



Abschlussprüfung Sommer 2026

Fachinformatiker für Anwendungsentwicklung

Dokumentation der betrieblichen Projektarbeit

Thema

Optimierung der Auftragsverarbeitung im Warehouse Control Systems mittels
Prozesssteuerung in C#

Prüfling

Kai Kröger

E-Mail: kai.oliver.k@web.de

Betreuer

Sebastian Badour

E-Mail: s.badour@gebhardt-group.com

Betrieb

Gebhardt Fördertechnik GmbH

Neulandstraße 28, 74889 Sinsheim

Inhaltsverzeichnis

Abbildungsverzeichnis	4
Tabellenverzeichnis.....	4
1. Einleitung.....	1
1.1 Das Unternehmen	1
1.2 Projektziel.....	1
1.3 Projektabgrenzung.....	1
1.4 Projektumfeld & Schnittstellen	2
1.4.1 Administrative Schnittstelle (WMS/Host)	2
1.4.2 Ausführungsebene (SPS/PLC)	2
1.4.3 Datensynchronisation und Technologie-Stack.....	2
1.4.4 Test- und Simulationsumgebung.....	2
1.4.5 Relevante Prozesse	3
2. Projektplanung	3
2.1 Projektphasen & Zeitplanung.....	3
2.1.1 Vorgehensmodell	3
2.1.2 Tabellarische Zeitplanung	4
2.2 Ressourcenplanung.....	4
2.2.1 Hardware.....	4
2.2.2 Software.....	4
2.2.3 Personal.....	5
2.3 Kostenplanung.....	5
3. Analyse & Konzept.....	5
3.1 Ist-Analyse.....	5
3.2 Soll-Konzept.....	6
3.3 Wirtschaftlichkeitsbetrachtung.....	6
4. Design & Architektur.....	6
4.1 Technische Entscheidungen.....	6
4.2 Systemarchitektur.....	7
5. Realisierung.....	7
5.1 Implementierung	7
5.1.1 Datenmodell anpassen	7
5.1.2 Anpassung HostBooking Prozess	8
5.1.3 Anpassung ConveyorDispo Prozess.....	8
5.2 Qualitätssicherung & Test.....	10
5.2.2 Testumgebung	10

5.2.3	Strukturierte Testszenarien	10
5.2.4	Aufgetretene Fehler	13
5.3	Abweichungen vom Soll-Konzept.....	14
6	Projektabschluss.....	14
6.1	Soll-Ist-Vergleich.....	14
6.1.2	Soll-Ist-Projektziele	14
6.1.3	Soll-Ist-Zeitplanung.....	15
6.2	Fazit & Ausblick	16
6.2.1	Persönliches und fachliches Fazit	16
6.2.2	Ausblick.....	16
Anhang	17
Kundendokumentation	17
Quellcode	17
Abnahmeprotokoll	17
Glossar	17
Prozess & Ablaufdiagramme	17
Quellenverzeichnis	17

Abbildungsverzeichnis

Abbildung 1 Taskboard.....	3
Abbildung 2 Ausschnitt Entity.....	7
Abbildung 3 Ausschnitt EF Core Entity Builder	7
Abbildung 4 IsDepartureReady Query.....	8
Abbildung 5 Flagging Restart Orders.....	8
Abbildung 6 Flagging Initial Orders	8
Abbildung 7 OrderListItem	8
Abbildung 8 departureFlaggedOrder Query.....	9
Abbildung 9 Scheduling	9
Abbildung 10 Pending Departure Orders Query.....	9
Abbildung 11 Order Restart.....	9
Abbildung 12 Pendingauftragsstart.....	10

Tabellenverzeichnis

Tabelle 1 Tabellarische Zeitplanung.....	4
Tabelle 2 Testmatrix	11
Tabelle 3 Soll-Ist-Projektziele	14
Tabelle 4 Soll-Ist-Zeitplanung.....	15

1. Einleitung

1.1 Das Unternehmen

Die Gebhardt Fördertechnik GmbH ist ein international agierendes Familienunternehmen mit Sitz in Sinsheim, das sich auf die Entwicklung und Herstellung modularer Intralogistiksysteme spezialisiert hat. Mit über 70 Jahren Erfahrung bietet Gebhardt ein breites Spektrum an Lösungen – von der Förder- und Lagertechnik bis hin zu komplexen Warehouse Control Systemen (WCS). Ein zentraler Baustein des Portfolios ist die Softwareplattform GEBHARDT StoreWare, die Materialfluss- (MFS), Lagerverwaltungs- (LVS) und Visualisierungssysteme integriert, um eine effiziente Steuerung und Kontrolle der Warenströme in automatisierten Lagern zu ermöglichen.

1.2 Projektziel

Das Ziel des Projekts ist die Optimierung und Stabilisierung der Auftragsverarbeitung innerhalb des WCS. Aktuell führt die dezentrale Logik beim Starten von Transportaufträgen (OrdersHost) zu Race Conditions, da parallellaufende Prozesse die Möglichkeit haben, simultan denselben Auftrag zu starten. Dies resultiert bei ungünstigem Timing in inkonsistenten Datenbeständen und nicht abschließbaren Auftragsleichen innerhalb der Datenbank. Im Rahmen dieser Arbeit wird die Start-Logik im Prozess ConveyorDispo zentralisiert. Durch gezieltes Refactoring und die Implementierung thread-sicherer Mechanismen wird sichergestellt, dass Aufträge exklusiv verarbeitet werden, was die Fehleranfälligkeit der Anlage signifikant reduziert, und die Stabilität steigert.

1.3 Projektabgrenzung

Um den strengen Zeitrahmen von 80 Stunden zu wahren, wurde eine präzise Abgrenzung des Leistungsumfangs vorgenommen, um eine unkontrollierte Ausweitung des Projektumfangs zu verhindern.

