Product Requirements Document (PRD)

Java-Programm (Vibe Coding Ansatz)

---

1. Dokument-Metadaten

- Produktname:
- Projektname (Repository):
- Version:
- Status: (Draft / Review / Approved)
- Product Owner:
- Technischer Lead:
- Letzte Aktualisierung:
- Verlinkte Artefakte (Repo, Tickets, Architekturdiagramme):

---

2. Vision & Zielbild

2.1 Produktvision

CodeVibrator ist ein Java-Swing-Programm, das KI-gestütztes Programmieren mittelgroßer Projekte mit ChatGPT vereinfacht, indem ein erprobter „Zip → Prompt → Zip zurück"-Workflow teilautomatisiert, standardisiert und sicherer gemacht wird.

2.2 Zielzustand

- Nutzer wählen projektweit oder verzeichnisspezifisch (vererbend) aus, **welche Dateien** in ein Zip-Paket für ChatGPT aufgenommen werden.
- Nutzer definieren mehrere **Zip-Profile** (z. B. „MVP", „Refactor", „Docs"), die jeweils eigene Auswahl- und Filterregeln haben.
- Für jeden Durchgang wird ein **Prompt** erfasst; **Prompt-Historie** unterstützt Wiederverwendung.
- Vor dem Senden können mehrere **Prompt-AddOns** (wiederverwendbare Prompt-Bausteine) aus einem editierbaren Vorrat aktiviert werden.
- Beim Zurückspielen der Antwort steuern **Rückimport-Filter** (ebenfalls vererbend pro Verzeichnis), welche Dateien überschrieben/nicht überschrieben werden.
- Optional kann vor jedem Durchgang ein **Git-Commit** erstellt werden; Commit-Message wird aus dem Prompt abgeleitet.

2.3 Abgrenzung

- Kein Ersatz für IDE/Build-System; Fokus auf Orchestrierung des Datei-/Prompt-/Import-Workflows.

- Keine direkte Kontrolle darüber, wie ChatGPT intern Code generiert; das Tool unterstützt Packaging, Prompting, und sichere Anwendung der Ergebnisse.

---

3. Zielgruppe & Stakeholder

4.1 Primäre Nutzer

- Entwickler:innen, die ChatGPT für mittelgroße Codebases nutzen und dafür wiederholt Projekt-Snapshots (Zip) übertragen.
- Technisches Niveau: fortgeschritten (Git, Projektstrukturen, Patterns, Build/Run).

4.2 Sekundäre Nutzer (optional)

- Teams, die einen standardisierten Workflow für KI-Änderungen etablieren wollen.

4.3 Interne Stakeholder

- Entwicklung
- Betrieb (falls Distribution/Updates relevant)
- Security/Compliance (Datei-Exfiltration, Secrets)

---

4. Use Cases

Format für jeden Use Case:

UC-XX: Titel

- Actor:
- Vorbedingung:
- Trigger:
- Hauptablauf:
- Alternativabläufe:
- Ergebnis:
- Edge Cases:

UC-01: Zip-Paket nach Profil erstellen

- Actor: User
- Vorbedingung: Projektverzeichnis ausgewählt; mindestens ein Zip-Profil vorhanden
- Trigger: „Build Zip“/„Send“
- Hauptablauf: Regeln (Patterns/Dateien) vererbend auslesen → Ergebnisliste berechnen → Zip erzeugen (Timestamp im Namen)
- Ergebnis: Zip liegt lokal vor und ist bereit zum Versand

UC-02: Prompt erstellen und um AddOns erweitern

- Actor: User
- Hauptablauf: Prompt eingeben → Prompt-Historie optional nutzen → mehrere AddOns aktivieren → finalen Prompt zusammensetzen

UC-03: Antwort-Zip anwenden mit Rückimport-Filtern

- Actor: User
- Hauptablauf: Antwort-Zip auswählen → Rückimport-Regeln pro Ebene anwenden → Dateien überschreiben/nicht überschreiben → Ergebnisreport

UC-04: Optionaler Git-Commit vor Durchgang

- Actor: User
- Hauptablauf: Option aktivieren → Commit-Message aus Prompt generieren → `git commit` ausführen (ggf. `git add` nach definierter Regel)

---

5. Functional Requirements (FR)

Alle Anforderungen nummerieren und testbar formulieren.

6.1 Projekt & Verzeichnisbaum

FR-01: Das Programm muss ein Projekt-Root-Verzeichnis auswählen und als Baum anzeigen. - der Tree-Ansicht werden **nur Verzeichnisse** angezeigt (keine Dateien).

