit-Codes. Die Zustände entsprechen dem `UpdateResult`-Status des `UpdateOrchestrator`:

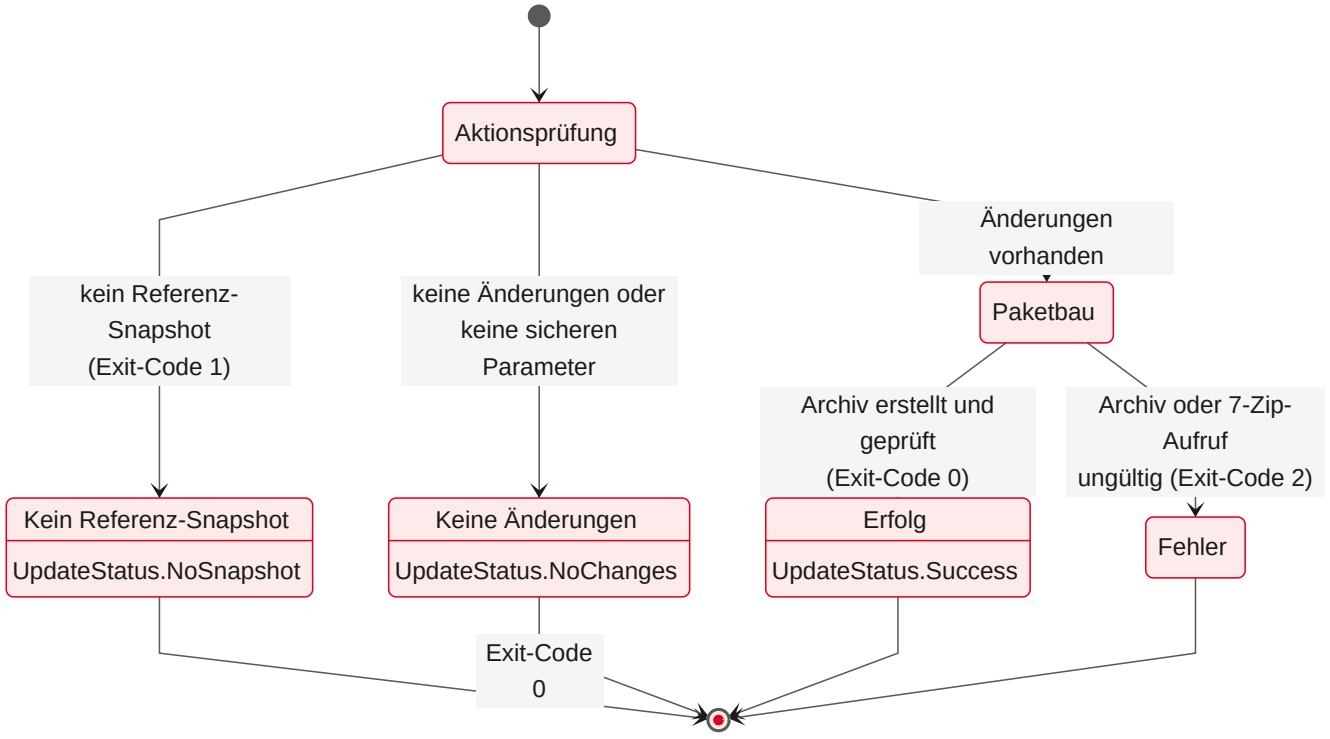

Abbildung 17: Fachliche Zustände des Paketbaus mit Exit-Codes

Die folgende Abbildung zeigt die Klassenstruktur und die wichtigsten Abhängigkeiten (reduzierte Darstellung ohne Attribute und Methoden. Die Schnittstellen sind gestrichelt dargestellt):

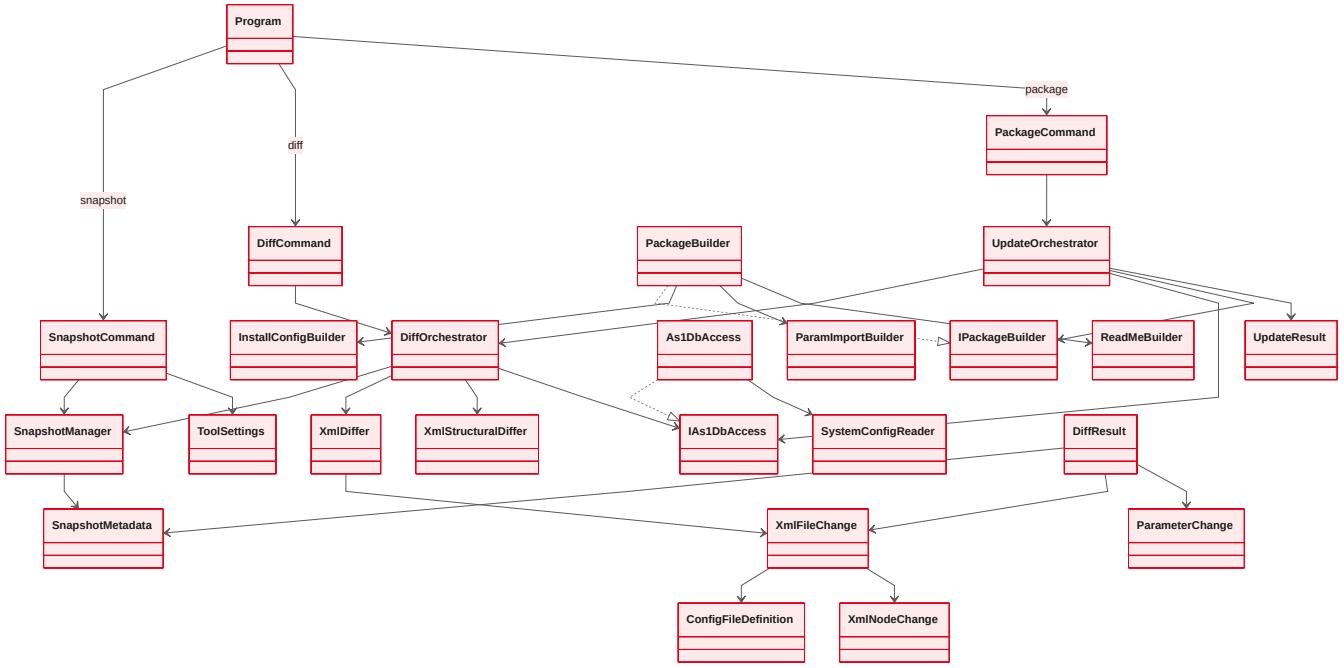

Abbildung 18: Klassenstruktur des As1UpdateTools (reduzierte Darstellung)

7.6.2 Klassen der Bedienschicht (Cli)

`Program` - Einstiegspunkt des Tools. Verarbeitet die Kommandozeilenargumente, leitet sie an die zuständige Befehlsklasse weiter und behandelt Ausnahmen zentral mit einheitlichen Exit-Codes (0 Erfolg, 1 fachliche Aufforderung, 2 Umweltfehler).

`SnapshotCommand` - Bindeglied zwischen CLI und Kern für `snapshot create` und `snapshot list`. Erzeugt die Verdrahtung (ToolSettings, SnapshotManager) und gibt das Resultat auf der Konsole aus. Die Anlagen-Kennung wird seit der Realisierungsentscheidung F-01 nicht mehr als Argument entgegengenommen, sondern aus der Datenbank bezogen, Tippfehler in der Kennung sind damit konstruktiv ausgeschlossen.

`DiffCommand` - Bindeglied für `diff`: Ruft den `DiffOrchestrator` auf und stellt das erkannte DiffResult strukturiert auf der Konsole dar (geänderte Dateien, erkannte Struktur-Operationen, Parameteränderungen mit Kennzeichnung unsicherer Fälle).

`PackageCommand` - Bindeglied für `package`: Verdrahtet die vollständige Kette (Diff, DB, PackageBuilder) und führt die Pflichtoption `--sign` (Kürzel des Erstellers, analog `-s` des **as1HotfixInstaller**) und optional `--out` (Zielordner, Standard `C:\Public\Hotfixes` - der Ordner, aus dem der HotfixInstaller lädt). Fachliche Zustände wie "keine Änderungen" werden mit erklärenden Meldungen quittiert.

7.6.3 Klassen des Kerns (Core)

Modelle (`Core.Model`). Die Datenklassen enthalten keine eigene Logik und tragen die Fachdaten durch die gesamte Kette: `SnapshotMetadata` (Metadaten eines Snapshots. Der Ordnerpfad ist als `[JsonIgnore]` nicht persistiert, sondern wird beim Laden aus dem Ordner ergänzt, damit der Pfad nur an einer Stelle gepflegt wird), `ConfigFileDefinition` (statisches Register der beobachteten XML-Konfigurationsdateien), `ParameterChange` (eine erkannte Parameteränderung inkl. Pfad, Werten, Autor und dem S2-Kennzeichen `IsUncertain`), `XmlFileChange` (eine erkannte Dateiänderung samt Struktur-Operationen), `XmlNodeChange`

(eine einzelne Add/Replace/Remove-Operation mit XPath-Anker), `DiffResult` (das Gesamtergebnis eines Vergleichs), `UpdateResult` (Status des Paketbaus) und `ToolSettings` (Pfadkonfiguration).

`SnapshotManager` (`Core.Snapshots`) - Verwaltet das Erzeugen, Auflisten und Laden von Snapshots. Ein Snapshot ist ein Ordner, der die drei beobachteten XML-Konfigurationsdateien als byte-identische Kopien sowie eine `snapshot.json` mit den Metadaten enthält. Snapshots werden pro Anlage gruppiert (Ordnername = Anlagen-Kennung aus der Datenbank, F-01). Die Klasse kennt weder Datenbank noch CLI, sie wurde während der Umstellung der Anlagen-Erkennung bewusst nicht verändert, was die Schichtentrennung (NF-05) belegt.

`HashUtils` (`Core.Snapshots`) - Hilfsklasse zur SHA-256-Hashberechnung für die Vorfilterung des Dateivergleichs (F-04).

`XmlDiffer` (`Core.Diff`) - Vergleicht die drei beobachteten XML-Konfigurationsdateien anhand ihrer Hashwerte zwischen Referenz-Snapshot und Ist-Zustand und liefert pro Datei den Änderungstyp (Unchanged/Added/Modified/Deleted) sowie bei Bedarf den aktuellen Inhalt als CopyFile-Payload (F-05).

`XmlStructuralDiffer` (`Core.Diff`) - Leitet für jede geänderte Datei die kleinstmöglichen Struktur-Operationen ab: Für jedes hinzugefügte, ersetzte oder entfernte XML-Element werden der XPath-Anker, die Position (ober-/unterhalb des Ankers) und das Element-Fragment als `XmlNodeChange` bestimmt (F-06). Sind die Operationen vollständig, kann die Installation über `ModifyXml` erfolgen, anstatt die gesamte Datei zu überschreiben. Schlägt die Ableitung fehl, greift der `CopyFile`-Fallback.

`DiffOrchestrator` (`Core.Diff`) - Führt die Erkennung zusammen: XML-Vergleich, Struktur-Analyse und Datenbank-Historie ergeben gemeinsam das `DiffResult`. Die Datenbank-Abfrage beginnt ausdrücklich beim Erstellungsdatum des Referenz-Snapshots (F-07).

`InstallConfigBuilder` (`Core.Import`) - Erzeugt die `install.config` gemäss dem Schema des **as1HotfixInstaller** (F-10). Kernentscheidung ist die Schritt-Zuordnung pro Datei: `NodeOperations` vorhanden -> `ModifyXml` mit separater Modifikationsdatei, sonst -> `CopyFile`-Fallback. Neue Dateien -> `CopyFile`, gelöschte -> `DeleteFile`. Zusätzlich erzeugt er die Sicherungsstrategie vor der Installation (`CreateBackup` für Parameter, Datenbank und jede ModifyXml-Zieldatei, bei `CopyFile` sichert der Installer selbst), den `ImportParameters`-Schritt (nur wenn mindestens ein sicherer Parameter existiert, dass der Import-Schritt genau dann erzeugt wird, wenn die Importdatei im Paket vorhanden ist, ist durch einen eigenen Test gesichert) und die ReadMe-Einbindung (`CopyFile` nach `Coma\Updates\Html` plus `ShowHtml` mit Rückfrage im `PostInstallation`, F-12).