Die eigene Entwicklungs- & Analyseleistung konzentriert sich auf folgende Schwerpunkte:

- **Architekturanalyse & Konzeptentwurf:** Detaillierte Codeanalyse der Prozesse HostBooking und ConveyorDispo zur Feststellung der zugrunde liegenden Race Condition. Identifizierung der relevanten Threads innerhalb der beiden Prozesse sowie der Codestellen. Konzeptionierung einer zustandsorientierten, zentralen Prozesssteuerung im ConveyorDispo Prozess mittels Flag.
- **Datenmodellierung:** Anpassung des Datenmodells OrdersHost um das DepartureFlag innerhalb des Codes für das Object-Relational-Mapping mit Entity Framework Core und der Datenbank selbst.
- **Zentralisierung der Startlogik:** Umbau des DepartureNotificationHandler Threads innerhalb von HostBooking. Entzug der Auftragsstartbefugnis hin zum flagbasierten Eventmelder. Des Weiteren werden die zyklischen Threads OrderManager und StartInitialOrdersHost zur Auswertung und zentralen Kapazitätsprüfung der Departureaufträge umgeschrieben.
- **Integrationstest:** Entwicklung und Durchführung von Praxisnahen Testszenarios innerhalb einer Emulation der Anlage. Dokumentation innerhalb einer strukturierten Testmatrix

Die folgenden angrenzenden Themengebiete wurden aus dem Projektrahmen ausgeschlossen:

- **Altssystem-Logik für Etra Boxen & PTL Sonderfunktionen:** Behälter vom Typ EtraBox und manuelle Bestätigungen an „Pick-To-Light“ Plätzen verbleiben dezentral im HostBooking. Diese

erfordern manuelle Sonderbehandlungen durch das Personal und wurden bewusst ausgegrenzt, um das Zeitfenster nicht zu überschreiten.

- **Keine Modifikation der WMS-Schnittstelle:** Die Schnittstelle zum übergeordneten Host-Systems wird als gegeben und stabil betrachtet. Es werden keine Anpassungen auf WMS-Ebene vorgenommen.

1.4 Projektumfeld & Schnittstellen

Das Warehouse Control System (WCS) fungiert innerhalb der Logistikkette als entscheidende Middleware. Es bildet die Brücke zwischen der administrativen Ebene (WMS) und der physischen Ausführungsebene (SPS).

1.4.1 Administrative Schnittstelle (WMS/Host)

Die Anbindung an übergeordnete Lagerverwaltungssysteme (WMS) ist konfigurierbar und unterstützt zwei Kommunikationswege:

Datenbankgestützter Datenaustausch: In Standard-Szenarien erfolgt der Austausch über dedizierte Schnittstellentabellen:

Wcs.FromWms für eingehende Nachrichten und **Wcs.ToWms** für ausgehende Rückmeldungen. Die Kommunikation verläuft asynchron: Ein WMS-Prozess schreibt Auftragsdaten in die Datenbank, die vom WCS-Prozess zyklisch gepollt und in interne initiale Transportaufträge (**OrdersHost**) transformiert werden.

Service-orientierte Architektur (REST-API): Für die Anbindung externer WMS-Anbieter oder Cloud-Systeme stellt das WCS einen **REST-Server** bereit. Hierbei erfolgt der Datenaustausch via HTTP/JSON-Requests, was eine nahtlose Integration in moderne IT-Infrastrukturen ermöglicht.

1.4.2 Ausführungsebene (SPS/PLC)

Die unterlagerte Kommunikation mit der Fördertechnik-Hardware erfolgt über **unverschlüsselte TCP/IP-Telegramme** in Echtzeit. Dieser Bereich ist für die physische Sicherheit und den Durchsatz der Anlage kritisch:

Event-Handling: Sobald eine Ladeinheit (LE) einen Arbeitsplatz verlassen will, sendet die SPS eine **DepartureNotification**. Dieses Telegramm enthält unter anderem die Identifikationsnummer der LE sowie die aktuelle Position.

Handshake-Verfahren: Das WCS antwortet mit einem **Departure** Telegramm. Dieser Fahrbefehl ist die notwendige logische Freigabe für die SPS, um die LE physisch vom Platz wegzubewegen. Ohne dieses Telegramm verbleibt das Transportgut im Stillstand.

1.4.3 Datensynchronisation und Technologie-Stack

Die prozessübergreifende Datenhaltung und Synchronisation innerhalb des WCS basiert auf einem **Microsoft SQL Server**. Der Datenzugriff wird über das **Entity Framework Core** abstrahiert, um eine objektorientierte Verarbeitung der Materialflussdaten in C# zu ermöglichen.

1.4.4 Test- und Simulationsumgebung

Zur Absicherung der Implementierung wird eine hochperformante **Emulation** der Fördertechnik innerhalb von Emulate3D eingesetzt. Diese umfasst die Förderanlage selbst, die Telegrammkommunikation zwischen WCS und der Fördertechnik mittels EMC Runner, die WCS-

Prozesse und eine Testdatenbank. Diese fungiert als "Digitaler Zwilling" der physischen Anlage und ermöglicht die Simulation von praxisnahen Szenarien.

1.4.5 Relevante Prozesse

Im Folgenden werden, die im Fokus der Arbeit stehenden Prozesse des WCS im Detail beschrieben, um die formalen Projektaufgaben zu erfüllen:

Der HostBooking Prozess: Dieser asynchrone Prozess verarbeitet alle eingehenden Nachrichten und Transportaufträge des übergeordneten WMS. Seine Hauptaufgaben sind das Einlesen der eingehenden Daten an der WMS-Schnittstelle (z.b. FromWMS Tabelle), das Anlegen initialer Transportaufträge in der Tabelle OrdersHost und die Pflege der LE-Stammdaten. Im IST-Zustand kann der Prozess in Spezialfällen Transportaufträge selbst starten und Freigabetelegramme an die SPS senden. Der Prozess greift auf die Datenbanktabellen FromWMS (welche vom Hostsystem befüllt werden) und ToWMS (welche von HostBooking selbst befüllt wird) zu. HostBooking ist ein Service, er läuft permanent im Hintergrund und reagiert auf Änderungen in der Datenbank oder auf eingehende Telegramme.

Der ConveyorDispo Prozess: Er ist das mathematische und logische Herzstück des Materialflussrechners. Seine primäre Aufgabe ist die Disposition und Durchführung aller physischen Transporte auf der Fördertechnik. Er koordiniert das zeitliche und räumlich Scheduling mehrerer Transporte für dieselbe LE, um Doppelstarts zu vermeiden. Er führt Ressourcen- & Kapazitätschecks durch und erteilt die finalen Fahrbefehle. Der Prozess kommuniziert via TCP/IP Telegrammen mit der unterliegenden SPS und liest/schreibt Zustandsdaten in die MSSQL-Datenbank. Wie HostBooking ist er ebenfalls ein Hintergrunddienst und arbeitet in festen Intervallen.