FR-01a: Die Datei-Auswahl-UI muss als dreigeteiltes SplitPane umgesetzt werden: - Links: Verzeichnisbaum (nur Verzeichnisse, optional mit Zusatzinfos) - Rechts oben: Kontrollfeld „Selektoren für das aktuell ausgewählte Verzeichnis (vorbefüllt mit Ebenenwerten) - Rechts unten: Dateiliste aktuell ausgewählten Verzeichnisses

FR-01b: In der Dateiliste muss pro Datei gut erkennbar angezeigt werden, wie sie von (auflösbaren) Selektoren betroffen ist (z. B. matcht/aktiv → im Zip; matcht/inaktiv → nicht im Z über geerbten Zustand).

FR-01c: Der Nutzer muss Verzeichnisse pro „**Exclude from Zip**“ markieren können (aus dem Zip komplett ausschließen), vererbend nach unten. - Akzeptanz: Excluded-Verzeichnisse werden im markiert; Dateiliste/Selektoren spiegeln „excluded“ Zustand.

FR-02: Der Nutzer muss pro Verzeichnisebene Regeln definieren können, die vererbend auf Verzeichnisse wirken (Force/Inherit, siehe 6.2). Der Nutzer muss pro Verzeichnisebene Regeln definieren können, die vererbend auf Sub-Verzeichnisse wirken (Force/Inherit, siehe 6.2).

6.2 Auswahlregeln für Zip-Input für Zip-Input (Upload)

Begriffe

- **Selektor**: Dateiname-Pattern, das nur auf die **aktuelle Verzeichnisebene** angewendet wird (keine Pfad-/Unterverzeichnis-Wildcards).
- **Aktiv**: Selektor ist eingeschlossen (checked) oder ausgeschlossen (unchecked).
- **Force**: Aktiv-Zustand wird in dieser Ebene festgelegt und gilt ab hier nach unten.
- **Inherit** (not-force): Aktiv-Zustand wird von der Parent-Ebene übernommen.

Pattern-Syntax

- Pattern sind einfache Dateiname-Globs, z. B. `*.jpg;*.jpeg;*.java`.
- Es gibt **keine** Pfadpattern wie `src/**/*.java` und **kein** vollständiges Regex.
- Separator für mehrere Pattern in einem Feld: `;`.

Vererbungs- und „Ausmultiplizierungs“-Regel

- Je Verzeichnisebene existiert ein **Textfeld** für Pattern-Eingabe (z. B. `*.java;*.md`).
- Alle im Textfeld genannten Selektoren sind in dieser Ebene **immer Force** und setzen den Aktiv-Zustand für diese Ebene.
- Selektoren aus Parent-Ebenen werden in der Child-Ebene **ausmultipliziert** und erscheinen als einzeln auswählbare Zeilen.
- Für jede ausmultiplizierte Zeile gibt es zwei Checkboxen:
- **Force**
- **Aktiv**
- Default in der Child-Ebene:
- Aktiv-Zustand wird vom Parent übernommen.
- Force ist standardmäßig **aus** (Inherit), bis der Nutzer Force aktiviert.
- Wenn Force in einer Child-Zeile aktiviert ist, kann der Nutzer Aktiv explizit setzen (aktiv oder nicht aktiv); andernfalls gilt der geerbte Wert.
- Selektoren, die in einer Ebene neu über das Textfeld hinzugefügt werden, werden ab dieser Ebene nach unten weitergereicht und in tieferen Ebenen als ausmultiplizierte Zeilen angezeigt.

Zip-Input-Entscheidungsregel

- Der Zip-Input wird **ausschließlich** über **Aktiv-Selektoren** bestimmt, **außer** ein Verzeichnis ist als **Exclude from Zip** markiert.
- Wenn ein Verzeichnis (oder ein Parent) „Exclude from Zip“ ist, werden **keine** Dateien aus diesem Verzeichnis und seinen Subverzeichnissen ins Zip aufgenommen.
- Andernfalls: Eine Datei wird in das Zip aufgenommen, wenn sie mindestens einen aktiven Selektor matcht (OR-Logik), sofern keine widersprüchliche Konfiguration vorliegt.

Konflikterkennung bei Selektoren

- Wenn eine Datei mehrere Selektoren matcht, deren aufgelöste Aktiv-Zustände (nach Force/Inherit) widersprüchlich sind, gilt dies als **Konfliktfall**.
- Konfliktfälle müssen in der UI automatisch mit einem **Warnsymbol** markiert werden (sowohl im Selektoren-Kontrollfeld als auch in der Dateiliste).
- Der Nutzer kann die Warnung akzeptieren (keine automatische Blockade).