`ParamImportBuilder` (`Core.Import`) - Erzeugt aus den erkannten Parameteränderungen eine partielle ParamTransfer-Importdatei, die der **as1HotfixInstaller** über den `ImportParameters`-Schritt importiert (siehe Realisierungsentscheidung zu F-08, Abschnitt 7.6.6). Parameterpfade werden exakt aus der Datenbank übernommen. Diese Klasse ist der einzige Ort, an dem die S2-Filterung stattfindet: Unsichere Parameter (`IsUncertain`) werden hier ausgeschlossen und erscheinen stattdessen im ReadMe-Warnblock (Gegenmassnahme zu S2).

`ReadMeBuilder` (`Core.Import`) - Erzeugt die Update-Dokumentation als HTML-Datei: Kopfdaten (Anlage, Referenz-Snapshot, Ersteller mit Kürzel), Änderungstabellen für Dateien und Parameter, ein hervorgehobener Warnblock für manuell zu prüfende (S2-)Parameter mit konkreter Handanweisung sowie die Installationsschritte (F-12, F-19). Alle Anlagendaten werden HTML-escapten ausgegeben. Parameter-Namen und Kommentare sind Fremddaten und dürfen in einer im Browser geöffneten Datei nie als Markup interpretiert werden.

`PackageBuilder` (`Core.Import`, implementiert `IPackageBuilder`) - Baut das physische Updatepaket: In einem temporären Staging-Ordner werden `install.config` (auf oberster Archivebene, wie der Installer es erwartet), die Datei-Payloads, die ParamTransfer-Importdatei und die ReadMe an den durch die Konstanten des `InstallConfigBuilder` festgelegten Pfaden abgelegt. Anschliessend wird der Ordner per Aufruf der installierten `7z.exe` (`a -sfx`) zu einem selbstentpackenden Archiv mit dem Pflichtpräfix `Package-` verpackt (F-13, F-14). Ohne das Präfix behandelte der HotfixInstaller das Archiv als Hotfix und würde die install.config ignorieren. Das Präfix ist deshalb fest verdrahtet. Die Struktur-Erzeugung ist als eigene Methode abgetrennt und unit-getestet. Der externe 7-Zip-Aufruf ist bewusst dem Realversuch vorbehalten.

`UpdateOrchestrator` (`Core.Orchestration`) - Steuert den kompletten Ablauf `diff` -> `package`: Holt das DiffResult, lehnt Paketbau ohne Referenz-Snapshot oder ohne Änderungen ab (leere Pakete werden nicht erstellt) und übergibt Kennung, Anlagen-GUID und Ersteller-Kürzel an den Paketbau. Fachliche Zustände werden als `UpdateResult`-Status an die CLI zurückgegeben statt als Exceptions.

7.6.4 Klassen der Datenschicht (Data)

`As1DbAccess` **(implementiert `IAs1DbAccess`)** - Einziger Ort mit SQL-Zugriff. Liest die Änderungshistorie ab einem Datum über eine rekursive CTE-Abfrage [8] aus (inklusive Auflösung der Parameterpfade) sowie Anlagen-Name und -GUID. Die Verbindung erfolgt ausschliesslich lesend über Windows-Authentifizierung (Integrated Security). Das Tool speichert und verwaltet keinerlei Datenbank-Credentials (NF-02).

`SystemConfigReader` - Liest Server- und Datenbankname aus der `System.config` der **as1** und setzt daraus den Connection-String zusammen. Ein Unit-Test sichert dauerhaft, dass der erzeugte String keine Zugangsdaten enthält.

7.6.5 Qualitätssicherung und Testkonzept

Das Testprojekt umfasst 77 Tests in zehn Test-Fixtures sowie die zwei Grenz-Fakes `FakeAs1DbAccess` und `FakePackageBuilder` (Implementierungen der beiden Schnittstellen mit aufzeichnenden Eigenschaften). Zwei Testgattungen sind bewusst getrennt:

- **Klassen-Einzeltests** prüfen jeden Baustein für sich (z.B. Hash-Berechnung, JSON-Roundtrip, XPath-Anker, install.config-Schritte, HTML-Escaping).
- **Zusammenspiel-Tests** prüfen die Kette über mehrere Bausteine hinweg (z.B. Snapshot -> Änderung -> Erkennung, Paketstruktur als Nachweis der Staging-Logik, Snapshot einer fremden Anlage wird nicht als Referenz verwendet).

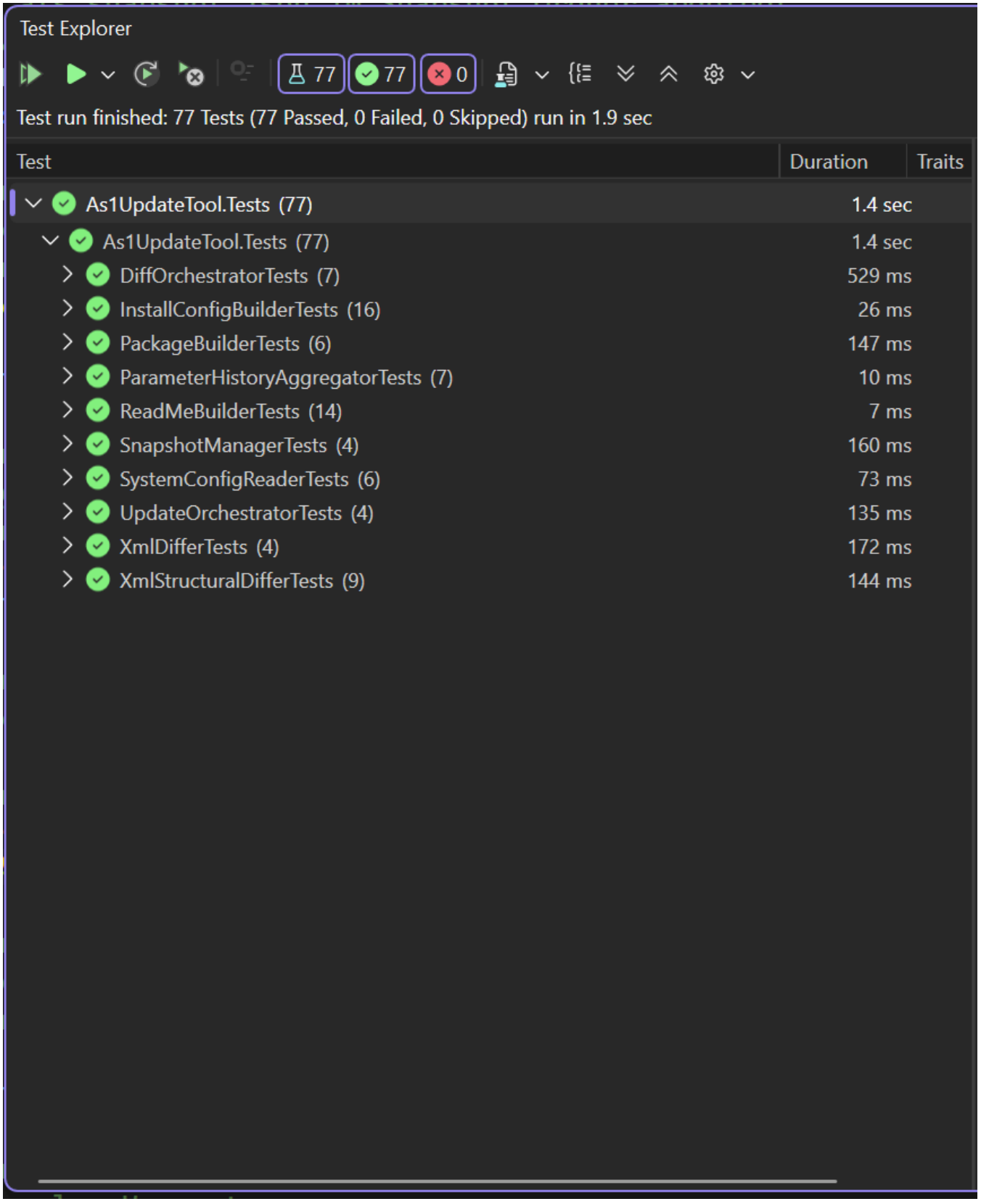

Abbildung 19: Ausgabe der Unit-Tests, sämtliche Tests sind bestanden

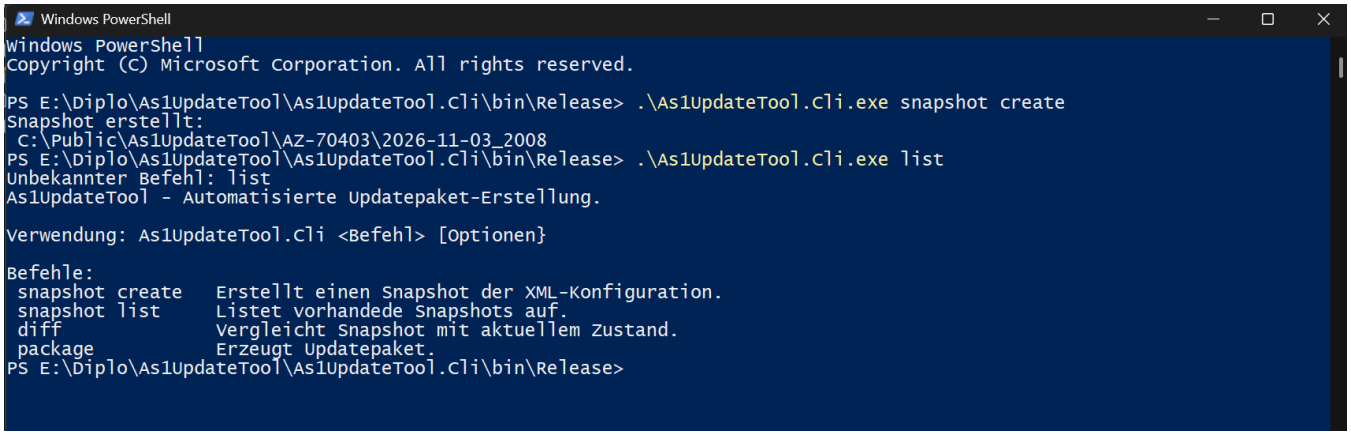

Abbildung 20: Erstellen und Auflisten eines Snapshots über die Kommandozeile

Fremdsystemgrenzen (SQL-Schematypen, 7-Zip-Verhalten, Installer-Erwartungen) sind mit Unit-Tests prinzipbedingt nicht abdeckbar. Sie wurden im Realversuch validiert: Komplettdurchlauf auf einer **as1**-Entwicklungsumgebung, manuelle Prüfung der Archivstruktur (install.config auf oberster Ebene), Installation mit dem **as1HotfixInstaller** und anschliessende Verifikation der installierten Werte. Der Realversuch deckte dabei einen Typ-Mismatch zwischen einer SQL-Spalte (Guid statt String) und dem Lesezugriff auf, der vor dem Realversuch nicht sichtbar war, die Erkenntnis bestätigt die Notwendigkeit der Trennung von Unit-Tests und Realversuch.

