Testkonzept & Testfallspezifikation

MT5 Multi-Backtester - Qualitaetssicherung fuer Release-Faehigkeit

Praxisorientiertes QA-Konzept fuer Management, Entwicklung und Release-Entscheidung. Schwerpunkt: robuste Unit-/Component-Tests, risikoarme MT5-Integration, Datenintegritaet und automatisierte CI-Qualitaetsgates.

Dokumentenversion	2.0 - Managementfassung
Datum	07. Mai 2026
Projekt	MT5 Multi-Backtester / Java Desktop Orchestrator
Klassifizierung	Intern / Vertraulich
Zielgruppe	Geschaeftsfuehrung, Projektleitung, Entwicklung, QA

Inhaltsuebersicht

1. Management Summary

Der MT5 Multi-Backtester ist eine Java-basierte Desktop-Anwendung zur Automatisierung komplexer Backtest-, Optimierungs- und Robustheitslaeufe fuer MetaTrader 5. Aus QA-Sicht ist das System besonders kritisch, weil es lokale Prozesse steuert, MT5-Konfigurationsdateien erzeugt, externe Tickdaten verarbeitet, Reports parst, Strategiekennzahlen persistiert und daraus fachliche Entscheidungen ueber Trading-Strategien ableitet.

73
ausgearbeitete Unit-/Component-Testfaelle

> 85%
Ziel-Coverage fuer Engine und Report-Parser

13,5 PT
realistische Umsetzungsschaetzung inkl. CI und Review

0
echte MT5-/Internet-Abhaengigkeiten in Unit-Tests

> **Entscheidungsvorlage:** Empfohlen wird die Umsetzung der beschriebenen Testbasis als verbindliches Release-Gate. Der groesste Nutzen entsteht durch Entkopplung externer Systeme, deterministische Parser-Fixtures und automatisierte Regressionstests in der CI-Pipeline.

1.1 Managementrelevante Kernaussagen

Aspekt	Bewertung	Konsequenz fuer Management
Geschaeftsrisiko	Fehlerhafte Backtest- oder Robustheitskennzahlen koennen zu falschen Strategieentscheidungen fuehren.	QA muss Kennzahlenberechnung, Parser und Datenintegritaet priorisieren.
Technisches Risiko	MT5-Prozesssteuerung, Dateikodierungen, SQLite-Persistenz und BI5-Datenverarbeitung sind fehleranfaellige Integrationspunkte.	Externe Systeme werden in Unit-/Component-Tests konsequent gemockt; Integrationslaeufe erfolgen separat.
Release-Risiko	Ohne automatisierte Regressionstests sind Parser- und Range-Logik-Aenderungen nur manuell absicherbar.	CI-Gates mit Testpassrate, Coverage und Smoke-Test werden verbindlich.
Investitionsnutzen	Die Tests senken manuelle QA-Aufwaende und reduzieren Risiko bei Weiterentwicklung von Optimizer, Robustness Scanner und Reporting.	Einmaliger Aufwand von ca. 13,5 PT ist wirtschaftlich vertretbar.

2. Projektkontext und Qualitaetsziele

Die Anwendung orchestriert MetaTrader 5 ueber CLI/INI-Dateien, erzeugt Parameterdateien, startet Backtests und Optimierungen sequenziell, extrahiert Ergebnisse aus HTML/XML-Reports, persistiert diese in SQLite und erstellt HTML-/Chart-Reports. Zusaetzlich integriert sie Dukascopy-Tickdaten und verarbeitet BI5-Dateien in MT5-kompatible CSV-Formate.

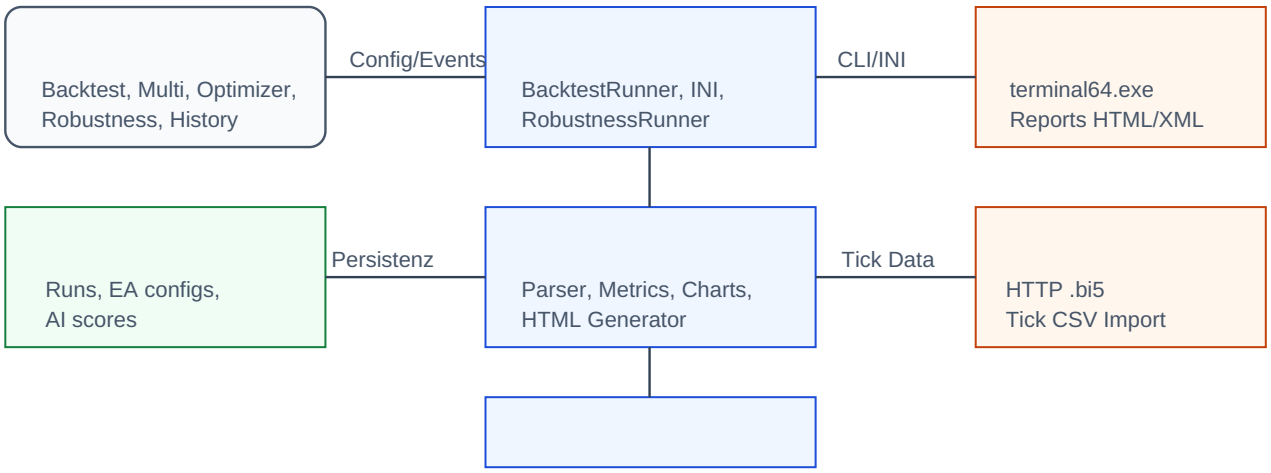

Abbildung 1: Architekturuebersicht mit QA-relevanten Test-Seams

2.1 Qualitaetsziele

Ziel	Messbares Kriterium	Nachweis
Deterministische Kernlogik	Parameterbereiche, Permutationen, CV-/Plateau-Logik und Score-Parsing liefern reproduzierbare Ergebnisse.	Unit-Tests mit festen Testdaten und Grenzwerten.
Parser-Zuverlaessigkeit	Deutsch-/englischsprachige MT5-Reports, UTF-16LE/UTF-8 und korrupte Reports werden korrekt behandelt.	Fixture-basierte Parser-Tests und Negativtests.
Datenintegritaet	SQLite-CRUD, Transaktionen und Bulk-Inserts sind atomar und nachvollziehbar.	Temp-SQLite-Tests, Rollback-Szenarien, Query-Tests.
Resilienz externer Abhaengigkeiten	MT5, HTTP/Dukascopy und KI-API koennen ausfallen, ohne die Anwendung unkontrolliert zu beenden.	Mocks fuer Prozess-, Netzwerk- und Dateifehler.
Release-Faehigkeit	CI-Pipeline blockiert Releases bei roten Tests oder Coverage-Unterschreitung.	JUnit/Jacoco-Berichte und Pull-Request-Gates.

3. Teststrategie, Teststufen und Scope

Das Konzept folgt einem risikobasierten Testansatz. Im Sinne aktueller ISTQB-Leitlinien werden Testbasis, Testbedingungen, Testfaelle, Testergebnisse und Fehler nachvollziehbar miteinander verknuepft. Die Testpyramide wird projektspezifisch interpretiert: breite Unit-/Component-Testbasis fuer schnelle Regression, selektive Integrationstests fuer MT5/Dukascopy/SQLite und nur wenige UI-nahe End-to-End-Szenarien.