- Falls der Nutzer die Warnung hinnimmt, gilt für die finale Zip-Entscheidung die **OR-Logik** (mindestens ein aktiver Selektor → Datei wird aufgenommen).

FRs (präzisiert)

FR-03c: Das System muss widersprüchliche Selektoren-Konfigurationen erkennen und visuell kennzeichnen. - Akzeptanz: Konflikt wird ohne manuelle Aktion angezeigt; keine stille Fehlentscheidung.

FR-03d: Bei bestätigtem/ignoriertem Konflikt muss die Zip-Entscheidung per OR-Logik erfolgen. Akzeptanz: Verhalten ist deterministisch und dokumentiert.

FR-03: Der Nutzer muss pro Verzeichnisebene Selektoren als einfache Dateiname-Globs definiere können (z. B. `*.jpg;*.java`). - Akzeptanz: Keine Pfad-/Unterverzeichnis-Wildcards; Separator `;`.

FR-03a: Das System muss Selektoren aus Parent-Ebenen in jeder Child-Ebene als einzelne auswählbare Zeilen darstellen (Ausmultiplizierung). - Akzeptanz: Jede Zeile hat Force + Aktiv; Def Aktiv wird geerbt.

FR-03b: Das System muss pro Selektor den Vererbungsmechanismus Force/Inherit und den Ak Zustand korrekt auflösen. - Akzeptanz: Ohne Force gilt Parent-Aktiv; mit Force gilt Child-Aktiv.

FR-04: Die Regeln (Textfeld-Selektoren und Overrides für ausmultiplizierte Selektoren) müssen pr betroffenem Verzeichnis in `.code.vibrator` als JSON persistiert werden.

FR-05: Das System muss beliebig viele Zip-Profile verwalten können.

FR-06: Profile teilen sich pro Verzeichnis eine `.code.vibrator` Datei (oder keine Datei, wenn in dieser Ebene keine Abweichungen konfiguriert sind). - Akzeptanz: Fachliche Trennung per JSON Struktur; Dateiname immer `.code.vibrator`.

FR-07: Das System muss aus den aufgelösten Regeln ein Zip-Archiv erzeugen können, dess Dateiname einen Zeitstempel enthält.

6.3 Prompting

FR-08: Zu jedem Durchgang muss der Nutzer einen Prompt erfassen können.

FR-09: Das System muss eine Prompt-Historie bereitstellen. - Persistenz: projektlokal.

FR-10: Das System muss einen editierbaren Vorrat an Prompt-AddOns bereitstellen; mehrere AddOns können gleichzeitig aktiv sein. - Persistenz: projektlokal.

FR-10a: Das System muss einen sinnvollen Satz Default-AddOns bereitstellen, der im User-Hor verwaltet wird. - Akzeptanz: Beim Anlegen/Öffnen eines Projekts können Defaults als Startbestan übernommen werden.

FR-11: Das System muss den finalen Prompt aus Basis-Prompt + aktiven AddOns zusammens (inkl. Vorschau).

6.4 Rückimport / Überschreibschutz

Readonly-Regeln

- Regeln sind pro Verzeichnisebene definierbar und vererbend.
- Es gibt:
- **Readonly-Dateien**: dürfen nicht durch Inhalte aus dem Antwort-Zip ersetzt werden.
- **Readonly-Verzeichnisse**: in diesen Verzeichnissen dürfen keinerlei Dateien ersetzt werden.

Konfiguration: - Readonly-Dateien werden per einfacher Dateiname-Pattern je Ebene definiert (analo Selektoren, z. `*.properties;*.pem`). - Readonly-Verzeichnisse werden über den Tree-Knoten wählbar gemacht („Dieses Verzeichnis readonly“), vererbend nach unten.

FR-12: Das System muss pro Verzeichnisebene vererbbare Readonly-Regeln für Dateien und Unterverzeichnisse definieren können. - Akzeptanz: Readonly-Directory hat Vorrang (blockiert alle Dateien darunter).

FR-12a: Beim Anwenden eines Antwort-Zips muss das System Readonly-Regeln strikt einhalten. Akzeptanz: Geblockte Dateien werden nicht überschrieben; Report listet geblockte Einträge.

FR-12b: Das System darf niemals automatisch Dateien im Zielverzeichnis löschen. - Akzeptan Delete-Operationen sind ausgeschlossen.