Die folgenden Abbildungen zeigen die wesentlichen Schritte und Ergebnisse des Realversuchs:

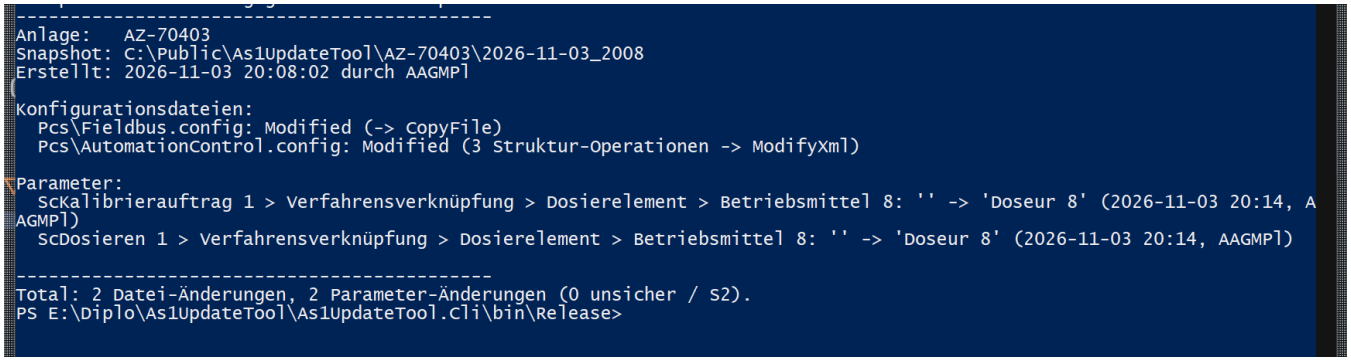

Abbildung 21: Erkannte Änderungen der Ausgabe des Befehls diff

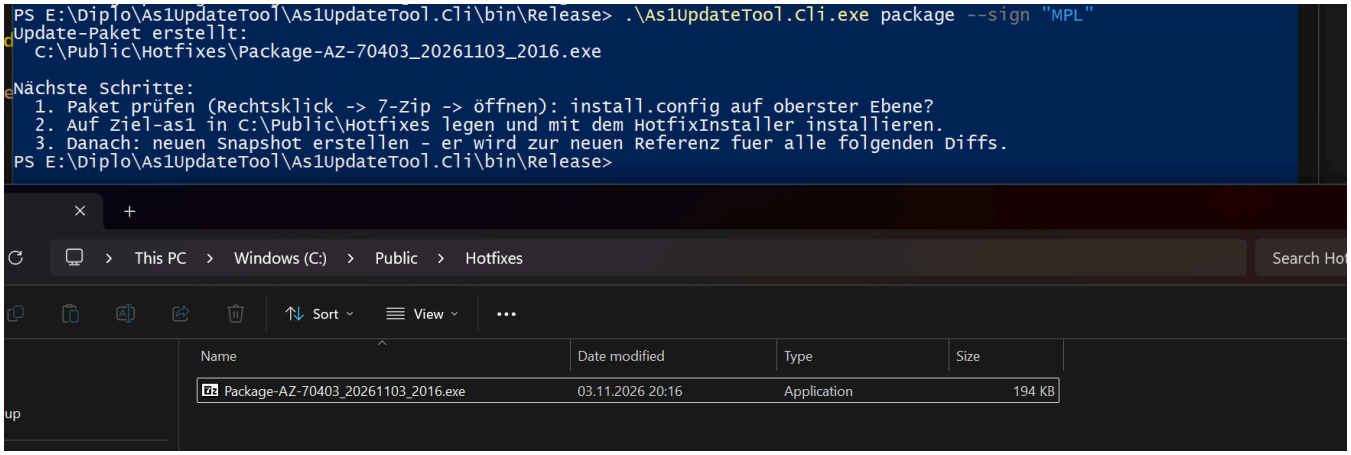

Abbildung 22: Erzeugtes Updatepaket mit dem Pflichtpräfix `Package-`

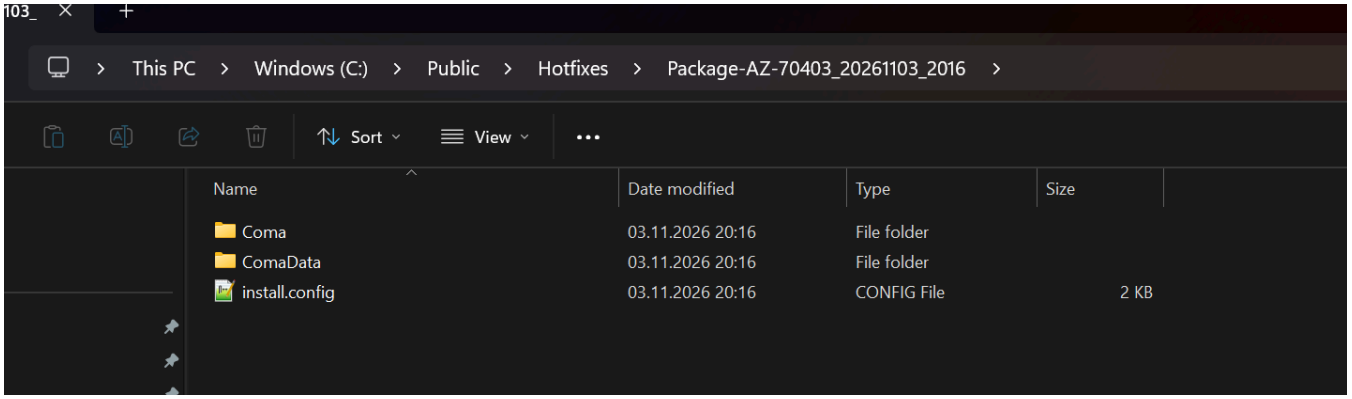

Abbildung 23: Entpackte Paketstruktur mit `install.config` auf oberster Ebene

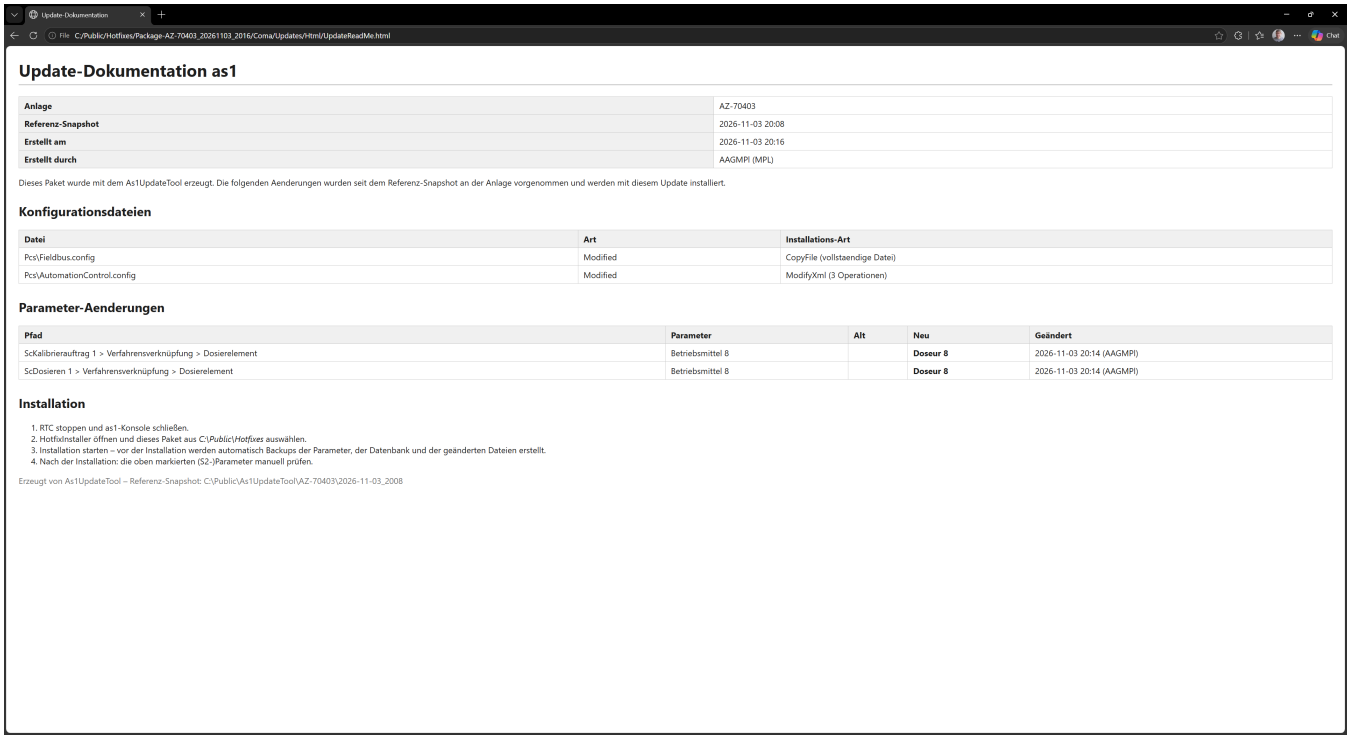

Abbildung 24: Automatisch generierte ReadMe mit Änderungsübersicht und Warnblock

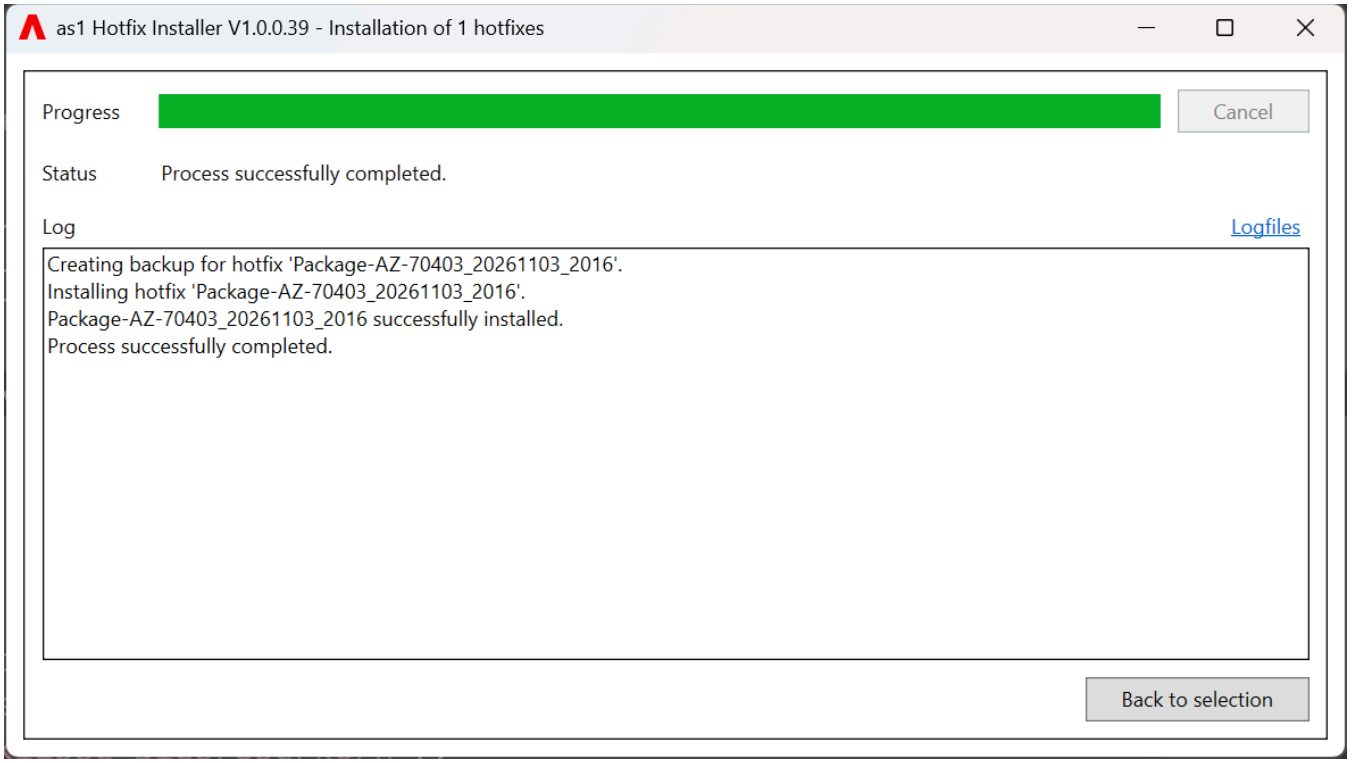

*Abbildung 25: Installation des erzeugten Pakets mit dem **as1HotfixInstaller***

Validierung im realisierten Tool. Die Validierung erfolgt im realisierten Tool über die fachlichen Zustände und Abbruchbedingungen des Ablaufs: Fehlende Voraussetzungen (kein Referenz-Snapshot, keine Datenbankverbindung, keine Änderungen) werden mit konkreten Meldungen quittiert und brechen den Vorgang kontrolliert ab (NF-04). Unsichere Parameteränderungen werden vor der Paketgenerierung ausgeschlossen und im ReadMe-Warnblock zur manuellen Prüfung ausgewiesen (Gegenmassnahme zu S2). Die in Variante 2 konzipierte Schema-Validierung der `install.config` (F-11) und die GUI-Eingabevalidierung (F-17) sind gemäss Umfangsentscheidung (Kapitel 7.1.3) den SOLL- bzw. KANN-Anforderungen zugeordnet und wurden nicht realisiert. Stattdessen wird die erzeugte `install.config` im Realversuch gegen die Erwartungen des **as1HotfixInstaller** geprüft. Die inhaltliche Korrektheit der erkannten Parameteränderungen kann das Tool nicht prüfen. Sie bleibt Aufgabe des Engineers und wird durch die ReadMe-Übersicht unterstützt (vgl. Validierungskonzept in Kapitel 6.4.2).

```
PS E:\Diplo\As1UpdateTool\As1UpdateTool.Cli\bin\Release> .\As1UpdateTool.Cli.exe package --sign "MPL"
Kein Snapshot für diese Anlage vorhanden.
Bitte zuerst 'as1update snapshot create' ausführen.
PS E:\Diplo\As1UpdateTool\As1UpdateTool.Cli\bin\Release>
```

Abbildung 26: Kontrollierter Abbruch bei fehlendem Referenz-Snapshot mit fachlicher Meldung (NF-04, Prüffall 1)

7.6.6 Realisierungsentscheidungen und Abweichungen vom Anforderungsprofil

Im Zuge der Realisierung wurden einzelne Anforderungen gegenüber der Planung angepasst. Die Änderungen sind bewusste Entscheidungen und werden im Folgenden begründet:

- **F-08 ParamTransfer-Import statt SQL-Skript:** Statt eines generierten SQL-Skripts (F-08 wie geplant) erzeugt das Tool eine ParamTransfer-Importdatei, die der **as1HotfixInstaller** über den nativ unterstützten `ImportParameters`-Schritt einspielt. Begründung: (1) Der Import läuft über den **as1**-eigenen, geprüften Mechanismus statt über rohes SQL, das Paket verhält sich wie ein von Ammann vorgesehenes Paket. (2) Es sind keinerlei schreibende Datenbankzugriffe des Tools nötig, was NF-02 verstärkt und das Risiko S5 (Berechtigungen auf Kundensystemen) reduziert. (3) Der `ImportParameters`-Schritt ist Bestandteil des dokumentierten install.config-Schemas (Ist-Analyse 6.1.4).