3.1 Teststufen

Teststufe	Ziel	Beispiel	Automatisierungsgrad
Unit-Test	Isolierte Absicherung einzelner Methoden/Klassen.	PermutationCount, ScoreParser, CSV-Row-Parser.	Sehr hoch; JUnit 5, AssertJ.
Component-Test	Mehrere Klassen innerhalb eines Moduls mit gemockten Randsystemen.	BacktestRunner mit gemocktem ProcessExecutor und ReportFixture.	Hoch; Mockito/WireMock/ Fakes.
Integrationstest	Kontrolliertes Zusammenspiel mit echter Temp-SQLite oder lokalen Testdateien.	DatabaseManager gegen Temp-SQLite; ReportGenerator schreibt HTML.	Mittel; in CI mit stabilen Fixtures.
System-/Smoke-Test	Minimale End-to-End-Verifikation einer installierten Anwendung.	Ein exemplarischer Backtestlauf mit vorbereitetem Testprofil.	Niedrig bis mittel; manuell oder nightly.

3.2 In Scope / Out of Scope

In Scope	Out of Scope fuer diese Testbasis
• Engine, ReportParser, Optimization-/Robustness-Logik • SQLite-Persistenz und Query-Logik • Dukascopy Downloader, BI5 Decoder, CSV Converter • UI-Modelle, Formatter und Exporter • CI-Gates, Coverage und Regression	• Visuelles Pixel-Testing kompletter Swing-Oberflaechen • Performance-Benchmarking realer MT5-Optimierungen • Finanzfachliche Validierung einzelner Trading-Strategien • Produktive OpenRouter-/Dukascopy-Verfuegbarkeitstests in Unit-CI

4. Testprozess, Rollen, Umgebungen und Werkzeuge

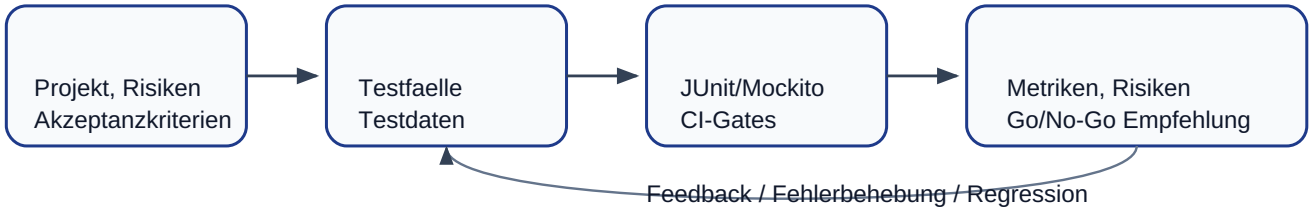

Abbildung 2: Testprozess von Testbasis bis Release-Entscheidung

4.1 Rollen und Verantwortlichkeiten

Rolle	Verantwortung	Ergebnisartefakte
Management / Sponsor	Priorisierung, Budgetfreigabe, Release-Entscheidung.	Freigabeentscheidung, akzeptiertes Restrisiko.
QA Lead / Test Manager	Teststrategie, Testabdeckung, Reporting, Qualitaetsgates.	Testkonzept, Testbericht, Risiko-/ Metrikendashboard.
Java Development	Implementierung testbarer Architektur, Unit-Tests, Fixes.	Testcode, Mocks, Refactoring, PRs.
DevOps / Build	CI-Pipeline, Maven, JaCoCo, Artefakt-Archivierung.	Pipeline-Konfiguration, Coverage-Reports.
Product Owner / Fachseite	Akzeptanzkriterien fuer Kennzahlen und Reports.	Abnahmekriterien, fachliche Priorisierung.

4.2 Testumgebungen

Umgebung	Zweck	Konfiguration	Abhaengigkeiten
Local Developer	Schnelle Unit-/Component-Tests vor Commit.	Java 17+, Maven, JUnit 5, Mockito, JaCoCo.	Keine reale MT5-/Internetpflicht.
CI Linux	Regression bei Pull Requests.	Headless Java, Temp-Verzeichnisse, statische Fixtures.	MT5-Prozess gemockt; SQLite als Temp-Datei.
Windows QA	Pfad-, Installer- und Smoke-Tests nahe Zielplattform.	Windows, gebuendeltes JRE/MSI, optional MT5-Testinstallation.	Kontrollierter Smoke-Test, keine Massentests.

Testdaten-Repository	Versionierte Reports, BI5- und CSV-Fixtures.	Read-only Fixtures mit Checksummen.	Keine produktiven Kundendaten.

4.3 Werkzeuge

Kategorie	Tool	Einsatz
Test Runner	JUnit 5 Jupiter	Standard fuer Java Unit-/Component-Tests.
Mocking	Mockito / Mockito-inline; optional WireMock	Prozess-, HTTP-, Filesystem- und API-Simulation.
Assertions	AssertJ	Lesbare Assertions und aussagekraeftige Fehlermeldungen.
Coverage	JaCoCo Maven Plugin	Line-/Branch-Coverage als CI-Qualitaetsgate.
Static Fixtures	Versionierte HTML/XML/BI5/CSV-Testdaten	Reproduzierbare Parser- und Converter-Tests.

4.4 Fehlerlebenszyklus

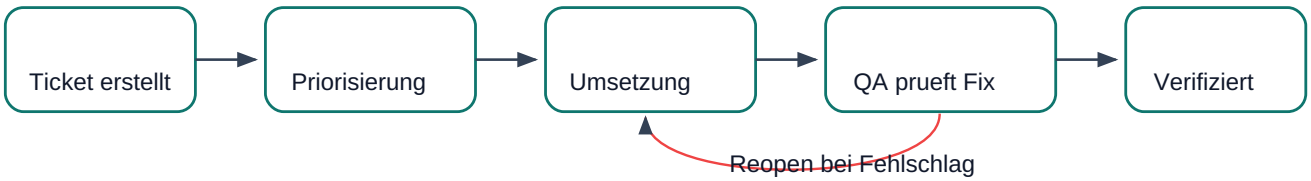

Abbildung 3: Fehlerlebenszyklus mit Retest und Reopen

Status	Definition	Exit-Kriterium
Neu	Fehler wurde durch Test, Review oder Smoke-Test gefunden.	Reproduzierbare Schritte und erwartetes Ergebnis dokumentiert.
Triage	Prioritaet, Schweregrad und Verantwortlicher sind festgelegt.	Ticket ist sprint-/releasefaehig priorisiert.
In Fix	Entwicklung implementiert Korrektur und Regressionstest.	PR enthaelt Fix und Testnachweis.
Retest	QA verifiziert Fix und fuehrt relevante Regression aus.	Testfall gruene Ausfuehrung; keine Folgefehler.
Closed	Fehler ist behoben oder fachlich akzeptiert.	QA Lead schliesst mit Begruendung.