FR-12c: Das System soll die Readonly-Information zusätzlich in den finalen Prompt aufnehmen Format: Als JSON-Block (für maschinenlesbare Nutzung durch ChatGPT).

Refactoring-Policy

- Default: ChatGPT soll keine Refaktorierung vornehmen; Refactoring erfolgt manuell durch den Nutzer.
- Ausnahme: Nur wenn der Nutzer explizit „Refactor“ wünscht (Details TBD).

FR-13: Das System muss die Antwort-Zip-Datei in das Projektverzeichnis anwenden können, oh Löschungen, unter Beachtung der Readonly-Regeln. - Akzeptanz: Report über angewandte/ übersprungene Dateien.

6.5 Git-Integration

FR-14: Das System muss optional vor jedem Durchgang einen Git-Commit erstellen können.

FR-14a: Wenn aktiviert, muss der Commit immer ein `git add -A` (alles) ausführen und anschließend committen. - Akzeptanz: Add umfasst alle Änderungen im Repo.

FR-15: Das System muss aus dem Prompt automatisch eine Commit-Message ableiten; Nutzer final editieren.

6 . Non-Functional Requirements (NFR)

Non-Functional Requirements (NFR)

7 . 1 Performance

- Maximale Antwortzeit:
- Erwartete gleichzeitige Nutzer:

7 . 2 Skalierbarkeit

- Horizontale / Vertikale Skalierung:

7 . 3 Sicherheit

- Authentifizierung:
- Autorisierung:
- Verschlüsselung:

7 . 4 Verfügbarkeit

- Zielverfügbarkeit (z. B. 99,9 %):

7 . 5 Wartbarkeit

- Logging
- Monitoring
- Testabdeckung (% Zielwert)

7 . 6 Portabilität

- Unterstützte Betriebssysteme:
- Java-Version:

---

7 . Systemarchitektur (High-Level)

8 . Systemarchitektur (High-Level)

Fokus: Lokale Java-Swing-Desktop-Anwendung ohne Server- oder Datenbank-Komponente.

8 . 1 Architekturtyp

- Desktop-Anwendung (Java Swing)
- Modularer Aufbau (UI / Domain-Logik / Persistenz / Git-Integration)

8.2 Kernmodule

- UI (Swing, Tree, SplitPane, Dialoge)
- Selektor-Engine (Vererbung, Force/Inherit, Konflikterkennung)
- Zip-Engine (Export / Import, Konflikt-Report)
- Prompt-Engine (History, AddOns, JSON-Augmentation)
- Git-Integration (add -A + commit)
- JSON-Persistenz (`.code.vibrator`)

Import-Engine (technische Struktur, stateless)

Die Import-Engine ist **stateless**. Alle Schritte erhalten ihre Inputs als Parameter und liefern Outputs als Ergebnisobjekte (immutable DTOs). UI hält keinen Import-Zustand, sondern rendert Ergebnisobjekte.

Zusätzliche Grundlage: **Textdatei-Erkennung erfolgt ausschließlich über Konfiguration in den Home-Defaults** (keine MIME-Erkennung, keine Inhalts-Heuristiken). - Zwei Konfigurationslisten (semikolon-separiert): 1. `textFileExtensions` → Dateiendungen **mit führendem Punkt** (z. B. `.java;.xml;.md`). Vergleich case-insensitive. `textFileExactNames` → Exakte Dateinamen ohne Pfad (z. `Makefile;Dockerfile;gradlew`). Vergleich case-sensitive oder case-insensitive je nach Dateisystem (Implementierungsdetail festzulegen). - Eine Datei gilt als Textdatei, wenn: - ihr exakt in `textFileExactNames` enthalten ist, ODER - ihre En `textFileExtensions` enthalten ist. - Alle anderen Dateien gelten als Nicht-Textdateien. - Sind die Listen nicht vorha leer, werden sie beim ersten Start mit einem sinnvollen Standardset für Softwareentwicklung initialisiert.