2. Projektplanung

2.1 Projektphasen & Zeitplanung

2.1.1 Vorgehensmodell

Für die Durchführung des Projekts wird ein hybrides Vorgehensmodell gewählt. Dieses kombiniert die Strukturierungsstärke des Wasserfallmodells mit der Flexibilität agiler Methoden (Kanban-Tasktracking via Azure DevOps).

Aufgrund des festen Projektzeitraums ist eine Aufteilung der Arbeit auf sequenzielle Phasen für eine terminliche Fertigstellung ideal. Jede Phase baut logisch auf den Ergebnissen der vorherigen auf. Um einer möglichen Ausweitung des Projektumfangs entgegenzuwirken, wird ein digitales Taskboard innerhalb von Azure DevOps verwendet (Siehe Abbildung 1). Dabei wurde das Projekt in 6 Phasen unterteilt:

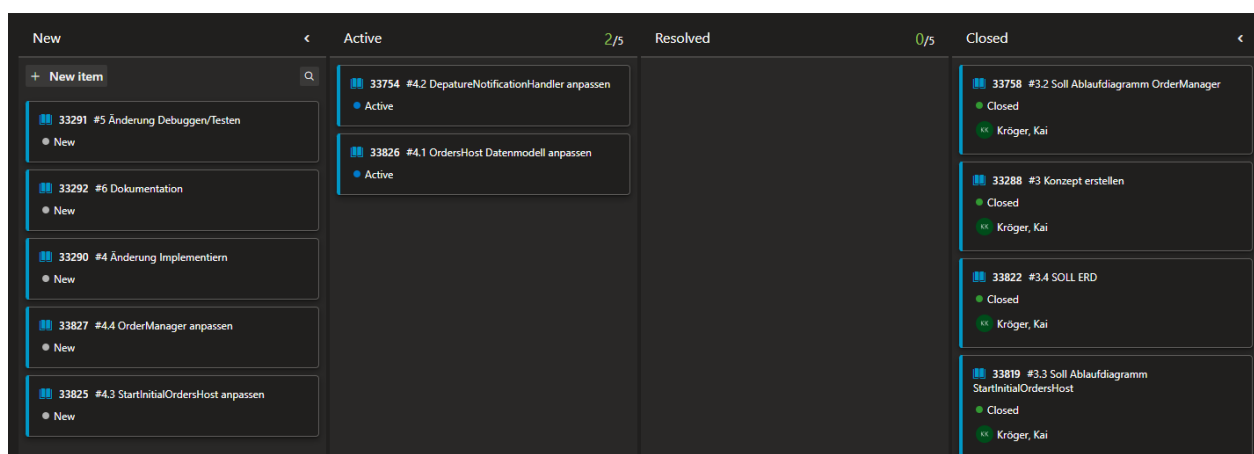


Abbildung 1 Taskboard

- **Vorbereitung/Planung** (Repository klonen, Emulation vorbereiten, Zugänge prüfen),
- **Analyse** (relevante Codestellen definiert, IST-Zustand untersucht, Fehlerquellen lokalisiert),

- **Konzeptionierung** (SOLL-Logik entwerfen, UML-Modellierung),
- **Implementierung** (Refactoring der C#-Klassen und DB-Anpassung)
- **Testung & Bugfixing** (Änderungen gegen realistische Szenarien im Emulator testen, ggf. Fehler beheben),
- **Dokumentation** (Schriftliche Ausarbeitung der IHK-Arbeit)

Für jede Phase werden Teilaufgaben in Form von Issues erstellt. Die nächste Phase wird erst nach Beendigung aller Phasen-Issues begonnen. Dies verhindert unkontrollierte parallele Arbeitsschritte und sichert die Qualität der Einzelergebnisse ab.

2.1.2 Tabellarische Zeitplanung

Die im Projektantrag grob geschätzte Phasen wurden zu Projektbeginn im Rahmen der Detailplanung präzisiert und in die nachfolgenden Arbeitsphasen überführt, um eine präzisere Zeitplanung zu ermöglichen.

Tabelle 1 Tabellarische Zeitplanung

Phase	Geplante Stunden
Vorbereitung/Planung	8
Analyse	15
Konzeptionierung	15
Implementierung	10
Testung	12
Dokumentation	20

2.2 Ressourcenplanung

2.2.1 Hardware

Entwicklungs-Workstation: Entwicklungs-PC zur Softwareerstellung und Dokumentation.

SQL-Server: Server zum Hosten der MSSQL Test-Datenbank der Förderanlage.

Application-Server: Server zum Hosten der GEBHARDT StoreWare und WCS Dienste.

Emulations-VM: Ein dedizierter Server, auf dem der „Digitale Zwilling“ der Fördertechnik läuft, um die SPS-Kommunikation (TCP/IP-Telegramme) & den Materialfluss realitätsgetreu zu simulieren.

2.2.2 Software

IDE: Microsoft Visual Studio 2022 als primäre Entwicklungsumgebung.

Datenbank: Microsoft SQL Server 2019/2022 inklusive SQL Server Management Studio (SSMS) zur Datenanalyse und Schema-Anpassung.

Frameworks: .NET Framework / .NET Core unter Verwendung von Entity Framework Core als Object-Relational Mapper (ORM).

Versionsverwaltung: Git (Azure DevOps) zur Quellcode-Verwaltung und Dokumentation der Entwicklungsschritte.

Analyse-Tools: Visual Studio Debugger, Interner Telegramm-Logger zur Analyse der TCP/IP-Kommunikation zwischen WCS und Emulation, Wcs Test Tool.

Dokumentation: Microsoft Word zum Erstellen der Dokumentation und Creately zum Erstellen der UML- & Entity-Relationship-Diagramme.

2.2.3 Personal

Prüfling (Kai Kröger): Verantwortlich für die Analyse, Konzeption, Implementierung und Qualitätssicherung des Projekts.

Fachbetreuer (Sebastian Badour & Jesper Larrson): Ansprechpartner für fachspezifische Rückfragen zur WCS-Architektur und Abnahme der Projektergebnisse.

2.3 Kostenplanung

Die Kostenplanung basiert auf dem Personalaufwand für die Projektdauer von 80 Stunden. Da die benötigte Infrastruktur (Hardware, Software, Lizenzen) bereits im Unternehmen vorhanden ist, fallen keine zusätzlichen Investitionskosten an.

Für die Kalkulation wird ein **Stundensatz von 65,00 €** angesetzt. Dieser setzt sich aus dem Brutto-Arbeitslohn, den Lohnnebenkosten sowie einem Gemeinkostenaufschlag für den Arbeitsplatz und die IT-Infrastruktur zusammen.