5. Risiken, Metriken und Release-Gates

5.1 Projektspezifische Risiken

Risiko	Auswirkung	Wahrscheinlichkeit	Massnahme
Falsches Report-Parsing	Fehlerhafte Profit-/Drawdown-/Sharpe-Werte fuehren zu falscher Strategieauswahl.	Hoch	Mehrsprachige Fixtures, negative Parser-Tests, Grenzwerttests.
MT5-Prozess haengt oder crasht	Batch-Laeufe blockieren, UI/CI wird unzuverlaessig.	Mittel	ProcessExecutor abstrahieren, Timeout-/Crash-Tests, Daemon-Pipe-Reader testen.

Falsche Kodierung bei .set/Reports	Parameter werden von MT5 falsch interpretiert oder Reports nicht gelesen.	Mittel	UTF-16LE/BOM/UTF-8-Fallback explizit testen.
Dukascopy-Datenluecken	Backtests beruhen auf unvollstaendigen Tickdaten.	Mittel	Gap Detection, Checksummen, Retry/Resume-Tests.
SQLite-Inkonsistenz bei Bulk-Insert	Historie/Optimierungsergebnisse werden unvollstaendig gespeichert.	Mittel	Transaktions- und Rollback-Tests.
Zu stark gekoppelte Klassen	Tests werden teuer oder instabil.	Mittel	Dependency Injection fuer Process, HTTP, Clock, FileSystem.

5.2 Qualitaetsmetriken

Metrik	Zielwert	Kommentar
Automatisierte Testpassrate	100% fuer Merge und Release	Rote Tests blockieren Pull Request/Release.
Line Coverage Engine/Report	> 85%	Fokus auf wertstiftende Logik; Coverage ist Indikator, kein Selbstzweck.
Branch Coverage kritischer Parser/Range-Logik	> 75%	Grenzwerte, Fehlerpfade und Lokalisierungsvarianten abdecken.
Unit-/Component-Testlaufzeit	< 15 Sekunden lokal fuer die 73 Tests	Verhindert versteckte echte I/O-/Sleep-Abhaengigkeiten.
Defect Leakage	0 kritische Parser-/Persistenzfehler nach Release	Messbar ueber Bugs, die im produktiven Gebrauch statt in QA gefunden werden.
Flaky Test Rate	0 tolerierte Flaky Tests	Instabile Tests werden repariert oder isoliert, nicht ignoriert.

5.3 Release-Gates

Gate	Kriterium	Go/No-Go-Regel
PR Gate	Alle Unit-/Component-Tests gruen, Maven Build erfolgreich.	No-Go bei rotem Test oder Buildfehler.
Coverage Gate	JaCoCo erreicht Zielwerte fuer kritische Pakete.	No-Go bei Unterschreitung ohne genehmigte Ausnahme.
Fixture Gate	Parser-/BI5-/CSV-Fixtures versioniert und reproduzierbar.	No-Go bei Tests, die Live-Internet/MT5 benoetigen.
Smoke Gate	Windows QA Smoke-Test mit Installer und Beispielkonfiguration erfolgreich.	No-Go bei Blocker/Critical Defects.

6. Detaillierte Testfallspezifikation

Die folgenden 73 Testfaelle sind aus den Stichpunkten der vorliegenden Spezifikation auf Senior-QA-Niveau ausformuliert. Jeder Testfall enthaelt ID, Ziel, Vorbedingungen, Testdaten, konkrete Schritte und erwartete Ergebnisse. Die Testfaelle sind als Mindestumfang fuer die naechste releasefaehige Testautomatisierungsstufe zu verstehen.