Das Muss-Ziel 1a (automatische Erkennung und Berücksichtigung von Parameteränderungen im Updatepaket) bleibt vollständig erfüllt. Die Anforderung F-08 wurde bewusst nicht nachträglich im Anforderungsprofil (Kapitel 7.1) umgeschrieben, damit die Planung als abgestimmter Ist-Zustand erhalten bleibt.

- **F-06 und F-14 wurden realisiert:** Beide waren als SOLL ("wenn Zeit bleibt") eingeteilt. Der Projektfortschritt erlaubte die Umsetzung. F-06 (ModifyXml-Operationen) ist vollständig umgesetzt und getestet, F-14 (Paketprüfung) prüft Existenz und Exit-Code des Archivs. Der erweiterte Realisierungsumfang ergibt sich aus dem zügigeren Verlauf von Phase 3 (vgl. Abschnitt 7.2.5).
- **F-16 und F-19 teilweise realisiert:** Die zentrale Konfigurationsstelle (`ToolSettings.CreateDefault()`) umfasst die Pfad-Vorbelegung, eine Anpassung der Pfade ausserhalb des Codes (Konfigurationsdatei, Kommandozeile) ist jedoch nicht realisiert. Die ReadMe enthält strukturierte Änderungstabellen, der Detailgrad bleibt jedoch hinter dem technisch machbaren zurück. Beides ist als Erweiterungspfad in der Architektur vorgesehen (NF-06) und wird in der Evaluation der Zielerreichung als teilweise erfüllt ausgewiesen (Kapitel 8.2).
- **NuGet-Abhängigkeiten (Korrektur zu Kapitel 7.2.3):** Real eingesetzt sind Newtonsoft.Json (Serialisierung) [10] und NUnit (Test-Framework) [9]. Das Argument-Parsing ist bewusst handimplementiert (kein CommandLine-Paket), die 7-Zip-SFX-Erstellung erfolgt per Aufruf der auf dem Zielsystem installierten `7z.exe` (kein SevenZip-SDK). Dies entspricht exakt dem bisherigen manuellen Prozess (7-Zip-Kontextmenü) und vermeidet zusätzliche Abhängigkeiten.

---

8. Projektabschluss

8.1 Projektüberwachung (Soll-Ist Abweichungen)

(Abschluss: Der Vergleich entstand nach Abschluss der Realisierung. Die Ist-Werte sind eine nachträglich rekonstruierte Schätzung des Verfassers, da während der Realisierung keine separate Zeiterfassung je Phase geführt wurde.)

Die folgende Tabelle stellt die geplanten Aufwände aus dem Phasenplan (Kapitel 7.2) den tatsächlich benötigten Stunden gegenüber. Die geplante Gesamtsumme betrug 10 Arbeitstage à 8 h (80 h), der tatsächlich angefallene Aufwand rund 70 h.

Tabelle 33: Soll-Ist-Vergleich der Phasenaufwände

Phase	Soll (h)	Ist (h)	Abweichung	Begründung
P1: Fundament	16	24	+8 h	Das Aufsetzen der Projekte dauerte länger als erwartet. Das Erstellen eines kompletten Projekts aus dem nichts war für den Verfasser wegen fehlender Erfahrung in diesem Bereich neu.
P2: Erkennung	24	22	−2 h	Die XML-Differenzerkennung und die Auswertung der Datenbankhistorie verliefen schneller als gedacht. Die Implementierung der detaillierten Differenzerkennung im XML-Bereich (XPath-Anker, Struktur-Operationen) kostete jedoch so viel Zeit, dass die Phase insgesamt nahezu dem Plan entsprach.
P3: Paket	24	14	−10 h	Die Paketgenerierung (install.config, ReadMe, Paketierung) ging schneller als geplant. Die hier gesparte Zeit ermöglichte die zusätzliche Realisierung der SOLL-Anforderungen F-06 und F-14 innerhalb der Phase (vgl. Abschnitt 7.2.5).
P4: Integration & Test	16	10	−6 h	Dank guter Vorbereitung traten bei den Tests weniger Fehler auf. Bei der Nacharbeit konnte Zeit gespart werden. Der Realversuch bestätigte die Architektur-Annahmen weitgehend.
Total	**80**	**70**	**−10 h**	

Die folgende Abbildung zeigt den tatsächlichen Projektverlauf (Ist) im Ablaufplan. Die Abweichungen der Phasenaufwände gegenüber der Planung sind in Tabelle 33 quantifiziert.

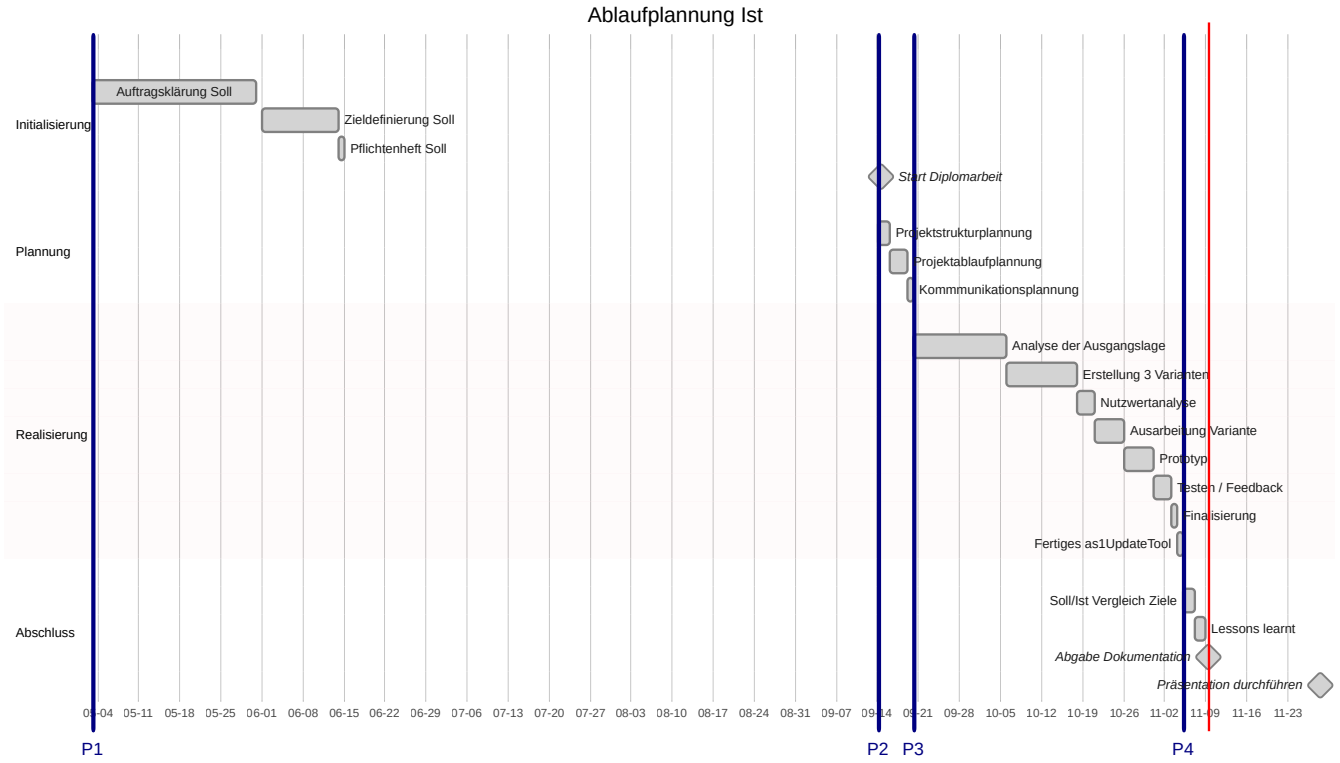

Abbildung 27: Projektablaufplanung (Ist)

Die Überwachung zeigt ein ausgewogenes Gesamtbild: Die Mehrarbeit in Phase 1 (Lerneffekt beim Projektaufbau) wurde durch die schnellere Umsetzung der Phasen 3 und 4 mehr als ausgeglichen. Das Projekt wurde mit rund 70 h und damit deutlich unter dem geplanten Zeitbudget von 80 h abgeschlossen. Der Verlauf bestätigt die Phasenplanung: Der inkrementelle Aufbau mit Meilensteinen erlaubte es, die in Phase 3 gewonnene Zeit direkt in die zusätzlichen SOLL-Anforderungen zu investieren, statt das Budget ungenutzt zu lassen.

Zwei Erkenntnisse aus dem Vergleich fliessen in die Lessons Learned (Kapitel 8.4) ein: Erstens ist bei unbekannten Tätigkeiten, hier dem Aufsetzen der Projektstruktur, ein zusätzlicher Aufwand für Einarbeitung einzuplanen. Zweitens kann gewonnene Zeit durch einen klar priorisierten Katalog an SOLL-Anforderungen (Kapitel 7.1.3) sinnvoll investiert werden, ohne neue Ziele ins Projekt zu bringen.

8.2 Evaluation der Zielerreichung

(Abschluss: Die Bewertung überprüft die Muss- und Kann-Erfolgskriterien des Pflichtenhefts (Kapitel 4.1) gegen den realisierten Stand. Die Erfolgskriterien selbst wurden dafür nicht angepasst.)

Die Bewertung erfolgt je Erfolgskriterium mit dem Erfolgsgrad **erfüllt**, **teilweise erfüllt** oder **nicht erfüllt**. Die Begründung verweist jeweils auf den Nachweis in der vorliegenden Arbeit.

