# Desktop-Fakturierungsanwendung

## Schlanke, lokal betriebene Fakturierung für Kleinstunternehmen

**Abschlusspräsentation · Software Engineering 1 (SoSe 2026)**

**Team 1**

29.06.2026 · Prof. Dr. Gerd Marmitt

# Agenda

1. **Team & Aufgabenverteilung**
2. **Kontext & Ziel** des Projekts
3. **Architektur** im Überblick
4. **Live-Demo** des Programms *(≈ 10 Min)*
5. **Entwicklungsprojekt** *(≈ 5 Min)*
    - Ergebnisse der Modultestpläne
    - Wo wurde KI eingesetzt – und wie gut?
    - Was würden wir nächstes Mal anders machen?
6. **Fragen** *(5 Min)*

# Team & Aufgabenverteilung

12 Personen, vier fachliche Komponenten (V-Modell, Einzelplatzanwendung):

| Gruppe | Komponente | Leitung | Mitglieder |
|---|---|---|---|
| **A** | Prozess / Dokumentenzyklus | Lucas Strubel | Luca Kaiser |
| **B** | Produktverwaltung | Meron Berhane | Jan-Micah SchulzeAmeling · Jessica Volz |
| **C** | Kundenverwaltung | Mahsuna Ahadyar | Kübra Kilic · Mara Weidmann |
| **D** | Programmoberfläche | Mirkan Güngör | Moritz König · Mohammed Bouhki |

Querschnitt (`gemeinsam`): EreignisBus, JSON-Persistenz, CSV-Hilfe – von allen Gruppen genutzt.

# Kontext & Ziel

**Problem:** SaaS-Fakturierung kostet laufend Lizenzgebühren; Open-Source-Alternativen (z. B. *Fakturama*) sind im Betrieb anspruchsvoll.

**Unsere Lösung:** schlanke Desktop-Anwendung mit voller Datensouveränität – ohne Cloud.

| Ziel | Inhalt |
|---|---|
| **PZ-01** | Digitale Verwaltung von Kunden- & Produktstammdaten (CRUD) |
| **PZ-02** | Vollständiger Dokumentenzyklus: Angebot → AB → Lieferschein → Rechnung |
| **PZ-03** | Bedienbar ohne Vorkenntnisse (Rechnung in < 10 Min) |
| **PZ-04** | 100 % lokale, DSGVO-konforme Datenhaltung |

*Nichtziele:* Mehrbenutzer/Netzwerk, vollständige Buchhaltung, E-Rechnung (ZUGFeRD/XRechnung).

# Architektur im Überblick

**Vier Fachmodule + gemeinsame Infrastruktur**, manuelle Dependency Injection in `Main.java` (kein Framework).

- **Repository-Interfaces** je Domäne mit JSON-Implementierungen; atomare Schreibvorgänge (`JsonPersistenz.schreibeAtomar`)
- **EreignisBus** (Observer): Services melden Datenänderungen, GUI-Panels aktualisieren sich selbst
- **Snapshot-Prinzip** in Belegen + **Löschsperren** (referenzielle Integrität)

### Technologie-Stack

`Java 21` · `Swing + FlatLaf 3.6` · `Maven` (Fat-JAR via shade) · `Jackson 2.17` (JSON) · `Apache PDFBox 3.0` (PDF) · `JUnit 5.10`

# Live-Demo

## Das entwickelte Programm

# Demo 1 — Stammdaten

### Kundenverwaltung (Gruppe C)

- Anlegen mit eindeutiger Kundennummer (`K-000017`), Pflichtfeld-Validierung *(BA-01)*
- Suchen & sortierte Liste, Ändern *(BA-02, BA-04)*
- **Löschsperre**: Kunde mit verknüpften Dokumenten wird nicht gelöscht *(BA-03, GR-04)*

### Produktverwaltung (Gruppe B)

- Anlegen mit Nettopreis & zulässigem Steuersatz, Nummernvergabe `P-000042` *(BA-05)*
- Ändern, Suche, **Löschsperre** bei referenzierten Produkten *(BA-06–BA-08)*

# Demo 2 — Dokumentenzyklus (Gruppe A)

**Angebot → Auftragsbestätigung → Lieferschein → Rechnung**

- Folgebeleg übernimmt Kunde, Positionen & Mengen und speichert eine **Rückreferenz** *(GR-05)*
- **Lückenlose** Belegnummern `R-2026-000124` (GoBD) *(GR-01)*
- Automatische **Steuerberechnung** netto / Steuer / brutto, Scale 2 *(GR-03)*
- Standard-**Zahlungsziel** +14 Tage, falls nicht angegeben *(GR-06)*
- Rechnung mit allen Pflichtangaben gem. **§ 14 UStG** *(BA-12)*

