Projekt: Testautomatisierung mit GitHub Actions für die Verwaltung von Kontaktdaten

Lernziele:

- Verstehen des Testpyramiden-Konzepts (Unit vs. Integration vs. End-to-End).
- Schreiben von Unit-Tests für Validierung, Formatprüfung und Geschäftslogik.
- Schreiben von Integrationstests, die mehrere Komponenten zusammenspielen lassen.
- Einrichtung einer GitHub-Actions-Pipeline, die Tests automatisch ausführt und Ergebnisse präsentiert.
- Nutzung von MTP (Microsoft Testing Platform) zur Berichterstattung.

Aufgabenstellung:
Zwei Studierende arbeiten gemeinsam an der Automatisierung von Tests für eine Kontaktdatenverwaltung in C# oder C++. Es ist dazu eine einfache Kontaktdaten-Verwaltung zu erstellen. Eine einfache Kontaktdatenverwaltung ist eine digitale Lösung, um Kontaktdaten wie Adressen, Telefonnummern und E-Mails zu sammeln, zu organisieren und jederzeit abrufbar zu speichern. Sie dient als digitales Adressbuch, um die Beziehung zu Kunden, Interessenten und Partnern zu pflegen.

Minimaler Funktionsumfang der Kontaktdatenverwaltung:

- Kontakt anlegen, bearbeiten, löschen (Name, Telefon, E-Mail, Firma, Adresse).
- Einfache Validierung (E-Mail-Format, Telefonnummer nicht leer).

  Suchen nach Namen funktioniert.
- Suchen nach Kontakten basierend auf verschiedenen Kriterien (Name, Firma etc.).

Es ist eine Testspezifikation mit TUnit oder xUnit zu erstellen. Danach erfolgt die Implementierung von Unit- und Integrationstests. Mit GitHub Actions sind die Tests zu automatisieren. Als Nachweise dienen die Spezifikation der Kontaktdatenverwaltung, die generierten Testprotokolle auf der Microsoft Testing Plattform (MTP).

Mögliches Vorgehen

Student A:

- Spezifikation der Kontaktdatenverwaltung (Anforderungen, Domänenmodell, API/Methodensignaturen).
- Unit-Tests für Validierung, kleine Hilfsfunktionen, edge cases.

Student B:

- Implementierung der Integrationstests (Zusammenspiel von Kontakterstellung, Persistenz im In-Memory-Store, Suche).
- GitHub Actions-Workflow einrichten (Testausführung, Protokollierung, Artefakte).

Gemeinsame Deliverables:

- Gemeinsames Repository mit klarer Ordnerstruktur (src, tests/unit, tests/integration, .github/workflows).
- README mit Build-/Test-Anleitung, Abhängigkeiten, Setup-Anweisungen.
- Spezifikation-Dokument (Dokumentation der Anforderungen, Domänenmodell, Schnittstellen).
- Konkrete, prüfbare Deliverables (Templates)
- Spezifikation (Vorlage):
- Domänenmodell: Contact (Id, Name, Company, Phone, Email, Address).
- Schnittstellen/Methoden: AddContact, UpdateContact, DeleteContact, FindContacts (mit Filter), ValidateContact.
- Nicht-funktionale Anforderungen: Validierung, Fehlerbehandlung, Logging (optional).

Testplan (Vorlage):

- Unit-Tests: ValidateEmail, ValidatePhone, AddContact ValidData, AddContact Duplicate (falls sinnvoll).
- Integrationstests: AddContact + GetContacts (Persistenz im In-Memory-Store), UpdateContact + Search.

Definition of Done (DoD):

- Alle vorgesehenen Tests grün.
- GitHub Actions läuft erfolgreich bei Push/PR.
- MTP-Berichte als Artefakte vorhanden oder in Logs sichtbar.