6.1 Core Engine & Backtest Logic (25 Tests)

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-ENG-01	**SetFile_ValidParams** Prioritaet: P1 | Typ: Unit/ Component Validiert, dass aus einer vollstaendigen EA-Parameterkonfiguration eine MT5-kompatible .set-Datei erzeugt wird.	EaParameterManager ist mit temporaerem Tester-Verzeichnis initialisiert; Beispiel-EA und valide Parameterliste liegen vor.	EA=MeanReversionEA; MagicNumber=123456; Lots=0.10; UseTrailingStop=true; RiskPercent=1.5; Encoding=UTF-16LE mit BOM.	• Parameterliste als EaParameter-Objekte aufbauen und Zielpfad in ein leeres Temp-Verzeichnis setzen. • writeSetFile/Save-Funktion ausfuehren. • Datei binaer einlesen und BOM, Zeilenstruktur sowie Schluessel/Wert-Paare pruefen. • Datei erneut ueber Reader einlesen und Roundtrip gegen Eingabe vergleichen.	Datei existiert, ist nicht leer, enthaelt BOM/ korrekte Zeilenenden und alle Parameter exakt einmal; boolesche/ numerische Werte bleiben unveraendert.
TC-ENG-02	**SetFile_MissingParams** Prioritaet: P1 | Typ: Unit/ Component Prueft die Validierung fachlich zwingender Parameter vor dem Schreiben einer .set-Datei.	Konfiguration enthaelt EA-Name und Zielpfad, aber mindestens ein fachlich benoetigter Parameter fehlt.	Fehlender Parameter Lots; vorhandene Parameter MagicNumber=123456, StopLoss=100.	• Minimalkonfiguration ohne Lots erzeugen. • Validierungs-/ Schreibmethode ausfuehren. • Exception und Logeintrag erfassen. • Pruefen, ob im Zielverzeichnis keine Teil-Datei erzeugt wurde.	Es wird eine fachlich eindeutige IllegalArgumentException oder ValidationException mit Parametername erzeugt; keine defekte Datei bleibt zurueck.
TC-ENG-03	**SetFile_InvalidPath** Prioritaet: P1 | Typ: Unit/ Component Sichert Fehlerbehandlung bei nicht vorhandenem oder nicht beschreibbarem Zielpfad ab.	Valide Parameterliste; Zielpfad zeigt auf schreibgeschuetzten oder absichtlich ungueltigen Ordner.	Zielpfad=/root/blocked/ ea.set oder Temp-Ordner ohne Write-Recht.	• Zielpfad auf nicht beschreibbaren Ordner setzen. • Schreibvorgang starten. • Rueckgabestatus, Exception-Typ und Logmeldung erfassen. • Pruefen, ob der aufrufende Runner den Fehler als kontrollierten Abbruch bewertet.	Der Fehler wird sauber protokolliert, an den Aufrufer propagiert und fuehrt nicht zu einem unkontrollierten Prozessabbruch.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-ENG-04	**SetFile_SpecialChars** Prioritaet: P1 | Typ: Unit/Component Verifiziert, dass Sonderzeichen in Parameternamen und Textwerten verlustfrei verarbeitet werden.	Parameterliste enthaelt Werte mit Umlauten, Leerzeichen, Gleichheitszeichen und Backslashes.	Kommentar="Trendfolge EUR/USD äöü = aktiv"; Pfad="C:\MT5\MQL5\Files".	• Sonderzeichen-Parameter anlegen. • Datei schreiben. • Datei mit erwarteter Kodierung einlesen. • Werte exakt gegen Eingabewerte vergleichen.	Alle Zeichen werden korrekt gespeichert und wieder eingelesen; Trennzeichen werden nicht faelschlich als neue Parameter interpretiert.
TC-ENG-05	**SetFile_EmptyConfig** Prioritaet: P1 | Typ: Unit/Component Prueft das Verhalten bei leerer Parameterkonfiguration.	EaParameterManager ist initialisiert; Parameterliste ist leer.	Leere Liste [] fuer EA=EmptyConfigEA.	• Leere Liste uebergeben. • Validierung/Schreibvorgang starten. • Ergebnisobjekt oder Exception pruefen. • Zielverzeichnis auf neue Dateien kontrollieren.	Der Vorgang wird mit klarer Fehlermeldung abgewiesen; es wird keine leere .set-Datei als gueltige Konfiguration erzeugt.
TC-ENG-06	**SetFile_LinuxPathing** Prioritaet: P1 | Typ: Unit/Component Prueft plattformneutrale Pfadbehandlung bei Linux/CI-Ausfuehrung.	Test laeuft in Linux-CI oder PathProvider ist auf Linux gemockt.	ReportPath=/tmp/mt5/reports/run-001; EA path mit Unix-Separators.	• OS-/PathProvider auf Linux setzen. • tester.ini und .set-Referenzen generieren. • Generierte Dateien textuell analysieren. • Keine hart codierten Windows-Trenner im Linux-spezifischen Pfadteil zulassen.	Pfade sind konsistent und CI-faehig; keine unbeabsichtigten Backslash-Escapes verhindern Dateizugriffe.
TC-ENG-07	**SetFile_WindowsPathing** Prioritaet: P1 | Typ: Unit/Component Prueft korrekte Windows-Pfadformatierung fuer MT5-Terminalaufrufe.	PathProvider ist auf Windows gemockt; MT5-Basispfad ist gesetzt.	terminal64.exe=C:\Program Files\MetaTrader 5\terminal64.exe; ReportPath=C:\Backtests\run-001.	• OS-/PathProvider auf Windows setzen. • BacktestConfig erzeugen. • tester.ini schreiben. • INI-Datei auf Backslashes, Quoting und nicht doppelte Trenner pruefen.	Windows-Pfade werden MT5-kompatibel ausgegeben; Leerzeichen im Programmpfad werden nicht falsch gesplittet.
TC-ENG-08	**SetFile_Overwriting** Prioritaet: P1 | Typ: Unit/Component Stellt sicher, dass bestehende .set-Dateien deterministisch ueberschrieben werden.	Im Zielverzeichnis existiert bereits eine .set-Datei mit alten Parametern.	Alt: Lots=0.01; Neu: Lots=0.20, RiskPercent=2.0.	• Alte Datei mit bekannten Werten erzeugen. • Neue Konfiguration schreiben. • Dateiinhalt lesen. • Pruefen, dass alte Werte nicht mehr enthalten sind.	Die Datei enthaelt ausschliesslich die neue Konfiguration; es entstehen keine doppelten oder veralteten Eintraege.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-ENG-09	**Runner_StandardExecution** Prioritaet: P1 | Typ: Unit/ Component Validiert den erfolgreichen Ablauf eines gemockten MT5-Backtests ohne reale terminal64.exe.	BacktestRunner nutzt injizierten ProcessExecutor; statischer Beispielreport liegt im Temp-Verzeichnis.	ExitCode=0; Report=valid-report-de.html; Symbol=EURUSD; Timeframe=H1.	• BacktestConfig erzeugen. • ProcessExecutor mit erfolgreichem ExitCode mocken. • ReportLocator auf Beispielreport zeigen lassen. • Runner starten und Ergebnis auswerten.	Runner liefert SUCCESS, ReportParser-Ergebnis ist gesetzt und Run kann in History persistiert werden.
TC-ENG-10	**Runner_MT5Crash** Prioritaet: P1 | Typ: Unit/ Component Prueft kontrollierte Behandlung eines ploetzlichen MT5-Prozessabbruchs.	ProcessExecutor ist so gemockt, dass ein nicht-null ExitCode oder abgebrochener Prozess geliefert wird.	ExitCode=1; stderr="terminal crashed"; kein Reportfile vorhanden.	• Runner mit Crash-Mock starten. • ExitCode und stderr erfassen. • Reportsuche ausfuehren lassen. • Resultstatus und Logmeldung pruefen.	Runner meldet FAILED mit technischer Ursache; Folgelaeufe im Multi-Backtest koennen weiterlaufen.
TC-ENG-11	**Runner_Timeout** Prioritaet: P1 | Typ: Unit/ Component Sichert Timeout-Verhalten bei haengendem MT5-Prozess ab.	ProcessExecutor simuliert Prozess ohne Abschluss; Timeout ist im Test auf kurzen Wert konfiguriert.	Timeout=500 ms; Prozess-ID=mock-4711.	• BacktestRunner starten. • Zeitablauf bis Timeout simulieren. • kill/destroyForcibly-Aufruf verifizieren. • Fehlerstatus und Logeintrag pruefen.	Der Prozess wird beendet, der Lauf als TIMEOUT/FAILED klassifiziert und keine Endlosschleife blockiert die CI-Pipeline.