Kostenberechnung:

- **Gesamtkosten:** 80 Stunden × 65,00 €/Stunde = **5.200,00 €**

3. Analyse & Konzept

3.1 Ist-Analyse

Nach Analyse der Codebasis des WCS lässt sich der HostBooking Prozess als klarer Verursacher der Race Condition identifizieren. Bei einem DepartureNotification Telegramm der SPS ist der Prozess unter bestimmten Bedingungen in der Lage, initial angelegte Transportaufträge eigenmächtig zu starten. Ist eine LE gerade an einem Arbeitsplatz und wird abgemeldet (DepartureNotification wird gesendet), schaut HostBooking ob es für diese LE noch Folgeaufträge an anderen Arbeitsplätzen gibt. Ist dem der Fall, startet HostBooking eigenmächtig besagten Folgeauftrag. Dadurch entstehen folgende Probleme:

Die Race Condition: Der ConveyorDispo Prozess pollt jede Sekunde nach initialen Transportaufträgen. Findet er einen, durchläuft er Ressourcen- und Kapazitätschecks für den Transport, um sicherzustellen, dass durch den Start keine physischen Blockaden auf der Anlage entstehen. Wurden die Checks durchlaufen, startet ConveyorDispo den Transportauftrag und setzt den Status auf „InProgress“. Fällt diese LE jedoch nun unter den Spezialfall des HostBooking Prozesses, könnte er zeitgleich denselben Transportauftrag starten. Dadurch entsteht eine Transportauftragsleiche in der Datenbank, die nicht beendet werden kann. Es gibt jetzt zwei Transportaufträge für eine physische LE auf der Anlage.

Fehlende Ressourcen- & Kapazitätschecks: Durch das Übergehen des ConveyorDispo Prozesses, der Auftragsstarts unter Gewährleistung eines effizienten Materialflusses durch Ressourcen- & Kapazitätschecks an Strecke und Ziel gewährleistet, erhöht man das Risiko von physischen Hindernissen auf der Förderanlage.

Des Weiteren gibt es einen architektonischen Konflikt, bei dem sich die Handlungsräume der Prozesse ConveyorDispo und HostBooking vermischen. Diese haben mit der funktionalen Integrität der Anlage in erster Linie allerdings nichts zu tun. HostBooking ist in der Theorie eine reine Kommunikationsschnittstelle zwischen dem WMS und der Anlage. Er soll Aufträge des Host-Systems verbuchen und Telegramme der Anlage entgegennehmen sowie verarbeiten. Aktuell kann er neben der bereits benannten Race Condition auch Aufträge neu starten. Kommt eine LE an einer Vorzone an und möchte sich jedoch in die Endzone bewegen, wird eine DepartureNotification an HostBooking gesendet. Hat der Transportauftrag der LE den Status „InDestinationZone“ (die LE befindet sich in der Vorzone/kurz vor ihrem Ziel) wird sie von HostBooking auf „InProgress“ gesetzt. ConveyorDispo ist jedoch in erster Linie die ausführende Gewalt und sollte demnach, auch wenn hier keine Ressourcenchecks nötig sind, den Transportauftrags(neu)start vornehmen.

Neben den Änderungen in den Transportaufträgen werden auch die Handshake Telegramme (Departure Telegramm) dezentral als Antwort auf die DepartureNotification Telegramme vom

HostBooking an die zuständige SPS gesendet. Diese werden benötigt, um die LE physisch auf der Anlage weiterfahren zu lassen.

3.2 Soll-Konzept

Die dezentrale Start-Logik soll einheitlich im ConveyorDispo Prozess zentralisiert werden, um eine klare Rollentrennung zu gewährleisten und die zugrundeliegende Race Condition vermeiden.

Nur der ConveyorDispo Prozess darf Änderungen am Status der Transportaufträge vornehmen und diese starten als auch das entsprechende Departure Telegramm versenden.

Der DeparturNotificationHandler von HostBooking wird zum reinen Eventmelder umgeschrieben. Er aktualisiert den physischen Ort der LE und signalisiert die entsprechende Departureanfrage durch eine Flag oder einen Status an den ConveyorDispo Prozess. Durch das Hinzufügen dieser oberflächlichen Flags, muss keine tiefliegende und verwurzelte Telegramm- oder WCS-Logik angepasst werden, was immense Zeitkosten sowie auch eine größere mögliche Fehleranfälligkeit für das gesamte System bedeuten würde.

3.3 Wirtschaftlichkeitsbetrachtung

Da es sich bei der Fehlerbehebung um ein präventives Refactoring handelt, basiert die Kalkulation auf Erfahrungswerten und plausiblen Annahmen aus dem Support-Alltag von Gebhardt Förderanlagen.

Im IST-Zustand sammeln sich durch die Race Condition Auftragsleichen in der Datenbank. Diese müssen manuell von einem Supportmitarbeiter über das MSSQL Server Management Studio entfernt werden. Es muss sich per VPN aufgeschaltet und die Datensätze händisch bereinigt werden.

Kostenaufwand: ca. 320 € / Monat (basierend auf 2 Vorfällen/Woche mit jeweils einem Zeitaufwand von 30 Minuten und einem Kostensatz von 80€/h)

Zudem verursacht das Starten von initialen Transportaufträgen durch den HostBooking ohne Ressourcen- & Kapazitätschecks auf lange Sicht, gerade bei Hochlastphasen, Staus auf der Anlage was zu Performanceeinbußen führt und den Durchsatz verschlechtert.

Kostenaufwand: ca. 250 € / Monat (basierend auf 1 Vorfall/Monat mit 15 Minuten Auswirkung, kalkuliert mit 1.000 €/h Linien-Ausfallkosten)

Gesamtkalkulation (Break-Even-Point):

Einmalige Projektkosten (basierend auf der Kostenplanung): **5.200,00 €**

Monatliche Einsparung: 570 €

Break-Even-Point: $5.200 / 570 = \text{nach } 9,12 \text{ Monaten}$

Amortisationszeit: ca. 9 Monate und 3 Tage

4. Design & Architektur

4.1 Technische Entscheidungen

Wahl des Synchronisationsmechanismus: Anstatt den globalen Auftragsstatus „TransportOrderStatus“ des OrdersHost Datenmodells für die Prozessübergabe zu verwenden durch einen neuen Status wie „ReadyForDispo“, wird ein dediziertes Flag „DepartureFlag“ eingeführt.