---

2. Mögliche Schritte im Überblick

a) Projekt vorbereiten:

- Erstelle Spezifikation der Kontaktdatenverwaltung
- Erstelle ein gemeinsames Repository auf GitHub für das Projekt.
- Strukturiere den Code und die Tests entsprechend.

b) Testspezifikation erstellen:

- Dokumentiere die Anforderungen an die Kontaktdatenverwaltung (z.B. Hinzufügen, Bearbeiten, Löschen, Suchen).
- Wähle das Testframework: TUnit oder xUnit (je nach Präferenz).
- Definiere die Testfälle:
    - *Unit-Tests:*
        - Testen einzelner Methoden (z.B. Validierung der Telefonnummer, E-Mail-Format)
    - *Integrationstests:*
        - Testen des Zusammenspiels mehrerer Komponenten (z.B. Datenpersistenz nach Hinzufügen)

c) Implementierung der Tests:

- Schreibe die Unit-Tests für einzelne Methoden.
- Schreibe die Integrationstests, die mehrere Komponenten zusammen prüfen.

d) Automatisierung mit GitHub Actions:

- Erstelle eine Workflow-Datei (.github/workflows/test.yml)
- Konfiguriere die Actions, um bei jedem Push oder Pull-Request die Tests automatisch auszuführen.
- Nutze die Microsoft Testing Plattform (MTP), um die Tests zu laufen lassen und die Protokolle zu generieren.

e) Nachweis:

- Stelle sicher, dass die Testprotokolle (z.B. als Artefakte oder in den GitHub Actions Logs) dokumentiert sind.
- Die Protokolle dienen als Nachweis für erfolgreiche Testläufe.

Hinweise:

- Die Kontaktdatenverwaltung kann z.B. eine Klasse ContactManager sein, die Methoden zum Anlegen, Bearbeiten, Löschen und Suchen von Kontakten enthält.
- Für die Tests können Mock-Objekte verwenden werden, wenn externe Abhängigkeiten bestehen.

- Nutze die MTP-Dokumentation, um die Tests auf der Plattform zu registrieren und auszuführen.

Ergebnis:

Am Ende des Projekts haben die Studierenden:

- Spezifikation (Anforderungen, Design) der Kontaktdatenverwaltung
- Eine vollständige Testspezifikation (Testplan, Testfallspezifikationen)
- Implementierte Unit- und Integrationstests
- Eine automatisierte Testausführung via GitHub Actions
- Testprotokolle als Nachweis der erfolgreichen Testläufe

Bewertungs- und Feedback-System
Kriterien (klar messbar):

- Vollständigkeit der Spezifikation (Abdeckung der Kernfunktionen).
- Qualität der Unit-Tests (Abdeckung, Grenzfälle, Mocking sinnvoll genutzt).
- Qualität der Integrationstests (Zusammenführung von Komponenten, End-to-End-Szenarien).
- Funktion der GitHub Actions (automatisierte Ausführung, Logs, Artefakte).
- Nachweis durchs Protokoll/Reports (MTP-Berichte oder Logs).
- Bewertungsrubrik (Mini-Checkliste):
    - 40% Tests (Unit + Integration, Abdeckung, Stabilität)
    - 20% Spezifikation & Design-Dokumentation
    - 20% Automatisierung & CI (GitHub Actions, Artefakte)
    - 20% Codequalität (Lesbarkeit, Struktur, Kommentare)
- Optionen zur Steigerung oder Vertiefung (optional)
- Gruppenumfang: 2–3 Studierende statt 2. Rollen können wechseln (A vs B) oder gemeinsam alle Aufgaben bearbeiten.
- Erweiterte Funktionen (je nach Zeit/Know-how):
- Gruppen/Kontaktlisten, Rollenbasierte Zugriffskontrollen (in Memory)
- Persistenz in einer Datei (JSON) oder SQLite (kleines Persistenz-Feature)
- Validierung von Adressformaten, E-Mail-Domänen-Checks
- Metriken: Testlaufzeiten, Anzahl der Tests, Abdeckungsgrad (Code Coverage) – optional.

Klarer Zeitplan (empfohlen)

Woche 1: Spezifikation finalisieren, Setup GitHub-Repo, Grundstruktur, erstes Unit-Tests-Template.
Woche 2: Unit-Tests implementieren, erste Integrationstests entwerfen.
Woche 3: Implementierung fortsetzen, GitHub Actions konfigurieren, MTP-Berichte vorbereiten.
Woche 4: Finalisierung, Dokumentation, Präsentation der Ergebnisse.