TC-ENG-12	**Runner_LogParsingSuccess** Prioritaet: P1 | Typ: Unit/ Component Prueft Extraktion zentraler Kennzahlen aus einem validen MT5-Report.	Statischer MT5 HTML/ XML-Report in deutscher oder englischer Lokalisierung liegt vor.	Net Profit=1250.50; Drawdown=7.2%; Profit Factor=1.65; Trades=104.	• Reportdatei bereitstellen. • ReportParser ausfuehren. • Kennzahlen aus BacktestResult lesen. • Werte gegen erwartete Dezimalwerte vergleichen.	Profit, Drawdown, Profit Factor, Sharpe/Recovery soweit vorhanden und Trade-Anzahl werden korrekt geparst.
TC-ENG-13	**Runner_LogParsingCorrupt** Prioritaet: P1 | Typ: Unit/ Component Validiert Fehlerrobustheit bei leerem oder korruptem MT5-Report.	Reportdatei existiert, enthaelt aber unvollstaendiges HTML/XML oder 0 Bytes.	report.xml="<html><table>" oder leere Datei.	• Korrupte Datei erzeugen. • ReportParser aufrufen. • Exception/ Fehlerobjekt erfassen. • Runner-Fehlerbehandlung pruefen.	Parser meldet einen klaren Parse-Fehler; Runner markiert Lauf als fehlgeschlagen, ohne NullPointerException.
TC-ENG-14	**Runner_ZeroTrades** Prioritaet: P1 | Typ: Unit/ Component Sichert die fachliche Behandlung erfolgreicher Backtests ohne Trades.	Valider Report mit Total Trades=0 und neutraler Equity liegt vor.	Profit=0.00; TotalTrades=0; Symbol=USDJPY; Zeitraum=2020-2024.	• Zero-Trades-Report parsen. • BacktestResult erzeugen lassen. • Klassifikationslogik ausfuehren. • Persistenz-/UI-Flag pruefen.	Der Lauf wird nicht als technischer Fehler behandelt, aber fachlich als NO_TRADES/ ungueltig fuer Strategieauswahl markiert.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-ENG-15	**Runner_NegativeProfit** Prioritaet: P1 | Typ: Unit/ Component Prueft korrekte Verarbeitung negativer Backtestergebnisse.	Valider Report mit Verlust und Trades liegt vor.	NetProfit=-435.75; Drawdown=18.4%; Trades=88.	• Loss-Report parsen. • BacktestResult-Kennzahlen pruefen. • Sortier-/ Bewertungslogik ausfuehren. • UI-/Report-Klassifikation auswerten.	Negative Werte bleiben numerisch korrekt erhalten und werden als Verlustlauf gekennzeichnet, nicht abgeschnitten oder positiv konvertiert.
TC-ENG-16	**Runner_MarginCall** Prioritaet: P1 | Typ: Unit/ Component Validiert die Erkennung fachlich nicht akzeptabler Margin-Call-Laeufe.	Report oder Log enthaelt Margin-Call-/ Stop-Out-Hinweis.	Logzeile="margin call" oder Equity Drawdown=100%.	• Report/Log mit Margin-Call-Indikator bereitstellen. • Parser und Klassifikationslogik ausfuehren. • Resultstatus pruefen. • Warnmeldung fuer Report/History pruefen.	Lauf wird als INVALID/ MARGIN_CALL klassifiziert und nicht als robuste Strategie in Auswahltabellen aufgenommen.
TC-ENG-17	**Math_PermutationCount** Prioritaet: P1 | Typ: Unit/ Component Prueft die Anzahl generierter Optimierungspasses fuer Start/Step/Stop-Bereiche.	Parameterbereichslogik ist isoliert testbar.	Start=10; Stop=20; Step=2; erwartete Werte: 10,12,14,16,18,20.	• Range-Objekt erzeugen. • PermutationCount berechnen. • Werteliste generieren. • Count und Grenzwerte pruefen.	Die Anzahl enthaelt beide Grenzen korrekt; fuer das Beispiel werden 6 Passes erwartet.
TC-ENG-18	**Math_InvalidStep** Prioritaet: P1 | Typ: Unit/ Component Sichert Validierung eines Step-Wertes von 0 ab.	Parameterbereichslogik wird mit Step=0 aufgerufen.	Start=1.0; Stop=2.0; Step=0.0.	• Range-Objekt mit Step=0 erzeugen. • Validierung ausfuehren. • Exception erfassen. • Fehlermeldung auf fachliche Klarheit pruefen.	Step=0 wird vor der Berechnung abgelehnt; keine Division durch Null oder Endlosschleife tritt auf.
TC-ENG-19	**Math_NegativeStep** Prioritaet: P1 | Typ: Unit/ Component Prueft rueckwaerts laufende Parameterbereiche.	Parameterbereichslogik unterstuetzt negative Steps oder lehnt sie kontrolliert ab.	Start=20; Stop=10; Step=-2.	• Range mit negativem Step erzeugen. • Werteliste generieren. • Count und Reihenfolge pruefen. • Validierungsstatus auswerten.	Wenn negative Steps erlaubt sind, entstehen 20,18,16,14,12,10; andernfalls wird eine klare Validierungsmeldung erzeugt.
TC-ENG-20	**Math_FloatingPointPrecision** Prioritaet: P1 | Typ: Unit/ Component Verifiziert robuste Dezimalarithmetik bei feinen Lot-/Step-Werten.	Berechnung nutzt BigDecimal oder vergleichbare Rundungsstrategie.	Start=0.01; Stop=0.05; Step=0.01; Scale=2.	• Dezimalbereich erzeugen. • Werteliste generieren. • String- und Numerikwerte vergleichen. • Auf typische Floating-Point-Artefakte pruefen.	Ergebniswerte lauten exakt 0.01, 0.02, 0.03, 0.04, 0.05; keine Artefakte wie 0.0200000001.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-ENG-21	**Math_MinMaxInverted** Prioritaet: P1 | Typ: Unit/ Component Prueft Validierung invertierter Grenzen bei positivem Step.	Parameterbereichslogik wird mit Start > Stop und positivem Step aufgerufen.	Start=20; Stop=10; Step=2.	• Range erzeugen. • Validierung starten. • Exception oder Fehlerobjekt erfassen. • UI-faehige Fehlermeldung pruefen.	Der Bereich wird vor Ausfuehrung abgelehnt; es werden keine 0-Pass-Optimierungen gestartet.
TC-ENG-22	**AI_ScoreExtraction_Valid** Prioritaet: P1 | Typ: Unit/ Component Validiert Extraktion eines Stabilitaets-Scores aus einer LLM-Antwort.	AI-Response-Parser ist isoliert; keine echte OpenRouter-Verbindung.	Antwort="Analyse... Score: 85/100 ... Begruendung".	• Antwortstring an Parser uebergeben. • Score extrahieren. • Grenzwertvalidierung 0-100 ausfuehren. • Synchronisation ins Ergebnisobjekt pruefen.	Der Integerwert 85 wird extrahiert, validiert und in Sensitivity/Combined/ Selected-Modellen konsistent gesetzt.
TC-ENG-23	**AI_ScoreExtraction_Invalid** Prioritaet: P1 | Typ: Unit/ Component Prueft Fallback bei nicht auswertbarer LLM-Antwort.	AI-Response-Parser erhaelt Antwort ohne Score oder mit ungueltigem Wert.	Antwort="Keine eindeutige Bewertung moeglich" oder Score=abc.	• Antwortstring an Parser uebergeben. • Parserresultat lesen. • Log/Warnung pruefen. • Folgeverarbeitung starten.	Parser liefert definierten Fallback 0 oder UNKNOWN; Anwendung bleibt bedienbar und protokolliert die Ursache.
TC-ENG-24	**AI_PromptGeneration** Prioritaet: P1 | Typ: Unit/ Component Prueft Vollstaendigkeit des Prompts fuer die KI-Stabilitaetsbewertung.	Sensitivity-Ergebnis mit Kennzahlen und Parameter-CV liegt vor.	Profit=2200; Drawdown=8.1%; Trades=130; ParameterCV: Lots=3.2%, SL=11.0%.	• Sensitivity-Ergebnisobjekt erzeugen. • Promptgenerator aufrufen. • Prompttext auf alle Muss-Felder pruefen. • Keine sensiblen Pfade/API-Keys im Prompt zulassen.	Prompt enthaelt relevante Kennzahlen, Parameterbereiche und klare Bewertungsskala 0-100; geheime Konfigurationswerte fehlen.
TC-ENG-25	**AI_NetworkError** Prioritaet: P2 | Typ: Component Sichert kontrollierten Umgang mit Netzwerk-/API-Fehlern bei OpenRouter ab.	AI-Client ist gemockt und liefert Timeout/ HTTP 5xx.	HTTP 503 bei Versuch 1-3; Retry-Policy aktiv.	• AI-Client mit Fehlerantwort konfigurieren. • Bewertung anfordern. • Retry-Anzahl und Backoff-Verhalten verifizieren. • Endstatus und UI-Meldung pruefen.	Retries werden begrenzt ausgefuehrt; danach bleibt der Sensitivity-Lau erhalten und der AI-Score wird als nicht verfuegbar markiert.