Muss-Erfolgskriterien

Tabelle 34: Evaluation der Muss-Erfolgskriterien

Nr.	Erfolgskriterium	Erfolgsgrad	Begründung / Nachweis
1	Das Tool erkennt Änderungen an Parametern zwischen zwei Softwarezeitpunkten automatisch und berücksichtigt diese im Updatepaket.	erfüllt	Die Änderungshistorie wird ab dem Erstellungsdatum des Referenz-Snapshots über eine lesende CTE-Abfrage ausgewertet (F-07). Die erkannten Änderungen werden als ParamTransfer-Importdatei im Paket berücksichtigt (F-08, vgl. Abschnitt 7.6.6). Die Kette ist durch Unit-Tests und den Realversuch gesichert.
2	Änderungen an aktiven Steuerungselementen werden automatisch erkannt und dokumentiert.	erfüllt	Der Hash-Vergleich der XML-Konfigurationsdateien erkennt geänderte Dateien vollständig automatisch (F-04, F-05). Die ReadMe dokumentiert die Änderungen in tabellarischer Form (F-12).
3	Die erzeugten Updatepakete können mit dem HotfixInstaller ohne manuelle Nachbearbeitung installiert werden.	erfüllt	Der Realversuch hat den vollständigen Durchlauf bestätigt: erzeugtes Paket, Installation mit dem **as1HotfixInstaller** und anschliessende Verifikation der installierten Werte ohne Nachbearbeitung (vgl. Abschnitt 7.6.5).
4	Das automatisch generierte ReadMe dokumentiert die erkannten Änderungen sowie die notwendigen Installationsinformationen nachvollziehbar.	erfüllt	Die HTML-ReadMe enthält Kopfdaten, Änderungstabellen, einen Warnblock für manuell zu prüfende Parameter und die Installationsschritte. Sie wird über den `ShowHtml`-Schritt nach der Installation automatisch angezeigt (F-12, F-19).
5	Die Lösung funktioniert mit den **as1**-Softwareversionen V6.0 und V6.1.	erfüllt	Das Tool wurde mit beiden Softwareversionen getestet.
6	Das Tool kann vollständig über Kommandozeilenparameter ausgeführt werden.	erfüllt	Der vollständige Ablauf (Snapshot erstellen, vergleichen, Paket erstellen) ist über die Befehle `snapshot`, `diff` und `package` ohne Interaktion ausführbar (F-15). Fachliche Zustände werden über Exit-Codes quittiert (vgl. Abbildung 17).
7	Die Lösung wird anhand mindestens eines bestehenden Kundenprojekts erfolgreich getestet und dokumentiert.	erfüllt	Der Realversuch wurde als Komplettdurchlauf auf einer **as1**-Entwicklungsumgebung mit einem bestehenden Kundenprojekt durchgeführt und in Abschnitt 7.6.5 dokumentiert.

Nr.	Erfolgskriterium	Erfolgsgrad	Begründung / Nachweis
8	Die technische Dokumentation beschreibt Aufbau, Funktionsweise und Einschränkungen der Lösung nachvollziehbar.	erfüllt	Die Kapitel 7.6.1 bis 7.6.4 beschreiben Aufbau und Funktionsweise (Projektstruktur, Entwurfsprinzipien, Klassen je Schicht). Die Einschränkungen sind in den Abschnitten 7.6.5 (Testgrenzen, nicht realisierte Validierung) und 7.6.6 (bewusste Abweichungen) ausgewiesen. Ergänzend dokumentiert die generierte ReadMe jede einzelne Installation.

Kann-Erfolgskriterien

Tabelle 35: Evaluation der Kann-Erfolgskriterien

Nr.	Erfolgskriterium	Erfolgsgrad	Begründung / Nachweis
1	Das Tool erkennt Änderungen an den definierten Datenbankobjekten automatisch und berücksichtigt diese im Updatepaket.	nicht erfüllt	Die Anforderung F-18 wurde gemäss Umfangsentscheidung (Kapitel 7.1.3) konzipiert, aber nicht realisiert. Die Auswertung der Änderungshistorie deckt die Parameteränderungen ab, allgemeine Datenbankobjekte (Tabellen, Sichten, Prozeduren) werden nicht ausgewertet. Die Erweiterung ist in der Architektur vorgesehen (NF-06).
2	Die passende Konfigurationsdatei wird automatisch gefunden oder ein geeignetes Standardverzeichnis vorgeschlagen.	teilweise erfüllt	Alle Standardpfade sind zentral in `ToolSettings.CreateDefault()` vordefiniert, der Datenbankzugriff wird aus der `System.config` der **as1** abgeleitet (Abschnitt 7.6.4). Die Pfade können jedoch nur im Code angepasst werden. Eine Konfigurierbarkeit ausserhalb des Codes, zum Beispiel über eine Konfigurationsdatei oder Kommandozeilenparameter, ist nicht realisiert.
3	Das ReadMe enthält eine strukturierte Übersicht sämtlicher erkannten Änderungen.	teilweise erfüllt	Die ReadMe enthält strukturierte Änderungstabellen für Dateien und Parameter (F-19, vgl. `ReadMeBuilder` in Abschnitt 7.6.3). Der Detailgrad der Dokumentation bleibt jedoch hinter dem technisch machbaren zurück. Der weitere Ausbau war im Rahmen der Diplomarbeit zeitlich nicht mehr möglich.

Gesamtbeurteilung: Sämtliche acht Muss-Erfolgskriterien sind erfüllt. Die Arbeit erreicht damit ihr Kernziel vollständig. Von den drei Kann-Erfolgskriterien ist keines vollständig erfüllt: Die Auswertung allgemeiner Datenbankobjekte wurde gemäss Umfangsentscheidung bewusst zurückgestellt, die Pfad-Konfigurierbarkeit und der Detailgrad der Änderungsübersicht sind teilweise umgesetzt und als Erweiterungspfad dokumentiert.

Die Kann-Ziele waren von Beginn an nicht erfolgskritisch. Ihre teilweise Erfüllung schmälert das Ergebnis der Arbeit nicht, bestätigt aber die Priorisierung der Muss-Ziele im Projektverlauf.

8.3 Reflexion Weg zum Ziel

(Abschluss: Die Reflexion ist eine persönliche Rückschau des Verfassers auf den Projektverlauf.)

Zu Beginn der Arbeit stand das Aufsetzen des kompletten Projekts aus dem Nichts. Diese Aufgabe hat sich in der Planung als unterschätzt erwiesen: Für die Struktur der Solution, die Schichten und das Zusammenspiel der Projekte fehlte dem Verfasser die Erfahrung aus vergleichbaren Vorarbeiten. Phase 1 dauerte dadurch acht Stunden länger als geplant. Im Rückblick war dieser Mehraufwand eine Investition, denn die saubere Projektstruktur hat sich in allen folgenden Phasen bezahlt gemacht. Änderungen an der Architektur waren nachträglich nicht mehr nötig.

Der Verlauf der Realisierung hat die Planung insgesamt bestätigt. Der inkrementelle Aufbau mit Meilensteinen hat sich bewährt: Als Phase 3 schneller als geplant verlief, konnte die gewonnene Zeit direkt in die beiden SOLL-Anforderungen F-06 und F-14 investiert werden. Das Projekt hatte zu jedem Zeitpunkt einen funktionsfähigen Zwischenstand. Die grösste Unsicherheit lag nicht in der eigenen Arbeit, sondern an den Grenzen zu den Fremdsystemen. Der Typ-Mismatch zwischen einer SQL-Spalte und dem Lesezugriff, den erst der Realversuch aufgedeckt hat, war mit Unit-Tests prinzipbedingt nicht sichtbar. Diese Erfahrung hat das Vertrauen in die Zweiteilung des Testkonzepts bestätigt: Unit-Tests für die eigene Logik, Realversuch für die Systemgrenzen.

Zwei Punkte würde der Verfasser bei einem ähnlichen Projekt anders angehen. Erstens würde er ab Projektbeginn eine einfache Zeiterfassung je Phase führen. Die Ist-Werte in Kapitel 8.1 sind nachträglich rekonstruiert und damit weniger belastbar, als es eine Aufzeichnung während der Realisierung wäre. Zweitens würde er den ersten Realversuch mit dem **as1HotfixInstaller** früher ansetzen, etwa bereits nach dem ersten generierten Paket in Phase 3, statt ihn auf Phase 4 zu legen. Die Gegenmassnahme zu S4 ("früher Test mit einem minimalen Testpaket") war zwar geplant, hätte aber noch konsequenter umgesetzt werden können.

Ein weiterer Verbesserungspunkt betrifft die Projektkommunikation. Zu den Besprechungen mit den Stakeholdern wurden keine Sitzungsprotokolle erstellt. Die besprochenen Absprachen und Entscheide sind damit nicht schriftlich festgehalten und waren nur aus der Erinnerung und aus der Dokumentation rekonstruierbar. Bei der nächsten Projektarbeit wird der Verfasser zu jeder Besprechung ein kurzes Protokoll mit den getroffenen Entscheiden und offenen Punkten erstellen und den Beteiligten zur Bestätigung zustellen. Dies kostet wenig Zeit, erhöht aber die Nachvollziehbarkeit der Entscheide und schafft eine verbindliche Grundlage für die spätere Bewertung.

Die Arbeit hat der Verfasser sowohl fachlich als auch methodisch weitergebracht. Fachlich stand das vollständige Aufsetzen einer .NET-Anwendung mit Schichtentrennung, Schnittstellen an den Systemgrenzen und einem durchgehenden Testkonzept im Zentrum. Methodisch hat sich gezeigt, dass die disziplinierte Trennung von Planung und Realisierung ein Projekt auch unter Zeitdruck auf Kurs hält: Anforderungen werden vor der Umsetzung fixiert, Abweichungen werden bewusst entschieden und dokumentiert.

8.4 Lessons Learned

(Abschluss: Die Erkenntnisse sind nach Methodik, Technik und Projektmanagement gegliedert.)

Methodik

- **Nutzwertanalyse mit Sensitivitätsprüfung macht Entscheidungen belastbar:** Die bewusst einseitigen Szenarien der Sensitivitätsanalyse (Kapitel 6.5.3) haben gezeigt, dass die Entscheidung für Variante 2 nicht von einer einzelnen Gewichtung abhängt. Eine Nutzwertanalyse ohne solche Prüfung wäre ein Scheinargument geblieben.
- **Der inkrementelle Aufbau ist der beste Puffer:** Ein expliziter Puffertag im Plan wäre ungenutzt verfallen oder unnötig verbraucht worden. Der Meilenstein M3 als funktionsfähiger Zwischenstand hat echten Handlungsspielraum geschaffen, ohne Zeit zu verschwenden.
- **Planung nicht rückwirkend anpassen:** Die bewussten Abweichungen (F-08, F-16, F-19) wurden in 7.6.6 begründet, statt die Planung stillschweigend umzuschreiben. Damit bleibt nachvollziehbar, was geplant war und was daraus wurde.

Technik

- **Unit-Tests erreichen die Fremdsystemgrenzen nicht:** SQL-Schematypen, das Verhalten von 7-Zip und die Erwartungen des **as1HotfixInstaller** sind mit Unit-Tests nicht abdeckbar. Der Typ-Mismatch aus dem Realversuch hat dies eindrücklich bestätigt. Der Realversuch ist keine Option, sondern ein Pflichtbestandteil.
- **Native Mechanismen rohen Techniken vorziehen:** Die Entscheidung für die ParamTransfer-Importdatei statt eines generierten SQL-Skripts (F-08) hat das Paket robuster gemacht und schreibende Datenbankzugriffe vollständig vermieden. Eine Lösung entlang der von Ammann vorgesehenen Wege verhält sich im Betrieb vorhersagbarer.
- **Weniger Abhängigkeiten senken den Wartungsaufwand:** Das handimplementierte Argument-Parsing und der Aufruf der installierten `7z.exe` haben sich bewährt. Jede eingesparte NuGet-Abhängigkeit ist eine weniger zu pflegende Stelle.

Projektmanagement

- **Einarbeitungsaufwand explizit einplanen:** Unvertraute Tätigkeiten wie hier das Aufsetzen der Projektstruktur brauchen zuverlässig mehr Zeit als geschätzt. In Phase 1 hat dieser Aufwand gefehlt, in den Folgephasen hat die gewonnene Routine Zeit eingebracht.
- **Gewonnene Zeit braucht einen vordefinierten Verbraucher:** Der priorisierte SOLL-Katalog (Kapitel 7.1.3) hat es erlaubt, die in Phase 3 gewonnene Zeit ohne neue Diskussion in F-06 und F-14 zu investieren. Ohne diesen Katalog wäre die Zeit verstrichen.
- **Fachwissen unmittelbar verschriftlichen:** Die Rückfragen an die internen Ansprechpartner (Risiko R2) wurden direkt in die Ist-Analyse übernommen. Das Wissen ist damit nicht personengebunden geblieben und hat die spätere Dokumentation erheblich erleichtert.

8.5 Ausblick

(Abschluss: Der Ausblick beschreibt die Weiterentwicklung der Lösung über den Rahmen der Diplomarbeit hinaus. Kurzfristig = innerhalb des bestehenden Tools umsetzbar, mittelfristig = mit Erweiterung der Umgebung.)

Das As1UpdateTool deckt den Kernprozess der Paketerstellung vollständig ab. Der Ausblick gliedert sich nach dem Aufwand der Weiterentwicklung.