# Demo 3 — Wizard, Export & Storno

- **Geführte Rechnungserstellung** (Wizard): Kunde → Positionen → Bestätigen → Zusammenfassung → Speichern *(BA-13)*
- **PDF-Export** der Belege (PDFBox) ins lokale Dateisystem *(IF-01)*
- **CSV-Vollexport** von Stamm- **und** Bewegungsdaten – offenes Format *(Q-08)*
- **Stornierung** einer offenen Rechnung mit Datum & Benutzer; versendete Belege sind **unveränderlich** *(BA-14, GR-02)*

# Oberfläche: Beitrag Gruppe D

- Die Demo läuft über die von Gruppe D umgesetzte **Swing-Oberfläche**
- Verantwortet: Navigation, Dialogführung, Pflichtfeldhinweise und Status-/Aktionsanzeige
- Fachlogik bleibt in den Komponenten A–C und wird über Services aufgerufen
- Beispiel Traceability: `BA-13` → `F-09…F-13` → `TC-01…TC-08` → `RechnungsWizardController`
- Nachweis: `OberflaechenControllerTest` mit **15 grünen JUnit-Tests**

# Das Entwicklungsprojekt

## Modultests · KI-Einsatz · Lessons Learned

# Ergebnisse der Modultestpläne

**71 deterministische JUnit-5-Testfälle – alle grün.** Nachbarkomponenten durch Stubs/Mocks ersetzt.

| Komponente | Testfälle |
|---|---|
| A — Dokumentenzyklus (Zyklus, Persistenz, PDF, CSV) | 18 |
| B — Produktverwaltung | 14 |
| C — Kundenverwaltung | 14 |
| D — Programmoberfläche (Controller) | 15 |
| Gemeinsame Infrastruktur (EreignisBus, JSON) | 6 |
| Performance/Last (Q-02/03/04/08) | 4 |
| **Summe** | **71** |

Performance-Schranken eingehalten: Start ≤ 5 s, Suche ≤ 1 s, PDF ≤ 2 s, Export ≤ 30 s (5.000 Kunden/Produkte).

# Traceability

**Lückenlose Kette: Anforderung → Codebeleg → Testfall.**

- Spezifikation der Testfälle: `Modultestplan.md` (Vorbedingung / Eingabe / erwartetes Ergebnis)
- Nachweismatrix: `Anforderungsabgleich.md` — BA-01…14, GR-01…06, Q-01…09, AC-01…11
- Jede Testfall-ID ist auch im Code per `@DisplayName` auffindbar (`TC-…`, `INF-…`, `Q-…`) → direkt grep-bar

> *Alle durch Code/Tests belegbaren Anforderungen sind ✅; offen bleiben nur organisatorische Usability-Tests (Q-05/AC-11).*

# Wo wurde KI eingesetzt – und wie gut?

> *Entwurf — vom Team mit eigenen Erfahrungswerten zu ergänzen.*

## Eingesetzt für

- **Dokumentation:** Entwürfe für Lasten-/Pflichtenhefte, Modultestplan, Traceability-Matrix
- **Code:** Boilerplate der Repository-/Service-Schicht, JSON-Persistenz, CSV-Export, JUnit-Testgerüste
- **Diagramme & Tooling:** PlantUML-Quellen, Maven-Konfiguration, pandoc-/PDF-Workflow

## Bewertung

- 👍 Hohes Tempo bei wiederkehrendem Code, Tests & Doku; konsistente deutsche Fachsprache; schnelle Iteration nach Feedback
- 👎 Fachlogik (§ 14 UStG, GoBD, Steuer-/Snapshot-Regeln) und Gruppen-Schnittstellen mussten **manuell geprüft & nachgeschärft** werden; Vorschläge teils zu generisch

# Was würden wir nächstes Mal anders machen?

*Entwurf — vom Team zu ergänzen.*

- **Früher integrieren:** modulübergreifende Tests nicht erst spät im V-Modell
- **Schnittstellen-Verträge** zwischen Gruppen A–D verbindlich zu Projektbeginn festlegen
- **Kleinere, häufigere Commits** + konsequentere Pull-Request-Reviews
- **KI-Output gezielter prüfen** statt übernehmen – besonders bei Fachlogik
- Aufgaben-/Lastverteilung über die Git-History früh sichtbar machen

# Zusammenfassung

- Vollständiger **Dokumentenzyklus** + **Stammdatenverwaltung** lauffähig umgesetzt
- **Lokale, DSGVO-konforme** Einzelplatzanwendung (Java 21, Swing/FlatLaf)
- **71 grüne Modultests** mit vollständiger **Traceability** zu den Anforderungen
- Alle Performance-Anforderungen erfüllt

# Vielen Dank!

## Fragen?

**Team 1 · Desktop-Fakturierungsanwendung · SE1 SoSe 2026**