Komponenten: -**ResultZipReader** - Öffnet Result-Zip - Validiert Struktur und Inhalte - Liefert `ZipIndex` (Liste von Entries mit Pfad, Typ, Metadaten) - Erkennt und verwirft ungültige In (Fehler/Exception)

- **WorkingDirSnapshotBuilder**
- Erfasst aktuellen Zustand des Working Directory
- Liefert `WorkingSnapshot`
- **ChangeDetector**
- Ermittelt je Pfad den Change-Typ (NEW/MODIFIED/IDENTICAL/BLOCKED/IGNORED etc.)
- Textdateien: Normalisierter Zeichenvergleich (Whitespace-Normalisierung)
- Nicht-Textdateien: Hashvergleich
- **ConflictValidator**
- Prüft harte Abbruchregeln (lokale Änderungen während Workflow, Readonly-Verletzung, Strukturfehler etc.)
- Liefert `ValidationResult` (ok/fatal + Details)
- **ImportExecutor**

- Führt nur bei `ValidationResult.ok` Schreiboperationen aus
- Schreibt direkt ins Working Directory
- Keine Shadow-Verzeichnisse

- Keine Löschungen

- **ResultLogger**

- Erzeugt zyklisches Result-Log (Warnungen, geblockte Dateien, Ignored Executables, Case-Konflikte, Statistiken)
- Speichert im Tages-Arbeitsverzeichnis

8.3 Technologiestack

- Java-Version: (festzulegen, z. B. LTS)
- UI: Swing
- JSON: (z. B. Jackson oder Gson)
- Git-Integration: CLI-Aufruf oder Library (noch festzulegen)

---

8. Datenmodell

9.1 Persistenzkonzept

- Persistiert wird ausschließlich in Dateien mit dem Namen `.code.vibrator`.
- Pro Verzeichnis kann es `0` **oder** `1` Datei geben (wenn keine Abweichungen/Regeln in dieser Ebene existieren, kann die Datei fehlen).
- Alle **Zip-Profile** teilen sich eine `.code.vibrator` Datei pro Verzeichnis.
- Projektweite Daten (Prompt-Historie, AddOn-Bibliothek, Defaults-Referenz) werden **projektlokal** gespeichert; dafür wird eine `.code.vibrator` im **Projekt-Root** verwendet (Project-Scope-Abschnitt).

9.2 JSON-Grundgerüst (Vorschlag)

9.2.1 Gemeinsame Top-Level-Struktur (in jeder `.code.vibrator`)

```
{
  "schemaVersion": "1.0",
  "scope": "dir",
  "dir": {
    "path": ".",
    "profiles": {
      "<profileId>": {
        "zipSelectors": {
          "excludeDir": false,
          "excludeChildDirs": {
            "node_modules": true,
```

```
            "target": true
          },
          "selectorsText": "*.java;*.md",
          "expandedOverrides": {
            "*.java": { "force": false, "active": true },
            "*.md": { "force": true, "active": false }
          }
        },
        "readonly": {
          "readonlyDir": false,
          "readonlyFilesText": "*.pem;*.key",
          "expandedFileOverrides": {
            "*.pem": { "force": false, "active": true }
          },
          "readonlyChildDirs": {
            "secrets": true,
            "build": false
          }
        }
      }
    }
  }
}
```

Bedeutung der Felder (ku `schemaVersion`: Versionierung für Migrati `scope`: `dir` für Verzeichnisebene; `project` nur im Root zusätzlich (siehe `dir.path`: - relativ zum jeweiligen Verzeichnis (typisc `"."`), kann für Debug/Tools genutzt `profiles`: Map `profileId` → Profil-spezifische Regeln für dieses Verz `zipSelectors.excludeDir`: Wen `true`, ist dieses Verzeichnis (und per Vererbung darunter) vollständig **vom Zip ausgeschlossen** - `zipSelectors.excludeChildDirs`: explizite Flags für direkte Subverzeichnisse, die im Tree als „Exclude from Zip" markiert w `zipSelectors.selectorsText`: in dieser Ebene neu gesetzte (immer Force) Selektoren als Textfeld (Semikolon-ge `zipSelectors.expandedOverrides`: Overrides für aus Parent „ausmultiplizierte" Selektoren (2 Chec `force`, `active`). - `readonly.readonlyDir`: Wenn `true`, ist dieses Verzeichnis (und per Vererbung darunter) vollständig readonly. `readonly.readonlyFilesText`: in dieser Ebene neu definierte readonly-Dateipatterns (analog `selectorsText`). - `readonly.expandedFileOverrides`: Overrides für readonly-Dateipatterns, die aus Parent-Ebenen star `readonly.readonlyChildDirs`: explizite Flags für direkte Subverzeichnisse, die im Tree als readonly markiert werden.