Kurzfristig umsetzbar (im bestehenden Tool):

- **Pfad-Konfigurierbarkeit ausserhalb des Codes:** Die zentralen Standardpfade (`ToolSettings`) sind nur im Code anpassbar. Eine Konfigurationsdatei oder ergänzende Kommandozeilenparameter würden die Nutzung auf abweichenden Umgebungen ohne Neukompilierung erlauben (F-16).
- **Erhöhter Detailgrad der ReadMe:** Die Änderungstabellen können um weitere Informationen aus den erkannten Daten ergänzt werden, zum Beispiel vollständige Historienverläufe pro Parameter (F-19).
- **Schema-Validierung der install.config:** Die erzeugte `install.config` wird derzeit im Realversuch geprüft. Eine Validierung gegen das dokumentierte Schema vor der Paketierung würde Fehler noch früher abfangen (F-11).

Mittelfristig umsetzbar (mit Erweiterung der Umgebung):

- **Grafische Bedienoberfläche:** Die Schichtenarchitektur erlaubt das Nachrüsten einer GUI, ohne die Kernlogik anzupassen (NF-05). Eine geführte Bedienung würde die Einführung bei den Software Engineers erleichtern und die Akzeptanz im Alltag erhöhen (F-17).
- **Erkennung von Änderungen an Datenbankobjekten:** Die Auswertung der Änderungshistorie könnte um definierte Datenbankobjekte (Tabellen, Sichten, Prozeduren) erweitert werden (F-18).
- **Direkter Aufruf aus der as1:** Die CLI-Schnittstelle ist so gestaltet, dass eine **as1**-Erweiterung für den Startzeilenaufruf nachrüstbar ist. Damit würde die Snapshot-Erstellung und Paketbildung direkt aus der gewohnten Bedienoberfläche der Engineers erfolgen.

Betrieblich:

- **Auswertung der tatsächlichen Ersparnis:** Die Kosten-Nutzen-Analyse basiert auf Annahmen. Im laufenden Betrieb sollte die tatsächlich eingesparte Zeit pro Paket erhoben werden, um die Szenarien zu validieren und die Amortisationsrechnung zu festigen.
- **Einführung bei Ammann:** Die Standardisierung der Paketerstellung entfaltet ihren vollen Nutzen erst mit dem produktiven Einsatz. Die Einführung umfasst die Schulung der Software Engineers und die Aufnahme des Tools in den internen Prozess für kundenspezifische Updatepakete.

---

9. Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und sämtliche Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, durch Quellenangaben kenntlich gemacht. Die Unterstützung durch KI-Systemen habe ich in der Erklärung zur Nutzung von Künstlicher Intelligenz transparent deklariert. Die Arbeit wurde weder als Ganzes noch in wesentlichen Teilen an einer anderen Bildungsstätte als Prüfungsleistung eingereicht.

Ort, Datum: Langenthal, 09.11.2026

Unterschrift: 

10. Verzeichnisse

10.1 Abkürzungsverzeichnis

Abkürzung	Bedeutung
as1	Anlagensteuerungssoftware der Ammann Schweiz AG für Asphalt- und Betonmischanlagen
CAM	Computer-Aided Manufacturing (rechnergestützte Fertigung)
CHF	Schweizer Franken
CLI	Command Line Interface (Kommandozeile)
CTE	Common Table Expression (SQL)
DB	Datenbank
E/A	Ein-/Ausgabe
EFZ	Eidgenössisches Fähigkeitszeugnis
F-XX	Funktionale Anforderung (Nummerierung gemäss Kapitel 7.1)
GUI	Graphical User Interface (grafische Benutzeroberfläche)
HF	Höhere Fachschule
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
MSSQL	Microsoft SQL Server
NF-XX	Nicht-funktionale Anforderung (Nummerierung gemäss Kapitel 7.1)

Abkürzung	Bedeutung
PSP	Projektstrukturplan
SDK	Software Development Kit
SFX	Self Extracting Archive (selbstentpackendes Archiv)
SHA-256	Secure Hash Algorithm (Hashwert-Länge 256 Bit)
SQL	Structured Query Language
SWOT	Strengths, Weaknesses, Opportunities, Threats
WPF	Windows Presentation Foundation
XML	eXtensible Markup Language
XPath	XML Path Language

Die projektspezifischen Nummerierungen (Muss-Ziele, Meilensteine M1-M4, Kriterien K1-K10, Risiken R1-R7 und S1-S6, Merkmale M1-M6 mit Ausprägungen A1-A4) werden bei ihrer ersten Verwendung im jeweiligen Kapitel definiert.

10.2 Abbildungsverzeichnis

Nr.	Titel	Kapitel
1	Zielscheibe der Diplomarbeit	4.2
2	Projektstrukturplan (PSP)	5.2
3	Projektablaufplanung (Soll)	5.3
4	Bewertungsmatrix der Projekt-Risiken (R1-R7)	5.5
5	Asphaltmischanlage von aussen zur Veranschaulichung der Anlagengrösse	6.1.1
6	Grundsätzlicher Aufbau der **as1** mit ihren Schichten	6.1.1
7	Message-Austausch zwischen Bedienung, Darstellung, Logik und Feldebene	6.1.1
8	Zustandsdiagramm eines virtuellen Kontrollers am Beispiel der Mineral-Dosierung	6.1.1
9	Abhängigkeiten der XML-Konfigurationsdateien und der Datenhaltung	6.1.1
10	**as1**-Standardvorlage der Steuerung (276 aktive Elemente)	6.1.1
11	**as1**-Steuerung eines Projekts durchschnittlicher Grösse (ca. 370 aktive Elemente)	6.1.1
12	Grosse **as1**-Steuerung, Konsole Mix (Misch- und Verladebereich)	6.1.1
13	Grosse **as1**-Steuerung, Konsole Dry (Gesteinstrocknung)	6.1.1
14	Ist-Prozess der Erstellung eines Updatepakets	6.1.2
15	SWOT-Analyse der gewählten Lösungsvariante (Übersicht)	7.3

Nr.	Titel	Kapitel
16	Jährliche Ersparnis der Szenarien im Vergleich zu den Entwicklungskosten	7.5.3
17	Fachliche Zustände des Paketbaus mit Exit-Codes	7.6.1
18	Klassenstruktur des As1UpdateTools (reduzierte Darstellung)	7.6.1
19	Ausgabe der Unit-Tests, sämtliche Tests sind bestanden	7.6.5
20	Erstellen und Auflisten eines Snapshots über die Kommandozeile	7.6.5
21	Erkannte Änderungen der Ausgabe des Befehls diff	7.6.5
22	Erzeugtes Updatepaket mit dem Pflichtpräfix `Package-`	7.6.5
23	Entpackte Paketstruktur mit `install.config` auf oberster Ebene	7.6.5
24	Automatisch generierte ReadMe mit Änderungsübersicht und Warnblock	7.6.5
25	Installation des erzeugten Pakets mit dem **as1HotfixInstaller**	7.6.5
26	Kontrollierter Abbruch bei fehlendem Referenz-Snapshot mit fachlicher Meldung (NF-04)	7.6.5
27	Projektablaufplanung (Ist)	8.1

10.3 Tabellenverzeichnis

Nr.	Titel	Kapitel
1	Stakeholder der Kommunikationsplanung	5.4
2	Bewertungsmatrix Eintrittswahrscheinlichkeit / Schadensausmass	5.5.1
3	Risikotabelle der Projekt-Risikoanalyse	5.5.2
4	Komponenten der **as1**	6.1.1
5	XML-Konfigurationsdateien der **as1**	6.1.1
6	Kennzahlen einer Anlagensteuerung	6.1.1
7	Hauptelemente der install.config	6.1.4
8	Installationsschritte des **as1HotfixInstaller**s	6.1.4
9	Schwachstellen des Ist-Prozesses	6.1.5
10	Morphologischer Kasten	6.2.2
11	Verträglichkeitsprüfung der Ausprägungen	6.3
12	Variantenübersicht (Merkmale je Variante)	6.4
13	Bewertungskriterien der Nutzwertanalyse	6.5.1

Nr.	Titel	Kapitel
14	Bewertungsskala der Nutzwertanalyse	6.5.2
15	Bewertung der Varianten je Kriterium	6.5.2
16	Nutzwerte der Varianten	6.5.2
17	Szenarien der Sensitivitätsanalyse	6.5.3
18	Anforderungen: Snapshot-Verwaltung	7.1.1
19	Anforderungen: Differenzerkennung XML	7.1.1
20	Anforderungen: Differenzerkennung Parameter	7.1.1
21	Anforderungen: Paketgenerierung	7.1.1
22	Anforderungen: Bedienung	7.1.1
23	Anforderungen: Erweiterte Funktionen	7.1.1
24	Nicht-funktionale Anforderungen	7.1.2
25	Umfangsentscheidung und Abgrenzung	7.1.3
26	Übersicht der Realisierungsphasen	7.2.2
27	Risikotabelle der Risiko-Analyse (Detail/Umsetzung)	7.4.2
28	Kostenannahmen der Kosten-Nutzen-Analyse	7.5.2
29	Nutzenannahmen der Kosten-Nutzen-Analyse	7.5.2
30	Szenarien der jährlichen Ersparnis	7.5.3
31	Amortisationsrechnung	7.5.4
32	Projektstruktur des As1UpdateTools	7.6.1
33	Soll-Ist-Vergleich der Phasenaufwände	8.1
34	Evaluation der Muss-Erfolgskriterien	8.2
35	Evaluation der Kann-Erfolgskriterien	8.2

Nicht aufgeführt sind die Layout-Tabellen des Lebenslaufs (Kapitel 2) und des Qualifikationsprofils (Kapitel 3) sowie die Verzeichnisse in Kapitel 10.

10.4 Quellenverzeichnis

Firmeninterne Quellen (Ammann Schweiz AG)

- [1] Ammann Schweiz AG: DevDoc, interne Wissensdatenbank zur Anlagensteuerungssoftware **as1** (Versionsstände V6.0 und V6.1), in diversen verteilten Artikeln dokumentiert (konsultiert von August bis November 2026)
- [2] Ammann Schweiz AG: HotfixInstaller, interne Dokumentation (install.config-Schema, Installationsschritte), Stand 1. November 2026, DevDoc
- [3] Eigene Auswertung des Verfassers: Erhebung der Kennzahlen aus den SQL-Datenbanken von vier Kundensystemen, Ammann Schweiz AG, Oktober 2026

Methodische Quellen (Projektmanagement und Bewertungsmethoden)

- [4] Schweizerische Eidgenossenschaft: HERMES 2022, die Projektmanagementmethode des Bundes. https://www.hermes.admin.ch/ (abgerufen am 3. November 2026)
- [5] Springer Gabler Wirtschaftslexikon: Nutzwertanalyse. https://wirtschaftslexikon.gabler.de/definition/nutzwertanalyse-42926 (abgerufen am 3. November 2026)
- [6] ISO 31000:2018: Risk management — Guidelines. https://www.iso.org/standard/65694.html (abgerufen am 3. November 2026)

Technische Dokumentationen

- [7] Microsoft: Dokumentation zum .NET Framework und zur Programmiersprache C#. https://learn.microsoft.com/dotnet/ (abgerufen am 3. November 2026)
- [8] Microsoft: Dokumentation zu SQL Server, Common Table Expressions. https://learn.microsoft.com/sql/ (abgerufen am 3. November 2026)
- [9] NUnit Organization: NUnit-Dokumentation. https://docs.nunit.org/ (abgerufen am 3. November 2026)
- [10] Newtonsoft: Dokumentation zu Json.NET. https://www.newtonsoft.com/json (abgerufen am 3. November 2026)
- [11] Igor Pavlov: 7-Zip-Dokumentation. https://www.7-zip.org/ (abgerufen am 3. November 2026)
- [12] Mermaid Project: Dokumentation der Diagramm-Sprache Mermaid. https://mermaid.js.org/ (abgerufen am 3. November 2026)

Verwendete Werkzeuge (erwähnt in der Ist-Analyse)

- [13] Notepad++: https://notepad-plus-plus.org/ (abgerufen am 3. November 2026)
- [14] WinMerge: https://winmerge.org/ (abgerufen am 3. November 2026)

---

11. Anhang

- Benutzerdokumentation / CLI-Handbuch
- Testberichte (Abschnitte 11.2 bis 11.5)

11.1 Benutzerdokumentation: As1UpdateTool (CLI-Handbuch)

1. Einführung

Das As1UpdateTool unterstützt Software Engineers der Ammann Schweiz AG bei der Erstellung kundenspezifischer Updatepaketen für **as1**-Anlagensteuerungen. Es erstellt automatisch die Differenz zwischen einem Referenzzustand (Snapshot) und dem aktuellen Zustand einer Anlage und paketiert die erkannten Änderungen als installierbares Paket für den **as1HotfixInstaller**.