6.2 Database Management (16 Tests)

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-DB-01	**DB_InitSQLite** Prioritaet: P1 | Typ: Unit/ Component Prueft Initialisierung der lokalen SQLite-Testdatenbank inklusive Tabellenanlage.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Temp-SQLite-Datei; Tabellen HISTORY_RUNS und EA_SAVED_CONFIGS erwartet.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Verbindung ist aktiv, Schema existiert und Testdatenbank ist isoliert vom Benutzerprofil.
TC-DB-02	**DB_InvalidPath** Prioritaet: P1 | Typ: Unit/ Component Validiert Fehlerbehandlung bei nicht erreichbarem DB-Pfad.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	DB-Pfad zeigt auf nicht beschreibbaren Ordner.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	SQLException wird kontrolliert gemeldet; Anwendung startet nicht mit halb initialisierter DB.
TC-DB-03	**DB_SchemaMigration** Prioritaet: P1 | Typ: Unit/ Component Sichert idempotente Schemaanlage beim ersten und zweiten Start ab.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Leere DB-Datei, danach erneuter Init-Aufruf.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Tabellen werden einmalig angelegt; erneuter Start erzeugt keine Dubletten/Fehler.
TC-DB-04	**DB_GracefulShutdown** Prioritaet: P1 | Typ: Unit/ Component Prueft geordnetes Schliessen der SQLite-Verbindung.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Aktive Verbindung mit einer offenen Leseoperation.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	close() gibt Ressourcen frei; Datei kann danach geloescht werden.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-DB-05	**CRUD_InsertValid** Prioritaet: P1 | Typ: Unit/ Component Prueft Speichern eines vollstaendigen Backtest-/ Optimierungslaufs.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	HistoryRun mit RunType=BACKTEST, EA, Symbol, ReportPath und JSON-Metriken.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Datensatz ist per SELECT wieder auffindbar und Felder stimmen exakt.
TC-DB-06	**CRUD_InsertDuplicate** Prioritaet: P1 | Typ: Unit/ Component Sichert Verhalten bei doppelten IDs oder gleichen fachlichen Schluesseln ab.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Zwei HistoryRun-Objekte mit gleicher ID.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Duplicate wird abgefangen; definierte Upsert- oder Fehlerrichtlinie wird eingehalten.
TC-DB-07	**CRUD_ReadById** Prioritaet: P1 | Typ: Unit/ Component Validiert Lesen eines einzelnen HistoryRun anhand der ID.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Vorbereiteter Datensatz mit JSON-Kennzahlen.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Korrektes Objekt wird inklusive JSON und Zeitstempel rekonstruiert.
TC-DB-08	**CRUD_ReadAll** Prioritaet: P1 | Typ: Unit/ Component Prueft gefiltertes Lesen aller Laeufe fuer eine Strategie.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Drei Laeufe fuer EA_A, zwei Laeufe fuer EA_B.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Query liefert nur EA_A mit korrekter Anzahl und Reihenfolge.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-DB-09	**CRUD_UpdateScore** Prioritaet: P1 | Typ: Unit/ Component Prueft Aktualisieren des AI Stability Scores.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Bestehender Sensitivity-Lauf mit Score NULL, neuer Score=87.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Nach Update ist Score=87 in allen relevanten Views/Queries sichtbar.
TC-DB-10	**CRUD_DeleteSelected** Prioritaet: P1 | Typ: Unit/ Component Prueft Loeschen einzelner markierter History-/Pass-Datensaetze.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	IDs 2 und 4 sollen geloescht werden.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Nur diese IDs fehlen; andere Datensaetze bleiben unveraendert.
TC-DB-11	**CRUD_DeleteAll** Prioritaet: P1 | Typ: Unit/ Component Validiert Reset-Funktion ohne Schemaverlust.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Tabelle enthaelt mehrere Laeufe.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	COUNT=0, Tabellenstruktur und Indizes bleiben erhalten.
TC-DB-12	**Bulk_InsertPerformance** Prioritaet: P1 | Typ: Unit/ Component Prueft Transaktionsperformance bei grossen Optimierungsresultaten.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	10.000 Passes mit Profit/Drawdown/ Sharpe.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Bulk-Insert laeuft in akzeptabler Zeit und nutzt eine Transaktion.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-DB-13	**Bulk_RollbackOnError** Prioritaet: P1 | Typ: Unit/ Component Sichert atomare Transaktion bei Fehler waehrend Bulk-Insert.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Fehlerhafte Zeile 50 mit NULL in Pflichtfeld.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Gesamte Transaktion wird zurueckgerollt; keine Teilmenge bleibt bestehen.
TC-DB-14	**Query_BestProfit** Prioritaet: P1 | Typ: Unit/ Component Prueft Sortierung nach hoechstem Net Profit.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Passes mit Profit 100, 500, -50.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Ergebnis ist strikt absteigend nach Profit sortiert.
TC-DB-15	**Query_LowestDrawdown** Prioritaet: P1 | Typ: Unit/ Component Prueft Sortierung/Filter nach niedrigstem Drawdown.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Drawdown-Werte 3.5%, 12.1%, 7.0%.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Ergebnis ist aufsteigend sortiert; niedrigster Drawdown steht oben.
TC-DB-16	**Query_ComplexFilter** Prioritaet: P1 | Typ: Unit/ Component Validiert kombinierten Strategiefilter.	DatabaseManager nutzt temporaere SQLite-Testdatenbank; Tests laufen transaktional und unabhaengig.	Profit > 1000 und Drawdown < 10%.	• Testdatenbank/ Schema vorbereiten. • Zu pruefende Datenbankoperation ausfuehren. • Daten direkt per Query bzw. DAO wieder lesen. • Rueckgabestatus, Persistenzzustand und Fehlermeldungen verifizieren.	Nur Datensaetze, die beide Kriterien erfuellen, werden geliefert.