9.2.2 Projekt-Root `.code.vibrator` (zusätzlich Project-Scope-Daten)

Im Projekt-Root v `scope` a `project` gesetzt und `project`-Block ergänzt (o `scope` als `mixed`, falls beides in einer Datei zusammengeführt werden soll).

```
{
  "schemaVersion": "1.0",
  "scope": "project",
  "project": {
    "profiles": {
```

```
      "default": { "name": "Default", "createdAt": "2026-02-24T12:00:00Z" },
      "docs": { "name": "Docs" }
    },
    "activeProfileId": "default",
    "promptHistory": [
      {
        "id": "ph_001",
        "createdAt": "2026-02-24T12:34:56Z",
        "basePrompt": "Bitte implementiere ...",
        "activeAddOnIds": ["ao_no_refactor", "ao_tests"],
        "finalPrompt": "... zusammengesetzt ..."
      }
    ],
    "promptAddOns": {
      "ao_no_refactor": {
        "title": "Keine Refaktorierung",
        "content": "Bitte keine Refaktorierung durchführen ...",
        "enabledByDefault": true
      },
      "ao_tests": {
        "title": "Tests ergänzen",
        "content": "Bitte ergänze/aktualisiere Unit-Tests ...",
        "enabledByDefault": false
      }
    },
    "defaults": {
      "homeDefaultsPath": "~/.code.vibrator/defaults.code.vibrator",
      "importOnNewProject": true
    },
    "git": {
      "preRunCommitEnabled": false,
      "commitMessageTemplate": "<derivedFromPrompt>",
      "alwaysAddAll": true
    },
    "promptAugmentation": {
      "includeReadonlyJson": true
    }
  }
}
```

9.2.3 Home-Defaults-Datei (User-Home)

Defaults werden im User-Home ebenfalls in einer Da`.code.vibrator` (oder in einem Default-Verzeichnis mit dieser Datei) gehalten und können als Startbestand in Projekte importier werden.

```
{
  "schemaVersion": "1.0",
  "scope": "defaults",
  "defaults": {
```

```
    "textFileExtensions":
".java;.kt;.groovy;.xml;.json;.yml;.yaml;.properties;.gradle;.md;.txt;.csv;.ts;.js;.jsx;.tsx;.
    "textFileExactNames": "Makefile;Dockerfile;gradlew;gradlew.bat",
    "promptAddOns": {
      "ao_no_refactor": { "title": "Keine Refaktorierung", "content": "..." }
    }
  }
}
```

Regel: - We `defaults.textFileExtensions` fehlt oder leer ist, wird sie beim ersten Start/bei Bedarf mit einem vordefinierten Standardset für Softwareentwicklung initialisiert.

(User-Home) Defaults werden im User-Home ebenfalls in einer `.code.vibrator` (oder in einem Default-Verzeichnis mit dieser Datei) gehalten und können als Startbestand in Projek importiert werden.

```
{
  "schemaVersion": "1.0",
  "scope": "defaults",
  "defaults": {
    "promptAddOns": {
      "ao_no_refactor": { "title": "Keine Refaktorierung", "content": "..." }
    }
  }
}
```

9.3 Ableitungslogik (für Implementierung)

- **Exclude from Zip**:
- Wenn `zipSelectors.excludeDir=true`, ist das Verzeichnis (und alle Children) für den Zip-Export ausgeschlossen.
- Wenn in einem Parent `excludeDir=true` gilt, ist Child ebenfalls ausgeschlossen (vererbend).
- `excludeChildDirs` markiert direkte Subdirs als ausgeschlossen, ohne dass dafür in deren eigener Ebene zwingend eine Datei existieren muss.

- Exclude hat Vorrang vor Selektoren (Selektoren werden für Exportentscheidung in excluded Bereichen nicht ausgewertet).

- **Selektoren-Text** `selectorsText`) erzeugt pro Pattern automatisch einen Selektor in dieser Ebene, stets `force=true`.

- **Ausmultiplizierte Selektoren** aus Parent erscheinen in `expandedOverrides` und sind standardmäßig `force=false` (inherit). Wird `force=true`, gilt `active` aus der Child-Ebene.
- Dasselbe Modell gilt für readonly-Dateipatterns.
- Readonly-Verzeichnisse können über `readonlyDir=true` (für die aktuelle Ebene) oder über `readonlyChildDirs` (für direkte Subdirs) gesetzt werden.

---

9. ChatGPT-Integration (Pro-Abo, No-API, Human-in-the-Loop)

9.1 Grundprinzip

- Keine programmgesteuerte Steuerung der ChatGPT-Weboberfläche.
- Keine automatisierte DOM-Interaktion, kein automatisches Auslesen von Antworten.
- Der Nutzer bleibt aktiv im Loop (Upload, Download erfolgen manuell im Browser).

Ziel: Maximale Konformität mit Nutzungsbedingungen bei gleichzeitig hoher Ergonomie.