Dies erlaubt eine atomare Übergabe zwischen dem asynchronen Nachrichtenempfang HostBooking und der zyklischen Logikverarbeitung ConveyorDispo, ohne den fachlichen Status des Auftrags zu verlieren. Es ermöglicht zudem die notwendige Fallunterscheidung zwischen Auftragsneustarts (vorrücken von Vorzone in Endzone, demnach keine Ressource- & Kapazitätschecks nötig) und Folgeauftrag (voller Startprozess).

4.2 Systemarchitektur

Zentralisierung im ConveyorDispo: Der Thread StartInitialOrders wird der zentrale Verarbeitungspunkt für initiale Transportaufträge, für die ein Departure von HostBooking angefordert wurde, analog zu den initialen Transportaufträgen, die schon jetzt von diesem Thread bearbeitet werden. Zusätzlich kümmert sich der OrderManager Thread des Prozesses um die Unterscheidung zwischen Auftragsneustarts (Status war „Initial“, ist nun durch StartInitialOrders „Pending“ und das „DepartureFlag“ ist auf true) welche das volle Programm an Ressourcen- & Kapazitätschecks erhält und Trackingaufträgen (Status ist „InDestinationZone“ und DepartureFlag auf true) welche direkt gestartet werden.

ConveyorDispo sowie HostBooking haben Zugriff auf das neu implementierte DepartureFlag innerhalb des OrdersHost Datenmodells. Anstelle eines Auftragsstarts setzt HostBooking lediglich besagtes Flag auf true, um einen Departure Request zu signalisieren. ConveyorDispo pollt nach solchen Aufträgen und entscheidet über den Start exklusiv.

5. Realisierung

5.1 Implementierung

5.1.1 Datenmodell anpassen

Das Datenmodell von OrdersHost wurde um das Feld DepartureFlag erweitert (Nach Testung wurde ein weiteres Feld (DepartureLocation) hinzugefügt, **siehe 5.3 Abweichung vom Projektplan**). In der bestehenden Datenbanktabelle OrdersHost wurden mittels Queries:

```
ALTER TABLE Wcs.OrdersHost ADD DepartureFlag bit NULL;
```

```
ALTER TABLE Wcs.OrdersHost ADD DepartureLocation NVARCHAR(50) NULL;
```

die neuen Felder hinzugefügt. Des Weiteren wurden die Felder im OrdersHost Entity nachgezogen (Siehe Abbildung 2) und in den Entity Builder von Entity Framework Core aufgenommen (Siehe Abbildung 3).

```
// Used to signal ConveyorDispo that a departure request has been made by HostBooking (DepartureNotificationHandler)
public bool? DepartureFlag { get; set; }

// Used to store the location of a DepartureNotification handled by HostBooking (DepartureNotificationHandler)
public string? DepartureLocation { get; set; }
```

Abbildung 2 Ausschnitt Entity

```
builder.Property(e => e.IdOrderWmsPOS);

builder.Property(e => e.DepartureFlag);

builder.Property(e => e.DepartureLocation);

builder.Property(e => e.Info).HasMaxLength(100);
```

Abbildung 3 Ausschnitt EF Core Entity Builder

Darüber hinaus wurde eine Query für die OrdersHost Entity geschrieben, um alle Einträge zu erhalten, die bereit für ein Departure sind (Siehe Abbildung 4).

```
/// <summary>
/// Filters by all orders that have been marked for departure by HostBooking (DepartureNotificationHandler)
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public static IQueryable<OrdersHost> IsDepartureReady(this IQueryable<OrdersHost> entity) {
    return entity.Where(o => o.DepartureLocation != null && o.DepartureFlag == true);
}
```

Abbildung 4 IsDepartureReady Query

5.1.2 Anpassung HostBooking Prozess

Dem DepartureNotificationHandler Thread von HostBooking wurden die Möglichkeiten eines eigenmächtigen Transportauftragsstarts entzogen. Anstatt das HostBooking bei einem vorrücken in der Vorzone zum eigentlichen Ziel den Auftrag selbst neu startet (Status InDestinationZone auf InProgress) wird nun nur das DepartureFlag auf true und die DepartureLocation gesetzt (Siehe Abbildung 5).

```
//ConveyorTelegrams.SendDepartureEtra(db, message.LeNo, message.Position);
//currentOrderToPlace.UpdateResources(TransportOrderStatus.InProgress, null);
//currentOrderToPlace.ForceSetStatusInProgress();
//currentOrderToPlace.Le.SetStatus(LeStatus.OnConveyor);

// Flags order for departure so that ConveyorDispo can restart it.
currentOrderToPlace.DepartureFlag = true;
currentOrderToPlace.DepartureLocation = message.Position;
db.SaveChanges();
Log.Write(LogLevel.Info, $"OrdersHost for {message.LeNo} to {message.Position} flagged for departure by DepartureNotification.");
return true;
```

Abbildung 5 Flagging Restart Orders

Bei einem Folgeauftrag wird dieser bei einer DepartureNotification nicht mehr sofort von HostBooking gestartet, sondern nur das DepartureFlag gesetzt und die DepartureLocation festgehalten. Der Prozess ConveyorDispo holt sich anhand der neuen Departure Felder die Einträge und startet sie nach eigenem Ermessen selbst (Siehe Abbildung 6).

```
//_transportOrderService.StartNextTransport(otherPicOrder.Id);
//DepartureFlag set to true to signal processing through ConveyorDispo instead of HostBooking
otherPicOrder.DepartureFlag = true;
otherPicOrder.DepartureLocation = message.Position;
db.SaveChanges();
return true;
```

Abbildung 6 Flagging Initial Orders

5.1.3 Anpassung ConveyorDispo Prozess

Zur Datenhaltung der wichtigsten Felder innerhalb der OrdersHost Tabelle für den Transportstart, wird der Record OrderListItem verwendet. Dieser wurde um die Felder DepartureFlag und DepartureLocation erweitert (Siehe Abbildung 7).

```
internal record OrderListItem(int OrdersHostId, TransportOrderStatus Status, Le Le, string Destination, int Priority, int? IdOrderWmsHead,
    DateTime Created, string HostDestination, bool? DepartureFlag, string? DepartureLocation);
```

Abbildung 7 OrderListItem

Der StartInitialOrdersHost Thread von ConveyorDispo wurde um einen weiteren Check erweitert. Obwohl das Starten von Departureaufträgen höchste Priorität hat, um Platz auf den Arbeitsplätzen zu schaffen wird innerhalb des Codes klar darauf hingewiesen, dass abgebrochene Sequenceraufträge immer zuerst gestartet werden sollen. Analogisch dazu werden die Departureaufträge nach dem Sequencercheck als zweites abgerufen und verarbeitet. Die initialen Departureaufträge werden mittels IsDepartureReady Query aus der Datenbank gezogen und in ein OrderListItem um modelliert (Siehe

Abbildung).