6.3 Dukascopy API & Tick-Data (18 Tests)

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-DUK-01	**HTTP_DownloadSingleBi5** Prioritaet: P1 | Typ: Unit/Component Prueft Download einer einzelnen .bi5-Stundendatei ueber gemockten HTTP-Client.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	Symbol=EURUSD; Datum=2024-01-02 10:00 UTC; HTTP 200 mit statischem .bi5-Bytearray.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Datei wird im Cache abgelegt und als vollstaendig markiert.
TC-DUK-02	**HTTP_NotFound** Prioritaet: P1 | Typ: Unit/Component Sichert Verhalten bei nicht vorhandener Dukascopy-Datei ab.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	HTTP 404 fuer Wochenende/Feiertag.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Downloader protokolliert Hinweis und setzt keinen Fehler fuer erwartbar leere Daten.
TC-DUK-03	**HTTP_RateLimit** Prioritaet: P1 | Typ: Unit/Component Prueft Retry/Backoff bei HTTP 429.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	HTTP 429, danach HTTP 200.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Backoff wird angewendet; Download wird erfolgreich fortgesetzt.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-DUK-04	**HTTP_NoConnection** Prioritaet: P1 | Typ: Unit/Component Prueft Netzwerkfehler ohne echte Internetverbindung.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	SocketTimeoutException im HTTP-Client.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Fehler wird nutzerverstaendlich gemeldet; Anwendung stuerzt nicht ab.
TC-DUK-05	**HTTP_InvalidSymbol** Prioritaet: P1 | Typ: Unit/Component Validiert Ablehnung nicht unterstuetzter Symbole.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	Symbol=XXXYYY.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Symbol wird vor HTTP-Request abgelehnt oder als fachlicher Fehler geloggt.
TC-DUK-06	**BI5_DecodeValid** Prioritaet: P1 | Typ: Unit/Component Prueft LZMA/BI5-Dekompression einer validen Tickdatei.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	Statisches .bi5-Fixture mit mehreren Bid/Ask-Ticks.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Decoder liefert korrekte Tickanzahl und Wertebereiche.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-DUK-07	**BI5_DecodeCorrupt** Prioritaet: P1 | Typ: Unit/Component Sichert Fehlerbehandlung bei korruptem BI5-Stream.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	Bytearray mit abgebrochener LZMA-Struktur.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Decoder meldet kontrollierten Fehler; Datei wird nicht als valide importiert.
TC-DUK-08	**DL_ChunkingMultiYear** Prioritaet: P1 | Typ: Unit/Component Prueft Aufteilung grosser Zeitraeume in handhabbare Requests.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	Zeitraum 2019-01-01 bis 2024-12-31.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Requests werden deterministisch in Stunden/Tage/Monatsbloecke aufgeteilt.
TC-DUK-09	**DL_ResumePartial** Prioritaet: P1 | Typ: Unit/Component Validiert Wiederaufnahme nach Teilabbruch.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	50% der erwarteten Dateien liegen bereits im Cache.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Downloader ueberspringt vorhandene valide Dateien und setzt am naechsten Chunk fort.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-DUK-10	**DL_DiskFull** Prioritaet: P1 | Typ: Unit/Component Prueft Verhalten bei fehlendem Speicherplatz.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	OutputStream wirft IOException: No space left on device.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Download stoppt sicher; Teildatei wird geloescht oder als unvollstaendig markiert.
TC-DUK-11	**DL_ChecksumVerify** Prioritaet: P1 | Typ: Unit/Component Sichert Integritaetspruefung heruntergeladener Dateien.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	Dateigroesse/Checksumme weicht vom erwarteten Fixture ab.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Fehlerhafte Datei wird erkannt und fuer erneuten Download markiert.
TC-DUK-12	**DL_WeekendHandling** Prioritaet: P1 | Typ: Unit/Component Prueft fachlich korrekte Behandlung leerer Wochenende-Daten.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	Samstag/Sonntag fuer EURUSD.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Leere Antworten werden nicht als technische Fehler gewertet.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-DUK-13	**CSV_ParseValid** Prioritaet: P1 | Typ: Unit/Component Validiert Parsen einer standardisierten Tick-CSV.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	CSV mit Timestamp, Bid, Ask, Volume.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Ticks werden mit korrektem Timestamp und Dezimalwerten in Objekte gewandelt.
TC-DUK-14	**CSV_ParseInvalidHeader** Prioritaet: P1 | Typ: Unit/Component Prueft Ablehnung inkompatibler CSV-Header.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	Header ohne Ask-Spalte.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Parser bricht mit klarer Fehlermeldung ab.
TC-DUK-15	**CSV_ParseCorruptRow** Prioritaet: P1 | Typ: Unit/Component Sichert Robustheit bei einzelnen defekten CSV-Zeilen.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	Eine Zeile ohne Ask-Preis zwischen validen Zeilen.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Defekte Zeile wird gezaehlt/geloggt; valide Zeilen bleiben nutzbar.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-DUK-16	**CSV_TransformToMT5** Prioritaet: P1 | Typ: Unit/Component Prueft Umwandlung in MT5-importfaehiges CSV-Format.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	UTC-Zeitstempel, Broker-Time-Offset +2, EURUSD-Point-Mapping.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Ausgabe entspricht MT5-Format mit korrekter Zeitzonenverschiebung.
TC-DUK-17	**CSV_GapDetection** Prioritaet: P1 | Typ: Unit/Component Validiert Erkennung groesserer Datenluecken.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	Luecke > 4 Stunden an einem Werktag.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Gap wird als Warnung im Analyseergebnis und Report markiert.
TC-DUK-18	**CSV_EmptyFile** Prioritaet: P1 | Typ: Unit/Component Prueft Verarbeitung einer 0-Byte-Datei.	HTTP-Client, Dateisystem und Zeitquelle sind gemockt; keine echten Netzwerkzugriffe in Unit-/Component-Tests.	Leere Datei im Cache.	• Testfixture und Mocks vorbereiten. • Downloader/Decoder/Converter-Funktion ausfuehren. • Zwischenergebnis und erzeugte Dateien/Objekte pruefen. • Fehler-, Retry- oder Warnpfade anhand Log/Statusobjekt validieren.	Datei wird ignoriert/neu angefordert; keine NullPointerException.

6.4 UI Models & Reporting (14 Tests)

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-UI-01	**Model_UpdateAsync** Prioritaet: P1 | Typ: Unit/ Component Prueft thread-sichere Aktualisierung von Swing-TableModels aus Hintergrundlaeufen.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	SwingWorker liefert BacktestResult im Hintergrundthread.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	Update erfolgt ueber EDT/ invokeLater; keine ConcurrentModificationException.
TC-UI-02	**Model_ClearData** Prioritaet: P1 | Typ: Unit/ Component Validiert clear() fuer Ergebnistabellen.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	TableModel mit 10 Eintraegen.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	Nach clear(): rowCount=0 und Change-Event wurde ausgeloest.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-UI-03	**Model_RemoveSelected** Prioritaet: P1 | Typ: Unit/ Component Prueft Entfernen mehrerer selektierter Zeilen.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	Selektierte Model-Indizes 2 und 4.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	Exakt diese Datenobjekte fehlen; Reihenfolge der Restdaten bleibt stabil.
TC-UI-04	**Model_PassColumnSync** Prioritaet: P1 | Typ: Unit/ Component Sichert Synchronisation von Passnummern ueber Analyse-Tabs.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	Pass #42 in Optimization, Sensitivity und Combined View.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	Alle Views referenzieren dieselbe fachliche Pass-ID.
TC-UI-05	**Model_FormatCurrency** Prioritaet: P1 | Typ: Unit/ Component Prueft lokalisierte Anzeige von Waehrungswerten.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	1000.5 EUR und -435.75 USD.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	Format ist konsistent mit Locale/ Projektvorgabe und negative Werte bleiben sichtbar.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-UI-06	**Sort_ByScoreAsc** Prioritaet: P1 | Typ: Unit/ Component Validiert aufsteigende Sortierung nach AI Score.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	Scores 0, 87, 42, 100.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	Sortierung ergibt 0, 42, 87, 100.
TC-UI-07	**Sort_ByScoreDesc** Prioritaet: P1 | Typ: Unit/ Component Validiert absteigende Sortierung nach AI Score.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	Scores 0, 87, 42, 100.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	Score 100 steht in Zeile 0, danach 87, 42, 0.
TC-UI-08	**Sort_HandleNulls** Prioritaet: P1 | Typ: Unit/ Component Prueft Sortierung bei fehlenden AI-Scores.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	Scores NULL, 80, NULL, 15.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	NULL-Werte werden konsistent ans Ende sortiert.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-UI-09	**Export_HTMLSuccess** Prioritaet: P1 | Typ: Unit/Component Prueft Generierung eines aggregierten HTML-Reports.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	Mehrere BacktestResult-Objekte inklusive Equity-Charts.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	HTML-Datei existiert, enthaelt Tabelle, Metrikkarten und eingebettete Chartdaten.
TC-UI-10	**Export_HTMLEmpty** Prioritaet: P1 | Typ: Unit/Component Sichert Verhalten bei Export ohne Daten.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	Leere Ergebnisliste.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	Warnung wird ausgegeben; kein irrefuehrender leerer Report wird publiziert.
TC-UI-11	**Export_PDFSuccess** Prioritaet: P1 | Typ: Unit/Component Prueft PDF-Export eines Reports mit Kennzahlen und Diagramm.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	BacktestResult mit Profit, Drawdown und Equity-Serie.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	PDF wird erzeugt, ist lesbar und enthaelt Metriken sowie Diagramm.