Das Tool ist als Kommandozeilen-Anwendung ausgeführt und wird vollständig über Parameter bedient. Es ist keine grafische Oberfläche vorhanden.

2. Voraussetzungen

- Windows-System mit Zugriff auf die **as1**-Installation und ihre XML-Konfigurationsdateien
- .NET Framework
- Lesender Zugriff auf die SQL-Datenbank der Anlage (Windows-Authentifizierung, Integrated Security). Das Tool speichert und verwaltet keine Datenbank-Credentials.
- Installierte `7z.exe` für die Erzeugung des selbstentpackenden Archivs
- Die Standardpfade sind im Tool vordefiniert und werden aus der Umgebung abgeleitet. Der Datenbankzugriff wird aus der `System.config` der **as1** gelesen. Eine Anpassung der Pfade ist aktuell nur im Code möglich.

3. Befehle im Überblick

Befehl	Zweck
`As1UpdateTool.Cli snapshot create`	Snapshot der XML-Konfigurationsdateien erstellen
`As1UpdateTool.Cli snapshot list`	Vorhandene Snapshots auflisten
`As1UpdateTool.Cli diff`	Snapshot mit aktuellem Zustand vergleichen
`As1UpdateTool.Cli package`	Updatepaket erzeugen

Wird das Tool ohne Befehl oder mit einem unbekannten Befehl aufgerufen, gibt es eine usage-Meldung mit der Befehlsübersicht aus.

4. Typischer Ablauf

1. **Vor der Arbeit:** Snapshot erstellen (`snapshot create`). Damit wird der Referenzzustand fixiert. Das Erstellungsdatum definiert den Anfang des Änderungszeitraums.
2. **Arbeiten:** Anpassungen an der Steuerung vornehmen (Parametrieren, XML-Konfiguration ändern).
3. **Prüfen:** `diff` aufrufen. Das Tool listet die geänderten XML-Dateien und die Parameteränderungen aus

der Änderungshistorie auf.

4. **Paketieren:** `package` aufrufen. Das Tool erzeugt die `install.config`, die ReadMe und das selbstentpackende Archiv mit dem Präfix `Package-`.
5. **Installieren:** Das Paket mit dem **as1HotfixInstaller** installieren. Die ReadMe wird nach der Installation automatisch angezeigt.

5. Befehle im Detail

5.1 snapshot create

Erzeugt einen Snapshot der drei XML-Konfigurationsdateien der **as1**. Der Snapshot wird in einem Anlagenordner mit zeitlich benanntem Unterordner abgelegt. Jedem Snapshot wird die Metadaten-Datei `snapshot.json` beigelegt mit den Feldern:

- **Erstellungsdatum:** Zeitpunkt der Erstellung, zugleich Beginn des Änderungszeitraums
- **Anlage:** Name der Anlage, aus der Datenbank gelesen
- **Engineer:** ausführende Person
- **Kommentar:** frei wählbare Beschreibung des Vorhabens

Der Kommentar kann optional als Argument mitgegeben werden. Wird er weggelassen, fragt das Tool nicht nach und trägt keinen Kommentar ein. Alle übrigen Felder werden automatisch ermittelt. Das Tool führt keine interaktiven Eingaben durch, der vollständige Ablauf ist ohne Interaktion ausführbar.

5.2 snapshot list

Zeigt die vorhandenen Snapshots, gruppiert nach Anlage. Dient der Auswahl des Referenz-Snapshots für `diff`.

5.3 diff

Vergleicht den Referenz-Snapshot mit dem aktuellen Zustand. Das Tool meldet:

- **Geänderte XML-Dateien:** erkannt über einen Hash-Vergleich der Konfigurationsdateien
- **Parameteränderungen:** ermittelt aus der Änderungshistorie der Datenbank ab dem Erstellungsdatum des Snapshots, aufgelöst über die Parameterpfade
- **Unsichere Parameteränderungen:** Parameter, die im Änderungszeitraum mehrfach geändert wurden, werden vom automatischen Import ausgeschlossen und als manuell zu prüfende Fälle ausgewiesen

Der Befehl verändert nichts. Er ist rein lesend und eignet sich zur Kontrolle vor der Paketerstellung.

5.4 package

Erstellt das Updatepaket aus allen erkannten Änderungen. Der Ablauf im Einzelnen:

1. Prüfung der Voraussetzungen (Referenz-Snapshot vorhanden, Datenbank erreichbar, Änderungen vorhanden)
2. Aufbau der Paketstruktur in einem temporären Staging-Ordner: `install.config` auf oberster Ebene, Zieldateien ab `ComaData` bzw. `ProgramData`, Parameterexport im Ordner `Coma\Database`, ReadMe als HTML-Datei

3. Erzeugung der `install.config` gemäss dem dokumentierten Schema mit den zwölf definierten Installationsschritten, verteilt auf die Phasen PreInstallation, Installation und PostInstallation
4. Erzeugung der ReadMe aus den Metadaten und den erkannten Änderungen, inklusive Warnblock für unsichere Parameteränderungen
5. Aufruf der `7z.exe` zur Erzeugung des selbstentpackenden Archivs mit dem Pflichtpräfix `Package-`

Das Präfix `Package-` ist zwingend. Ohne das Präfix behandelt der **as1HotfixInstaller** das Archiv als Hotfix und ignoriert die `install.config`.

6. Exit-Codes und Fehlerbehandlung

Fachliche Zustände werden über Exit-Codes quittiert. Es gilt:

Exit-Code	Bedeutung
0	Vorgang erfolgreich abgeschlossen
1	Aufruf unvollständig oder Befehl unbekannt, das Tool gibt die usage-Meldung aus
2	Technischer Fehler, das Tool bricht mit der Meldung "Fehler: <Beschreibung>" ab

Fachliche Abbruchbedingungen (kein Referenz-Snapshot, keine Datenbankverbindung, keine Änderungen) werden mit verständlichen Meldungen quittiert und brechen den Vorgang kontrolliert ab, ohne unbearbeitete Ausnahmen.

Typische Meldungen und ihre Ursachen:

Meldung	Ursache	Massnahme
usage-Meldung	Tool ohne Befehl oder mit unbekanntem Befehl aufgerufen	Befehl gemäss Übersicht in Abschnitt 3 verwenden
Kein Referenz-Snapshot vorhanden	`diff` oder `package` ohne bestehenden Snapshot aufgerufen	Zuerst `snapshot create` ausführen oder mit `snapshot list` einen gültigen Snapshot wählen
Keine Datenbankverbindung	Datenbank nicht erreichbar oder keine Berechtigung	Verbindung und Berechtigung prüfen, Vorgang wiederholen
Keine Änderungen gefunden	Zwischen Snapshot und Ist-Stand liegen keine Änderungen vor	Prüfen, ob der richtige Snapshot referenziert wird, sonst ist kein Updatepaket nötig
Snapshot einer fremden Anlage	Referenz-Snapshot gehört zu einer anderen Anlage	Snapshot der betroffenen Anlage verwenden
"Fehler: <Beschreibung>" (Exit-Code 2)	Technischer Fehler, zum Beispiel beim 7-Zip-Aufruf	Meldung lesen, Ursache beheben (z.B. `7z.exe` prüfen), Vorgang wiederholen

7. Ergebnis des Paketbaus

Das erzeugte Paket ist ein selbstentpackendes Archiv mit dem Namen `Package-bezeichnung.exe`. Es enthält:

- `install.config` auf oberster Ebene, mit den Installationsschritten für den **as1HotfixInstaller**
- **Datei-Payloads** in der Ordnerstruktur der Ziele, beginnend bei `ComaData` bzw. `ProgramData`
- **ParamTransfer-Importdatei** im Ordner `Coma\Database`, mit den erkannten Parameteränderungen
- **ReadMe** als HTML-Datei, mit Kopfdaten, Änderungstabellen, Warnblock für manuell zu prüfende Parameter und den Installationsschritten

Die Installation erfolgt mit dem **as1HotfixInstaller**. Der Silent-Modus wird über das Konsolenargument `-s` aktiviert. Nach der Installation wird die ReadMe über den Schritt `ShowHtml` automatisch geöffnet. Die inhaltliche Korrektheit der erkannten Parameteränderungen prüft das Tool nicht, sie bleibt Aufgabe des Engineers und wird durch die ReadMe-Übersicht unterstützt.

8. Grenzen des Tools

- **Keine Schema-Validierung der install.config:** Die erzeugte `install.config` wird nicht gegen ein Schema geprüft, sie wird im Realversuch gegen die Erwartungen des **as1HotfixInstaller**s getestet.
- **Keine Auswertung allgemeiner Datenbankobjekte:** Nur Parameteränderungen werden berücksichtigt. Tabellen, Sichten und Prozeduren sind nicht Teil des Pakets.
- **Unsichere Parameteränderungen:** Mehrfach geänderte Parameter werden bewusst nicht automatisch importiert. Sie sind im ReadMe-Warnblock aufgeführt und manuell zu prüfen.
- **Pfade:** Standardpfade sind vordefiniert und nur im Code änderbar.
- **Nur lesender Datenbankzugriff:** Das Tool schreibt nie in die Datenbank der Anlage.

11.2 Testbericht 1: Unit-Tests der Kernlogik

Projekt: Diplomarbeit As1UpdateTool, Ammann Schweiz AG
Getestetes Objekt: As1UpdateTool.Core, As1UpdateTool.Data, As1UpdateTool.Cli (Kernlogik ohne Fremdsysteme)
Testwerkzeug: NUnit (Test-Framework), dotnet test
Datum der Ausführung: 3. November 2026
Verfasser: Martin Plüss

1. Ziel und Umfang

Gegenstand dieses Berichts ist die Verifikation der Kernlogik des **As1UpdateTools** durch Unit-Tests. Fremdsystemgrenzen (SQL-Schematypen, 7-Zip-Verhalten, Erwartungen des **as1HotfixInstaller**s) sind mit Unit-Tests prinzipbedingt nicht abdeckbar und werden im Realversuch geprüft (Testbericht 3). Die Datenbank- und die Paketgrenze liegen hinter den Schnittstellen `IAs1DbAccess` und `IPackageBuilder`. In den Tests werden sie durch die Grenz-Fakes `FakeAs1DbAccess` und `FakePackageBuilder` ersetzt. Dadurch sind sämtliche Tests ohne **as1**-Installation, ohne Datenbank und ohne Netzwerk lauffähig.

2. Testumfang

Tabelle TB1.1: Übersicht der Test-Fixtures

Test-Fixture	Prüfling	Testgegenstand
SnapshotManagerTests	Snapshot-Verwaltung	Anlegen des Anlagenordners mit zeitlich benanntem Unterordner, Kopieren der drei XML-Konfigurationsdateien, Schreiben und Lesen von `snapshot.json` inklusive Pflichtfelder Erstellungsdatum, Anlage, Engineer, Kommentar (F-02), Snapshot-Liste
XmlDifferTests	Differenzerkennung	Hash-Vergleich der XML-Konfigurationsdateien zwischen Snapshot und Ist-Stand, geänderte, unveränderte und neue Dateien werden korrekt unterschieden (F-04, F-05)
XmlStructuralDifferTests	XML-Differenzierung	Strukturelle Änderungen in XML-Dateien, XPath-Anker
ParameterHistoryAggregatorTests	Änderungshistorie	Auswertung der Parameterhistorie ab Snapshot-Datum, mehrfach geänderte Parameter werden als unsichere Fälle erkannt (S2)
DiffOrchestratorTests	Diff-Kette	Zusammenspiel Snapshot zu Änderung zu Erkennung über mehrere Bausteine hinweg

Test-Fixture	Prüfling	Testgegenstand
SystemConfigReaderTests	System.config	Ableiten der Datenbankverbindung aus der `System.config` der **as1**
InstallConfigBuilderTests	install.config	Schrittzuordnung zu den Phasen PreInstallation, Installation und PostInstallation, nur die zwölf definierten Installationsschritte, konsistente Pfadangaben gemäss Paketstruktur
PackageBuilderTests	Paketstruktur	Staging-Logik: `install.config` auf oberster Archivebene, Datei-Payloads ab `ComaData` bzw. `ProgramData`, SQL-Skripte und Parameterexporte in `Coma\Database`, Pflichtpräfix `Package-` (F-13, F-14)
ReadMeBuilderTests	ReadMe	Kopfdaten, Änderungstabellen, Warnblock für manuell zu prüfende Parameter, HTML-Escaping (F-12, F-19)
UpdateOrchestratorTests	Gesamtablauf	Orchestrierung des Ablaufs über die Befehle `snapshot`, `diff` und `package` (F-15), fachliche Zustände und Abbruchbedingungen, Exit-Codes
Total		**77 Tests in zehn Test-Fixtures**

3. Testdurchführung

Sämtliche Tests wurden mit `dotnet test` ausgeführt. Die beiden Testgattungen sind bewusst getrennt: Klassen-Einzeltests prüfen jeden Baustein für sich, Zusammenspiel-Tests prüfen die Kette über mehrere Bausteine hinweg.

