ipso! Bildung seit 1876

Praktische Arbeit – Applikationen testen – Inf. EFZ

Auftragsbeschreibung (User Story) des Kunden:

Auf einem grossen Industriegelände werden mittels bemannten Gabelstaplern Güter zwischen verschiedenen Standorten transportiert (mehrere Produktions- und Lagerhallen sowie ein QA-Center). Jeder Standort verfügt über einen «Güter-Bahnhof», an dem die Stapler be- und entladen werden.
Der Auftraggeber möchte diese Gabelstapler nun durch autonome (d.h. unbemannte, selbst fahrende und selbst agierende) Elektro-Fahrzeuge ersetzen und stellt sich die Endsituation folgendermassen vor:

Die Güter werden für den Transport in Waren-Behältern gleicher Grösse untergebracht. Auf der Ladefläche jedes Fahrzeugs haben maximal zehn Behälter gleichzeitig Platz.

Die Fahrzeuge kommunizieren miteinander über 5G Mobilfunk, um betriebsrelevante Daten wie aktuelle Position und Beladung laufend abzugleichen. Die Standorte übermitteln ihre Ladeaufträge per Broadcast an alle Fahrzeuge gleichzeitig, wobei der noch zu implementierende Algorithmus entscheidet, welches Fahrzeug den Auftrag entgegennimmt und ausführt.

Ein Transport-Auftrag besteht aus den folgenden Daten:

- Abfertigungs-Standort («VON»)
- Ziel-Standort («NACH»)
- Liste der zu transportierenden Behälter

Ein Fahrzeug kann mehr als einen Auftrag gleichzeitig ausführen, sofern es über genügend Platz verfügt. Ein Auftrag wird aber von genau einem Fahrzeug ausgeführt und nicht auf mehrere Fahrzeuge verteilt. Dementsprechend darf ein Auftrag auch nicht mehr als zehn Behälter enthalten – grössere Chargen werden auf mehrere Aufträge aufgeteilt. Diese Aufteilung erfolgt manuell, d.h. ausserhalb des Systems.

Folgende Kriterien beeinflussen die Entscheidung, welches Fahrzeug den Auftrag entgegennimmt:

1. Aktuelle Beladung: Nur Fahrzeuge, die genügend freien Platz haben, kommen in die Auswahl.
2. Fahrzeugposition: Vorteil für Fahrzeuge, die sich näher beim Abfertigungs-Standort befinden.

Nun soll die Software entwickelt und getestet werden, mit der die **Fahrzeuge die Aufträge zuteilen und abwickeln** (die eigentliche Fahrzeugsteuerung ist nicht Teil des Auftrags).

ipso! Bildung seit 1876

Aufgabenstellung – Testermittlung und -durchführung nach dem «Test First» Ansatz:

1. Ermitteln Sie mit den Ihnen bekannten Werkzeugen Anforderungen (Use Cases) und Test Cases. Erstellen Sie aus den Anforderungen eine Testliste mit mindestens 3 Test Cases mit je 1 Positiv- und 2 Negativszenarien. Konzentrieren Sie sich auf den Prozess «Auftragsdurchführung».

2. Ermitteln Sie die nötigen Klassen. Programmieren Sie pro ermittelte Klasse ein Interface mit den entsprechenden Methoden (inkl. Getter- und Setter-Methoden). Achtung Sie auf Clean Code!

   WICHTIG: Programmieren Sie zunächst nur die Interfaces und nicht die eigentlichen Klassen!

3. Programmieren Sie in einem separaten Visual Studio Projekt die Unit Tests für Ihren zuvor erstellten Testplan. Testen Sie dabei gegen die Interfaces, nicht gegen konkrete Klassen!

   - Erstellen Sie eine Testklasse pro zu testendes Interface, und darin
   - eine Testmethode pro Positiv- und pro Negativ-Case.
   - Verwenden Sie wo sinnvoll parametrisierte Tests, um verschiedene Test-Daten zu prüfen.
   - Achten Sie auf selbsterklärende und konsistente Benennung der Test-Klassen / -Methoden.

4. Überlegen Sie sich an dieser Stelle, ob aus technischer Sicht weitere Unit Tests sinnvoll wären, die Sie in Ihrem Testplan noch nicht berücksichtigt haben, und ergänzen Sie diese in Liste und Code.

5. Erstellen Sie Mocks, wo dies wegen fehlender Implementierung zu Testzwecken notwendig ist. Lagern Sie diese in eine oder mehrere separate Klassen ausserhalb der Testklassen aus (SRP). Anstelle von echten Mocks dürfen Sie auch andere Test Doubles wie Stubs oder Spies verwenden.

6. Erstellen Sie Factory-Methoden, welche von den Unit Tests aufgerufen werden und entweder einen Mock oder eine konkrete (bisher noch nicht implementierte) Klassen-Instanz zurückliefern. Konfigurieren Sie ihre Factory-Methoden so, dass sie zunächst immer nur Mocks zurückliefern.

   Beispiel Factory-Methode (der Teil ab der «else»-Anweisung wird erst in Schritt 8 programmiert):

   ```
   IContainer GetContainer() {
      if (mode == "mock")
         return new ContainerMock(); // liefert einen Mock eines Container-Objekts
      else
         return new Container();     // liefert eine Instanz der realen Container-Implementierung
   }
   ```

   WICHTIG: Die Unit Tests dürfen nur die Factory-Methoden und nicht direkt die Mocks aufrufen!

7. Validieren Sie die Tauglichkeit der Mocks, indem Sie die Tests ausführen und allenfalls debuggen.

   WICHTIG: Tests müssen laut Test First fehlschlagen, bevor die konkrete Implementierung erfolgt.
   WICHTIG: Prüfen Sie bei Fehlern genau, ob der Mock oder aber der Test allfällige Fehler enthält!

8. Implementieren Sie den Prozess «Auftragsdurchführung» und die entsprechenden Klassen, d.h. programmieren Sie die reale Umsetzung der Anforderungen. Die Logik darf rudimentär sein, d.h. Sie dürfen eine sehr einfache Implementierung erstellen. Ergänzen Sie die Factory-Methoden.

9. Schalten Sie die entsprechende Factory-Methode von Mock auf Implementierung um, testen Sie den nun implementierten Prozess «Auftragsdurchführung», und korrigieren Sie ihn bei Bedarf.

10. Überprüfen und verbessern Sie bei Bedarf Ihre gesamte Lösung hinsichtlich Clean Code Prinzipien.

Lieferobjekte: Alle erstellen Dokumente (Testliste etc.), kompletter lauffähiger C# Code.