9.2 Unterstützter Workflow

Phase A: Vorbereitung

- Dateiauswahl und Profilkonfiguration abgeschlossen
- Prompt + AddOns definiert

Zentraler Aktionsbutton: „Send to ChatGPT“

Beim Klick werden deterministisch folgende Schritte ausgeführt: .1 Zip gemäß aktivem Profil erzeugen 2. Optiona`git add -A` `git commit` 3. Finalen Prompt zusammensetzen (inkl. AddOns + JSON-Kontext) 4. Prompt in Zwischenablage kopieren 5. ChatGPT im Standardbrowser öffnen oder in den Vordergrund bringen

Es erfolgt **keine automatische DOM-Interaktion** und kein automatisches Einfügen in das Eingabefeld.

Nach Ausführung erscheint ein Statusdialog: - ✔️ Zip erstellt - ✔️ Prompt in Zwischenablage kopiert - ➡️ Bitte im ChatGPT-Fenster einfügen (Strg+V / Cmd+V) - Hinweis auf zu uploadend Datei

Optional im Dialog: - Button: „Erneut kopieren" - Anzeige des Prompt-Anfangs (read-only)

Optionale Komforteinstellung (experimentell)

Einstellungsoption (Standard: deaktiviert):

"Automatisches Einfügen im Browser versuchen (experimentell)"

Wenn aktiviert: - Versuch eines OS-abhängigen Paste-Befehls (z. B. Ctrl+V / Cmd+V) - Keine Ga Erfolg - Fehler werden nicht kritisch behandelt

---

Phase B: Ergebnis-Übernahme

Nach manueller Interaktion im Browser: - Nutzer lädt Result-Zip herunter - Zurück in CodeVib
Button: "Result-Zip importieren"

Import erfolgt ausschließlich nach explizitem Nutzerklick.

Vor Import: - Diff-/Vorschau-Dialog - Anzeige blockierter Dateien (Readonly) - Keine Löschoperationen

- Button: "Result-Zip importieren"
- Optional: Automatische Erkennung der neuesten Zip-Datei im konfigurierten Download-Ordner
- Vorschau-Dialog (Diff-/Report-Ansicht)
- Anwendung unter Beachtung von:
- Readonly-Dateien
- Readonly-Verzeichnissen
- Keine Löschoperationen

9.3 Import- / Diff-Dialog (Sicherheits- und Ablaufdefinition)

9.3.1 Vergleichsbasis

- Vergleich erfolgt ausschließlich gegen das **aktuelle Working Directory**.
- Git-Stand wird nicht berücksichtigt.
- Vor jeglicher schreibender Operation werden:
- Vollständiger Datei-Vergleich
- Readonly-/Exclude-Prüfungen
- Strukturvalidierung des Result-Zips durchgeführt.
- Bei fatalen Konflikten wird **vor** jeder Schreiboperation vollständig abgebrochen.

9.3.2 Dateikategorien und Verhalten

1. **Neue Datei** (existiert lokal nicht) → Wird erstellt.

2. **Geänderte Datei** (Inhalt unterscheidet sich) → Wird überschrieben.

3. **Identische Datei** → Wird nicht überschrieben. → Ursprüngliches Dateidatum bleibt erhalten.

4. **Readonly-Datei** → Niemals überschreiben. → Warnung im Result-Log.

5. **Readonly-Verzeichnis** → Analog zu Readonly-Datei.

6. **Exclude-from-Zip-Verzeichnis (Importfall)** → Dateien werden nicht übernommen. → Warnung im Result-Log. → Nutzer kann bei Bedarf manuell aus dem gespeicherten Result-Zip arbeiten.

7. **Binärdateien** → Behandlung wie normale Dateien. → Ausnahme: ausführbare Dateien werde ignoriert. → Warnung im Result-Log.

8. **Lokal vorhandene Datei fehlt im Result-Zip** → Keine Löschoperation.

9.3.3 Namenskonflikte

- Bei Konflikten durch unterschiedliche Groß-/Kleinschreibung bleibt die bestehende Schreibweise erhalten.
- Warnung im Result-Log.

9.3.4 UI-Struktur

Layout: - Linke Seite: Verzeichnis-Tree - Rechte Seite: Datei-Tabelle (sortierbar nach Name un Eigenschaften)

Filteroptionen: - Gruppierung nach Kategorie (Checkbox ein/ausblendbar) - "Nur geänderte Dateien anzeigen"

Farbcodierung: - Neu (grün) - Geändert (gelb) - Blockiert (rot) - Konflikt (Warnsymbol)

Interaktion: - Keine Einzeldatei-Override-Möglichkeit.