ID	Ziel / Beschreibung	Vorbedingungen	Testdaten	Testschritte	Erwartetes Ergebnis
TC-UI-12	**Export_PDFPermissions** Prioritaet: P1 | Typ: Unit/Component Prueft Fehlermeldung bei Export in schreibgeschuetztes Ziel.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	Output-Pfad ohne Schreibrecht.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	Nutzer erhaelt Zugriff-verweigert-Meldung; keine defekte PDF bleibt zurueck.
TC-UI-13	**Chart_DataSeriesAdd** Prioritaet: P1 | Typ: Unit/Component Validiert Hinzufuegen eines Datenpunkts zum Equity-Chart-Modell.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	Balance/Equity-Datenpunkt fuer neuen Trade.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	Series-Count steigt und Repaint/Model-Event wird ausgeloest.
TC-UI-14	**Chart_DataSeriesMax** Prioritaet: P1 | Typ: Unit/Component Prueft Performance-Schutz bei sehr grossen Zeitreihen.	Headless-faehiger Test; Fokus auf TableModels, Formatter, Exporter und Chart-Modelle, nicht auf visuelle Robot-Tests.	>10.000 Equity-Punkte.	• Model/Exporter mit kontrollierten Testdaten initialisieren. • Aktion im EDT bzw. mit gemocktem Zielpfad ausfuehren. • Events, Datenzustand und erzeugte Artefakte pruefen. • Fehlerpfade und Nutzerhinweise validieren.	Downsampling oder Performanceschutz greift; Chart bleibt responsiv.

7. Aufwand, Roadmap und Empfehlung

7.1 Aufwandsabschaetzung

Phase	Umfang	Aufwand	Ergebnis
Setup & Testinfrastruktur	JUnit 5, Mockito, AssertJ, JaCoCo, Base-Fixtures	1,5 PT	Lauffaehiges Testgeruest und Maven/CI-Konfiguration.
Core Engine Tests	25 Tests	3,5 PT	Abgesicherte Runner-, INI/.set-, Parser- und Math-/AI-Logik.
Database Management	16 Tests	2,0 PT	Temp-SQLite, CRUD, Transaktionen, Query-Filter.
Dukascopy & Tickdaten	18 Tests	3,0 PT	HTTP-Fakes, BI5-Fixtures, CSV-Transformation, Gap Detection.
UI Models & Reporting	14 Tests	1,5 PT	Headless-faehige Model-/Exporter-Tests.
CI/CD Integration	Build Gates, JaCoCo, Artefakte	0,5 PT	Automatisierte Qualitaetssicherung bei PRs.
Review, Stabilisierung, Refactoring-Puffer	Vier-Augen-Prinzip, DI-Nacharbeiten	1,5 PT	Stabile, wartbare Tests ohne Flakiness.
Gesamt	**73 Tests**	**13,5 PT**	**Ca. 3 Entwicklerwochen bei 1 Senior Java Engineer.**

7.2 Roadmap

Woche	Schwerpunkt	Meilenstein
Woche 1	Testinfrastruktur, Engine-Refactoring-Seams, Core Engine Tests	MT5-Prozessaufrufe sind mockbar; erste CI-Gates aktiv.
Woche 2	Parser-, Database- und Dukascopy-Fixtures	Risikoarme Tests fuer Reports, SQLite und Tickdaten stehen.
Woche 3	UI Models/Reporting, Coverage-Schliessung, Review und Stabilisierung	Release-Gate erreicht; Management-Testbericht verfuegbar.

> **Empfehlung:** Umsetzung freigeben und als Voraussetzung fuer den naechsten groesseren Release festlegen. Besonders wichtig ist, vor der Testimplementierung technische Test-Seams fuer ProcessExecutor, HTTP-Client, Clock und FileSystem zu schaffen.

8. Referenzen und angewandte Standards

Die Webrecherche wurde bewusst ergaenzend eingesetzt und auf praxisrelevante Standards/Best Practices begrenzt. Die Inhalte wurden nicht als akademische Studie uebernommen, sondern in konkrete Projektvorgaben uebersetzt.

Quelle	Nutzung im Konzept
ISTQB Certified Tester Foundation Level Syllabus v4.0.1	ISTQB, 2024/2025: Testprozess, Testware, Traceability, Rollen und risikobasierter Testansatz.
ISTQB Glossary	Begriffsrahmen fuer Testfall, Testbasis, Testbed, Testcoverage und Defect.

Martin Fowler - Practical Test Pyramid	Einordnung der Testpyramide mit Fokus auf viele schnelle Unit-Tests und wenige teure End-to-End-Tests.
JaCoCo Java Code Coverage Library	Werkzeug fuer Java-Code-Coverage-Berichte in Maven/CI-Pipelines.

8.1 Begriffe

Begriff	Definition im Projektkontext
Testbasis	Projektbeschreibung, Architektur, Roh-Testfaelle, Akzeptanzkriterien und bekannte Risiken.
Testfall	Ausfuehrbare Spezifikation mit ID, Vorbedingungen, Testdaten, Schritten und erwarteten Ergebnissen.
Test-Seam	Architektonischer Entkopplungspunkt, um externe Systeme wie MT5, HTTP oder Dateisystem zu mocken.
Coverage	Messung, welche Codezeilen/Zweige durch Tests ausgefuehrt wurden; dient als Indikator, nicht als alleiniger Qualitaetsbeweis.

> **Hinweis:** Dieses Dokument ersetzt keine produktive Finanz- oder Handelsstrategievalidierung. Es sichert die Softwarequalitaet des Backtest-Orchestrators, der Datenverarbeitung und der Ergebnisdarstellung.