4. Resultate

Tabelle TB1.2: Ergebnisübersicht

Kennzahl	Wert
Tests insgesamt	77
Davon bestanden	77
Davon fehlgeschlagen	0
Davon übersprungen	0
Testdauer	1.9 Sekunden

Sämtliche 77 Tests sind bestanden. Es gab weder fehlgeschlagene noch übersprungene Tests.

5. Beurteilung

Die Kernlogik des **As1UpdateTools** ist durch die Unit-Tests flächendeckend gesichert. Die Grenzen zu den Fremdsystemen bleiben offen und werden durch den Realversuch abgedeckt (Testbericht 3). Das Ergebnis bestätigt das in Abschnitt 7.6.5 der Arbeit beschriebene zweigeteilte Testkonzept.

11.3 Testbericht 2: Fehlerfall- und Abbruchtests

Getestetes Objekt: As1UpdateTool.Cli mit Kernlogik (Anforderung NF-04, Gegenmassnahme S2)
Datum der Ausführung: 3. November 2026
Verfasser: Martin Plüss

1. Ziel und Umfang

Gefordert ist, dass Fehler mit verständlichen Meldungen quittiert werden und den Vorgang kontrolliert abbrechen. Getestet werden die fachlichen Zustände und Abbruchbedingungen des Ablaufs. Herbeigeführt werden die Fehlerfälle gezielt, indem die jeweilige Voraussetzung entfernt oder verändert wird.

2. Prüffälle

Tabelle TB2.1: Prüffälle der Fehlerfall- und Abbruchtests

Nr.	Prüffall	Herbeigeführte Bedingung	Erwartetes Verhalten	Resultat
1	Kein Referenz-Snapshot vorhanden	`diff` bzw. `package` ohne bestehenden Snapshot aufgerufen	Verständliche Meldung, kontrollierter Abbruch mit fachlichem Exit-Code	erfüllt
2	Keine Datenbankverbindung	DB-Server nicht erreichbar bzw. Verbindung getrennt	Verständliche Meldung, kontrollierter Abbruch, keine unbehandelte Ausnahme	erfüllt
3	Keine Änderungen seit Snapshot	`diff` ohne Änderungen zwischen Snapshot und Ist-Stand	Meldung, dass keine Änderungen vorliegen, kontrollierter Abbruch	erfüllt
4	Unsichere Parameteränderung (mehrfach geändert)	Parameter seit dem Snapshot mehrfach in der Historie verändert	Parameter wird vom automatischen Import ausgeschlossen und im ReadMe-Warnblock mit Handanweisung aufgeführt (S2)	erfüllt
5	Snapshot einer fremden Anlage	Referenz-Snapshot einer anderen Anlage angegeben	Snapshot wird nicht als Referenz verwendet, Abbruch mit Meldung	erfüllt
6	Ungültige Pfadangaben	Zielordner nicht vorhanden bzw. nicht schreibbar	Verständliche Meldung, kontrollierter Abbruch	erfüllt

3. Beurteilung

Alle geprüften Fehlerfälle werden mit konkreten Meldungen quittiert und brechen den Vorgang kontrolliert ab. Es traten keine unbehandelten Ausnahmen auf. Die Anforderung NF-04 ist damit erfüllt. Die Gegenmassnahme zu S2 (Ausschluss unsicherer Parameteränderungen mit Ausweis im ReadMe-Warnblock) ist nachweislich umgesetzt.

11.4 Testbericht 3: Realversuch auf der as1-Entwicklungsumgebung

Getestetes Objekt: As1UpdateTool im Zusammenspiel mit den Fremdsystemen (SQL Server, 7-Zip, **as1HotfixInstaller**)
Testumgebung: as1-Entwicklungsumgebung, Softwareversionen V6.0 und V6.1
Datum der Ausführung: 3. November 2026
Verfasser: Martin Plüss

1. Ziel und Umfang

Der Realversuch validiert die Systemgrenzen, die mit Unit-Tests nicht abdeckbar sind: SQL-Schematypen, das Verhalten von 7-Zip bei der SFX-Erstellung und die Erwartungen des **as1HotfixInstaller**s. Er wird als vollständiger Komplettdurchlauf mit einem bestehenden Kundenprojekt durchgeführt und deckt damit das Muss-Erfolgskriterium 7 (Kapitel 8.2 der Arbeit) ab.

Der Realversuch wurde auf beiden Softwareversionen mit dem gleichen Szenario durchgeführt, je auf einer **as1**-Entwicklungsumgebung mit V6.0 und einer mit V6.1. In beiden Umgebungen wurde ein zusätzlicher Doseur (Komponentenzugabe) in die Steuerung implementiert. Dafür wurden bei allen drei XML-Konfigurationsdateien Änderungen vorgenommen und die Parameter im betroffenen virtuellen Kontroller angepasst. Damit deckt das Szenario beide Änderungsarten des Tools ab: Änderungen an den XML-Konfigurationsdateien und Parameteränderungen.

2. Ablauf und Prüfschritte

Tabelle TB3.1: Prüfschritte des Komplettdurchlaufs

Nr.	Schritt	Prüfkriterium	Resultat
1	Snapshot erstellen (`As1UpdateTool.Cli snapshot create`)	Snapshot-Ordner und `snapshot.json` mit gültigen Metadaten werden erzeugt	erfüllt
2	Anpassungen vornehmen	Parametrierung und XML-Konfiguration werden auf der Entwicklungsumgebung verändert	erfüllt
3	Änderungen erkennen (`diff`)	Liste enthält die geänderten XML-Dateien und die Parameteränderungen aus der Historie	erfüllt
4	Paket erstellen (`package`)	Selbstentpackendes Archiv mit Präfix `Package-` wird erzeugt, 7-Zip-SFX-Erstellung meldet Erfolg	erfüllt
5	Paketstruktur prüfen	`install.config` auf oberster Archivebene, Zieldateien ab `ComaData` bzw. `ProgramData`, Parameterexport im Ordner `Coma\Database`, ReadMe im Paket	erfüllt
6	Installation mit dem **as1HotfixInstaller**	Paket wird akzeptiert und ohne manuelle Nachbearbeitung installiert, ReadMe wird über den `ShowHtml`-Schritt automatisch geöffnet	erfüllt

Nr.	Schritt	Prüfkriterium	Resultat
7	Installierte Werte verifizieren	Veränderte Werte sind auf der Anlage nach der Installation wirksam, sie stimmen mit den erkannten Änderungen überein	erfüllt

3. Festgestellte Abweichungen

Während des Realversuchs wurde ein Typ-Mismatch zwischen einer SQL-Spalte (Guid statt String) und dem Lesezugriff festgestellt. Die Abweichung war mit Unit-Tests prinzipbedingt nicht sichtbar, weil die Datenbankgrenze in den Tests durch einen Fake ersetzt wird. Der Mismatch wurde behoben und der Komplettdurchlauf danach wiederholt. Der zweite Durchlauf verlief vollständig erfolgreich.

4. Beurteilung

Der Realversuch bestätigt die Architektur-Annahmen (SFX-Archivstruktur, Verhalten des **as1HotfixInstaller**s) und die Installierbarkeit des erzeugten Pakets ohne Nachbearbeitung. Er bestätigt zugleich die Notwendigkeit der Trennung von Unit-Tests und Realversuch, wie sie in Abschnitt 7.6.5 der Arbeit begründet ist. Das Muss-Erfolgskriterium "Installation ohne Nachbearbeitung" ist damit am realen System nachgewiesen.

11.5 Testbericht 4: Abnahmetest der Muss-Erfolgskriterien

Getestetes Objekt: Gesamtlösung As1UpdateTool gegen die Muss-Erfolgskriterien des Pflichtenhefts
Datum der Ausführung: 3. November 2026
Verfasser: Martin Plüss

1. Ziel und Umfang

Der Abnahmetest prüft die acht Muss-Erfolgskriterien aus Kapitel 4.1 der Arbeit gegen den realisierten Stand. Die Erfolgskriterien wurden für die Bewertung nicht angepasst. Die Nachweise stützen sich auf die Testberichte 1 bis 3 sowie auf die Dokumentation in Kapitel 7.6 der Arbeit.

2. Prüfergebnis

Tabelle TB4.1: Abnahmetest der Muss-Erfolgskriterien

Nr.	Erfolgskriterium	Erfolgsgrad	Nachweis
1	Das Tool erkennt Änderungen an Parametern zwischen zwei Softwarezeitpunkten automatisch und berücksichtigt diese im Updatepaket.	erfüllt	Auswertung der Änderungshistorie über die lesende CTE-Abfrage (F-07), Berücksichtigung als ParamTransfer-Importdatei (F-08), gesichert durch Testbericht 1 und 3
2	Änderungen an aktiven Steuerungselementen werden automatisch erkannt und dokumentiert.	erfüllt	Hash-Vergleich der XML-Konfigurationsdateien (F-04, F-05), ReadMe-Dokumentation (F-12)
3	Die erzeugten Updatepakete können mit dem HotfixInstaller ohne manuelle Nachbearbeitung installiert werden.	erfüllt	Komplettdurchlauf mit Installation und Verifikation (Testbericht 3)
4	Das automatisch generierte ReadMe dokumentiert die erkannten Änderungen sowie die notwendigen Installationsinformationen nachvollziehbar.	erfüllt	HTML-ReadMe mit Kopfdaten, Änderungstabellen, Warnblock und Installationsschritten (F-12, F-19)
5	Die Lösung funktioniert mit den **as1**-Softwareversionen V6.0 und V6.1.	erfüllt	Komplettdurchlauf je auf einer V6.0- und einer V6.1-Entwicklungsumgebung mit dem gleichen Szenario (zusätzlicher Doseur, Änderungen an allen drei XML-Konfigurationsdateien, Parameteranpassungen im virtuellen Kontroller, vgl. Testbericht 3)

Nr.	Erfolgskriterium	Erfolgsgrad	Nachweis
6	Das Tool kann vollständig über Kommandozeilenparameter ausgeführt werden.	erfüllt	Befehle `snapshot`, `diff` und `package` ohne Interaktion ausführbar (F-15), fachliche Zustände werden über Exit-Codes quittiert
7	Die Lösung wird anhand mindestens eines bestehenden Kundenprojekts erfolgreich getestet und dokumentiert.	erfüllt	Komplettdurchlauf mit bestehendem Kundenprojekt auf der **as1**-Entwicklungsumgebung (Testbericht 3)
8	Die technische Dokumentation beschreibt Aufbau, Funktionsweise und Einschränkungen der Lösung nachvollziehbar.	erfüllt	Kapitel 7.6.1 bis 7.6.6 der Arbeit, Einschränkungen in 7.6.5 und 7.6.6 ausgewiesen

3. Beurteilung

Sämtliche acht Muss-Erfolgskriterien sind erfüllt. Die Arbeit erreicht damit ihr Kernziel vollständig. Die Grenzen der Lösung (nicht realisierte Schema-Validierung F-11, GUI-Eingabevalidierung F-17, Auswertung allgemeiner Datenbankobjekte F-18) sind bewusste Umfangsentscheidungen und in den Kapiteln 7.1.3 und 7.6.6 der Arbeit dokumentiert.