8).

```
// If there exists an order that has been flagged by HostBooking for departure
var departureFlaggedOrder = db.OrdersHost
    .ByLeNo(orderToBeScheduled.Le.LeNo)
    .ByStatus(TransportOrderStatus.Initial)
    .IsDepartureReady()
    .Select(o => new OrderListItem(o.Id, o.Status, o.Le, o.Destination, o.Priority,
    o.IdOrderWmsHead, o.Created, o.HostDestination, o.DepartureFlag, o.DepartureLocation))
    .ToList();
```

Abbildung 8 departureFlaggedOrder Query

Die OrderListItems werden anschließend mit der Methode ScheduleOrdersHost des transportOrderService von Initial auf Pending gesetzt. Das Ereignis wird nach Durchführung geloggt (Siehe Abbildung 9).

```
if (departureFlaggedOrder.FirstOrDefault() != null)
{
    var departureOrderToBeScheduled = departureFlaggedOrder.FirstOrDefault();
    Log.Write(LogLevel.Debug, $"[{nameof(OrdersHost)}] with Id {departureOrderToBeScheduled.OrdersHostId} will be scheduled since it has been flagged for departure");
    transportOrderService.ScheduleOrdersHost(departureOrderToBeScheduled.OrdersHostId);
    //Order has been scheduled. Do not consider orders for the same LE.
    orderList.RemoveSubsequentWithEqualLeNo(departureOrderToBeScheduled);
    workDone = true;
    continue;
}
```

Abbildung 9 Scheduling

Der **OrderManager** Thread wurde um einen weiteren Durchgang erweitert für die ausstehenden Departureaufträge im Status Pending und InDestinationZone. Mittels Query werden die Einträge aus der Datenbank ausgelesen und verarbeitet (Siehe Abbildung 10).

```
using IWcsDbContext db = _dbContextFactory.GetDbContext();

IQueryable<OrdersHost> departureNotificationOrders = db.OrdersHost
    .ByStatus(TransportOrderStatus.InDestinationZone, TransportOrderStatus.Pending)
    .IsDepartureReady();
```

Abbildung 10 Pending Departure Orders Query

Bei einem „InDestinationZone“ Auftrag wurde lediglich die vorhandene Logik aus HostBooking ausgelagert. Der Status wird auf „InProgress“ gesetzt, das Departure Telegramm gesendet und die Departure Felder zurückgesetzt (Siehe Abbildung 11).

```
foreach (OrdersHost order in departureNotificationOrders)
{
    // Restart order: Case when order has status InDestinationZone and has been flagged for
    if (order.Status == TransportOrderStatus.InDestinationZone)
    {
        order.UpdateResources(TransportOrderStatus.InProgress, null);
        order.ForceSetStatusInProgress();
        order.Le.SetStatus(LeStatus.OnConveyor);
        _destinationService.SetLeRequestDeparture(order.Le.LeNo, order.DepartureLocation);
        // TODO: Check if telegram can be sent from here
        ConveyorTelegrams.SendDepartureEtra(db, order.Le.No, order.DepartureLocation);
        order.DepartureFlag = false;
        order.DepartureLocation = null;
        db.SaveChanges();
    }
}
```

Abbildung 11 Order Restart

Die ausstehenden Aufträge durchlaufen jedoch nun, wie alle anderen auch die Ressourcen- & Kapazitätschecks, bevor sie vom ConveyorDispo mittels StartNextOrder Methode gestartet werden

(Siehe

Abbildung

12).

```
// Starting orders that are initial and were marked for departure by HostBooking (DepartureNotificationHandler)
OrderList departureOrders = new(departureNotificationOrders
    .ByStatus(TransportOrderStatus.Pending)
    .OrderBy(o => o.StartTime)
    .Select(o => new OrderListItem(o.Id, o.Status, o.Le, o.Destination, o.Priority,
        o.IdOrderWmsHead, o.Created, o.HostDestination, o.DepartureFlag, o.DepartureLocation))
    .AsNoTracking()
    .ToList());
workDone |= StartNextOrders(departureOrders);
```

Abbildung 12 Pendingauftragsstart

5.2 Qualitätssicherung & Test

5.2.2 Testumgebung

Die Testumgebung wurde folgendermaßen eingesetzt:

- **Emulate3D:** Visualisiert und berechnet das physikalische Verhalten der Förderanlage in Echtzeit
- **EMC Runner:** Ein Inhouse-Werkzeug zum Simulieren von Telegrammen auf TCP/IP Ebene zwischen SPS und WCS
- **Gebhardt Prozess Manager:** Ein Inhouse Programm in welchem die restlichen Prozesse des WCS und/oder des WMS zu testzwecken laufen und verwaltet werden (WMS Prozesse wurden beim Testen deaktiviert da das WCS Test Tool diese Rolle übernahm).
- **Visual Studio:** IDE in welcher die angepassten Prozesse ConveyorDispo & HostBooking zu testzwecken im Debugger laufen.
- **WCS Test Tool:** Ein Inhouse Werkzeug zum Simulieren des Host Systems. Mit diesem können Testszenarien durchgespielt werden, indem Befehle des WMS an das WCS simuliert werden. Zum Beispiel der Eingang eines Transportauftrags oder das Verbuchen einer LE in ein Lager.
- **MSSQL Testdatenbank:** Eine Kopie der Produktivdatenbank für die Datenhaltung der Prozesse sowie zur Überwachung der Anlage.

5.2.3 Strukturierte Testszenarien

Die Testmatrix basiert auf den offiziellen Abnahmekriterien der Fachabteilung und stellt sicher, dass alle geschäftskritischen Betriebszustände und potenziellen Fehlerquellen abgedeckt sind.