9.3.5 Diff-Anzeige

- Klick auf geänderte Textdatei öffnet Side-by-Side-Diff-Popup.
- Binärdateien kein Diff.
- Encoding und BOM unverändert aus dem Zip übernehmen.

9.3.5 a Änderungserkennung (präzisiert)

- **Textdateien**: Zeichenbasierter Vergleich nach Whitespace-Normalisierung:
- Whitespace-Definition entspricht der vom Java-Compiler erkannten Whitespace-Kategorie gemäß Java Language Specification.
- Technische Umsetzung: Verwendung von `Character.isWhitespace(char)` (bzw. äquivalente JLS-konforme Prüfung).
- Aufeinanderfolgende Whitespace-Sequenzen werden zu genau einem einzelnen Whitespace-Zeichen normalisiert.
- Vergleich erfolgt auf Basis der normalisierten Zeichenketten.
- **Textdatei-Erkennung**: ausschließlich anhand der Dateiendung (konfigurierbar, siehe Defaults).
- **Nicht-Textdateien**: Hashvergleich.

9.3.6 Konfliktfälle (harte Abbruchregeln)

(harte Abbruchregeln)

Der gesamte Vorgang wird vor Schreiboperationen abgebrochen bei: - Lokalen Änderungen währe des Workflows - Readonly-Verletzungen - Unerwarteter Zip-Struktur - Sonstigen strukturellen Inkonsistenzen

Nach Abbruch: - Vollständiges Result-Log anzeigen - Nutzer muss Zyklus neu starten

Regeln sind nicht verhandelbar.

9.3.7 Transaktionsmodell

- Schreibende Operationen nur, wenn keine fatalen Konflikte vorliegen.
- Kein partieller Import bei Konflikten.
- Kein automatisches Löschen.

9.3.8 Git-Integration

- Commit erfolgt ausschließlich **vor Send**.
- Standard: automatischer Commit aktiviert.
- Checkbox springt nach jedem Zyklus wieder auf "AN".
- Nach Import kein automatischer Commit.

9.3.9 Arbeitsverzeichnis & Logging

- Result-Zip-Dateien werden in einem tageweisen Arbeitsverzeichnis gespeichert.
- Zeitstempel-Format: `yyyy_MM_dd_HH_mm`.
- Result-Log wird dort mit Zeitstempel gespeichert.
- Result-Log wird nach jedem Zyklus angezeigt.

9.4 Prompt-Struktur (Erweiterung)

Der finale Prompt kann folgende strukturierte Bestandteile enthalten:

1. Freitext-Basis-Prompt
2. Aktivierte AddOns
3. JSON-Block mit:
4. Aktives Profil
5. Readonly-Dateiregeln
6. Readonly-Verzeichnisse
7. Refactoring-Policy
8. Optional: Metadaten (Timestamp, Tool-Version)

Beispiel:

```
=== CODEVIBRATOR CONTEXT (JSON) ===
{
  "profile": "default",
  "readonly": {
    "dirs": ["secrets"],
```

```
    "filePatterns": ["*.pem", "*.key"]
  },
  "refactorAllowed": false
}
=== END CONTEXT ===
```

---

10. Schnittstellen & Integrationen

UX / Bedienkonzept

11.1 Anwendungstyp

- CLI
- Desktop (JavaFX / Swing)
- Web (z. B. REST-Backend)

11.2 Kern-User-Flow

11.3 Fehlerzustände

- Validierungsfehler
- Systemfehler
- Netzwerkfehler

---

11. MVP-Definition

12.1 Im MVP enthalten

- Funktion A
- Funktion B

12.2 Nicht im MVP

- Feature X
- Feature Y

---

12. Teststrategie

- Unit-Tests
- Integrationstests
- End-to-End-Tests
- Manuelle Tests

Testframeworks: - JUnit: - Mockito:

---

13. Distribution

- Lokale Desktop-Anwendung
- Keine Server-Komponenten
- Keine Datenbank
- Speicherung ausschließlich in `.code.vibrator` Dateien im Projekt sowie optional im User-Home

---

14. Risiken & Annahmen

Risiken & Annahmen

15.1 Technische Risiken

15.2 Abhängigkeiten

15.3 Annahmen

---

16. Offene Fragen

Fortlaufende Sammlung zur Iteration im Vibe-Coding-Prozess.

---

17. Änderungsverlauf

Version	Datum	Änderung	Autor
0.1		Initiale Struktur	