Tabelle 2 Testmatrix

Szenario	Ziel	Eingabe	SOLL Ergebnis	IST Ergebnis	Status
Normale Auslagerung	Nachweis für korrekte Standard-Prozessabwicklung nach Refactoring.	LE befindet sich im Lager. Ein neuer initialer Transportauftrag wird für diese angelegt mit Arbeitsplatz (PIC11) als Ziel.	ConveyorDispo identifiziert den neuen Initial-Auftrag über StartInitialOrdersHost und setzt ihn auf Pending. OrderManager checkt, ob der Auftrag gestartet werden kann und startet ihn ggf. (Status InProgress). LE wird korrekt ausgelagert und zu PIC11 transportiert.	Durchführung erfolgreich. LE wurde aus Lager (AKL01) ausgelagert und zum angegebenen Arbeitsplatz (PIC11) transportiert.	BESTANDEN
Mehrfach-Auftrag	Nachweis der korrekten Zustandskoordination bei Folgeaufträgen während der Fahrt.	LE befindet sich auf dem Weg zum Arbeitsplatz (PIC11) Transportauftrag befindet sich InProgress. Währenddessen kommt ein weiterer Transportauftrag im Status Initial für diese LE rein.	Der Folgetransportauftrag verbleibt im Status Initial und wird erst gestartet, wenn ein Departure Telegramm vom Ziel des ersten Transportauftrags eingeht.	Durchführung erfolgreich. Folgetransportauftrag verbleibt im Status Initial.	BESTANDEN
Sofortiger Folgeauftrag	Nachweis der prozesssicheren Übergabe via DepartureFlag zur Verhinderung von Race Conditions bei Platzabmeldung.	LE will vom Arbeitsplatz zurück in das Lager. Beim Senden der DepartureNotification wird ein neuer Transportauftrag für diese LE gesendet.	Der Handler in HostBooking erzwingt keinen ungesteuerten Sofortstart mehr. Er setzt stattdessen das DepartureFlag auf true. ConveyorDispo liest das Flag im nächsten Arbeitszyklus aus, führt alle Ressourcen- & Kapazitätsprüfungen durch und startet den Transport kontrolliert.	Durchführung nicht erfolgreich. Beim Simulieren des Departures einer LE von einem Arbeitsplatz wird der Status des vorausgehenden Transportauftrags, welcher die LE an besagten Arbeitsplatz gebracht hat, nicht wie erwartet auf Finished gesetzt. Sie verbleibt	NICHT BESTANDEN

				<p>im Status InDestinationZone. Daraus resultiert, dass dieser Transportauftrag in die Query für die Neustarts innerhalb einer Vorzone fällt. Der vorausgehende Transportauftrag wird fälschlicherweise wieder auf InProgress gesetzt. Des Weiteren wird der Folgeauftrag wie erwartet bearbeitet. HostBooking setzt die DepartureFlag sowie DepartureLocation zuverlässig & erfolgreich. Der Transportauftrag wird ebenfalls korrekt vom ConveyorDispo verarbeitet. Problem ist, man hat nun zwei Transportaufträge im Status InProgress. Stand jetzt ist unklar, ob es sich um einen Fehler im WCS Test Tool oder an der Implementierung der Änderung handelt. Beim manuellen auf Finished setzen des vorausgehenden Transportauftrags über die Datenbank ist die Durchführung des Testcases erfolgreich.</p>	
--	--	--	--	---	--

Leerbehälterwechsel	Nachweis der korrekten Daten- und Auftragsbereinigung am Arbeitsplatz.	LE wird am Arbeitsplatz leer. Über die FromWms-Schnittstelle wird eine HuChange Nachricht simuliert.	Das WCS verarbeitet die HuChange-Meldung, aktualisiert die LE-Stammdaten in der Datenbank, schließt den alten Transportauftrag ordnungsgemäß ab (Status = Finished) und bereitet den Transport des neuen Leerbehälters fehlerfrei vor.	Durchführung erfolgreich. Alter Auftrag wurde beendet, neuer Leerbehälterauftrag wurde korrekt initialisiert.	BESTANDEN
Sequencer-Einschleusung	Nachweis der gesteuerten Freigabe und Kapazitätsprüfung beim Übergang von der Vorzone in den Sequenzer/Arbeitsplatz.	LE befindet sich in der Vorzone vor PIC10 im Status InDestinationZone. Ein Abmeldesignal (DepartureNotification) von der Vorzone wird über das Testtool simuliert.	HostBooking schreibt das physische Ziel auf SEQ um und setzt DepartureFlag auf true. Der OrderManager im ConveyorDispo sucht nach Departureaufträgen. Beim fund wird besagter Auftrag neu gestartet (Status von „InDestinationZone“ auf „InProgress“). Darauf folgenden werden die Departure Felder innerhalb OrdersHost zurückgesetzt.	Durchführung erfolgreich. LE wurde kontrolliert unter der Nutzung des DepartureFlags und damit auch des OrderManager-Threads von ConveyorDispo in den Sequenzer überführt.	BESTANDEN

5.2.4 Aufgetretene Fehler

Das DepartureFlag war nicht korrekt zwischen Datenbank und Code synchronisiert. Was zu einem Nichtbestehen der Testcases „Sofortiger Folgeauftrag“ & „Sequencer Einschleusung“ führte. DepartureFlag war innerhalb des Codes (Entity Framework Kontext) als **Nullable Boolean** implementiert, während die Datenbank diese als not nullable bit abbildete. Diese Diskrepanz führte zu fehlerhaften If Abfragen innerhalb des Codes während der Runtime. Der Nullable Type wurde in der Datenbank nachgezogen da die Unterscheidung zwischen NULL und false innerhalb des Codes als unerheblich bewertet werden kann.

Des Weiteren verhielt sich das Positionsfeld der DepartureNotification anders als erwartet. Es hält die Position des nächsten Ziels und nicht wie angenommen die des aktuellen Standorts der LE. Die Position der DepartureNotification wird benötigt, um das Departure Telegramm an die SPS zu senden. Da aktuell weder die OrdersHost noch LE Tabelle diese Information beinhaltet muss sich diese auf andere Weise für den ConveyorDispo Prozess beschafft werden.

5.3 Abweichungen vom Soll-Konzept

Um ConveyorDispo zum Senden des Departure Telegramms die Position des vom HostBooking abgefangenen DepartureNotification Telegramms zugänglich zu machen, wurde sich dazu entschieden, neben dem DepartureFlag das OrdersHost Modell um ein weiteres Feld, DepartureLocation von Typ String, zu erweitern. Beim Eingehen des DepartureNotification Telegramms, setzt HostBookings DepartureNotificationHandler nun nicht mehr nur die Flag auf true, darüber hinaus wird auch die Position innerhalb des DepartureNotification Telegramms gespeichert. Diese kann ConveyorDispo per Entity Framework Core abrufen. Mit dieser Zusatzinformation kann ConveyorDispo das Departure Telegramm an die SPS senden.

6 Projektabschluss

6.1 Soll-Ist-Vergleich

6.1.2 Soll-Ist-Projektziele

Tabelle 3 Soll-Ist-Projektziele

Projektziel (Soll)	Projektziel (Ist)	Status
Exklusiver Start von OrdersHost-Aufträgen über den ConveyorDispo	Alle Starts und die dazugehörigen Departure Telegramme erfolgen exklusiv über ConveyorDispo. HostBooking hat keine Startbefugnis mehr.	ERREICHT
Beseitigung der Race Conditions	Durch das DepartureFlag werden Folgeaufträge exklusiv im ConveyorDispo gestartet. Dieser verhindert simultane Startversuche für die selbe LE und verhindert damit die Race Conditions.	ERREICHT
Verhinderung von Auftragsleichen	Die technische Race Condition (Doppelstart) wurde behoben. Im Testcase „Sofortiger Folgeauftrag“ wurde eine testumgebungsbedingte Auftragsleiche produziert, welche jedoch bei korrektem Abschluss eines vorausgehenden Auftrags nicht zustande kommt.	TEILWEISE ERREICHT
Zentralisierung der Kapazitätsprüfung	Folgeaufträge durchlaufen, wie alle anderen Initialaufträge auch die Checks innerhalb des ConveyorDispo Prozesses	ERREICHT
Strikte Trennung der Prozesszuständigkeit	HostBooking fungiert rein als Kommunikationsschnittstelle (Bei Spezialfällen, die außerhalb des Projektrahmens liegen sendet HostBooking immer noch Departure Telegramme). ConveyorBooking ist die alleinige ausführende Gewalt.	ERREICHT

6.1.3 Soll-Ist-Zeitplanung

Tabelle 4 Soll-Ist-Zeitplanung

Phase	Soll Stunden	Ist Stunden	Abweichung	Begründung
Vorbereitung/Planung	8	10	+2h	Es gab deutlich mehr Input zum Projekt und der Anlage als erwartet (Einführung in die Emulationsumgebung, Schulung zum WCS)
Analyse	15	22	+7h	Einarbeitung und Verständnis für die Threads innerhalb der Prozesse ConveyorDispo und HostBooking waren zeitintensiver als angenommen. Das Entwickeln von Ablaufdiagrammen half dem Verständnis, nahm jedoch ebenfalls Zeit in Anspruch
Konzeptionierung	15	10	-5h	Das Konzept war für die Zentralisierung der Startlogik durch die Visualisierung der Threads mittels Ablaufdiagramm schneller vollendet als geplant.
Implementierung	10	6	-4h	Durch das eingeführte DepartureFlag war das Refactoring überschaubar und deutlich weniger zeitintensiv als in der Planung angenommen.
Testung	12	12	0h	Phase konnte zeitlich nicht vollständig abgeschlossen werden. Das Beheben des Fehlers und Bestehen von Testcase „Sofortiger Folgeauftrag“ konnte innerhalb des Zeitfensters nicht erfolgen.
Dokumentation	20	20	0h	
Gesamt	80	80	0h	

6.2 Fazit & Ausblick

6.2.1 Persönliches und fachliches Fazit

Die Durchführung dieses Projekts war für mich eine fachlich äußerst wertvolle Erfahrung. Die größte Herausforderung lag darin, die Funktionen der einzelnen Threads tiefgehend zu durchdringen und ihr komplexes Zusammenspiel innerhalb der realen Anlage einzuordnen. Besonders hilfreich beim Systemverständnis sowie bei der anschließenden Konzeptionierung und Implementierung waren die entwickelten Ablaufdiagramme. Mittels dieser Visualisierungen ließen sich die logischen Schwachstellen schnell lokalisieren und ein Lösungsweg erarbeiten. Das Projekt wurde innerhalb des vorgegebenen Zeitrahmens von 80 Stunden erfolgreich und funktionsfähig abgeschlossen. Die Rollenverteilung zwischen ConveyorDispo und HostBooking ist nun deutlich einheitlicher, stabiler und prozesssicherer aufgebaut.

6.2.2 Ausblick

Die Abnahme des Projekts zeigte, dass das Positionsfeld der DepartureNotification doch über das LastWhere Feld der LE oder Source Feld von OrdersHost Beziehen lassen. Für reine Funktionalität könnte man demnach auf das DepartureLocation Feld verzichten. Um jedoch eine robuste Datenübergabe zwischen DepartureNotification und ConveyorDispo zu gewährleisten, wird das DepartureLocation Feld beibehalten.

In den Spezialfällen innerhalb des DepartureNotificationHandlers werden noch immer Departure Telegramme von diesem an die SPS versendet. Diese gilt es ebenfalls in den ConveyorDispo Prozess zu verschieben, um die klare Rollentrennung zu vereinheitlichen.

Durch die Verlagerung der Departure Telegramme vom ereignisgesteuerten HostBooking zum zyklisch pollenden ConveyorDispo entsteht ein potenzieller logischer Wettlauf. Wenn HostBooking zu lange braucht, um einen Folgeauftrag mit der DepartureFlag zu markieren, könnte der ConveyorDispo ihm zuvorkommen und den unmarkierten Folgeauftrag ganz normal Verarbeiten. Problem ist, bei einer normalen Verarbeitung wird die Logik für ein Departure nicht ausgeführt. Daraus resultiert, dass der Transportauftrag gestartet wird, die physische LE sich jedoch nicht vom Arbeitsplatz wegbewegt. Für die Behandlung des Szenarios würde sich ein zusätzlicher Status „DepartureReady“ anbieten, welcher für Folgeaufträge anstelle von Initial gesetzt wird.

Anhang

Kundendokumentation

Quellcode

Abnahmeprotokoll

Glossar

Prozess & Ablaufdiagramme

Quellenverzeichnis

- Gebhardt WCS Schulung
- Interne StoreWare Wiki
- <https://gebhardt-group.com/>

TODO // Kundendokumentation, Abnahmeprotokoll, Quellcode, Glossar, Prozess & Ablaufdiagramm, ~~Fazit & Ausblick, Soll-Ist Projektziele~~, Doku probelesen

Jetzt, 3 testcases, Fazit & Ausblick, Soll-Ist Projektziele