Dokumentation Praktikum 4

Inhalt

1. Dokumentorganisation

1.1. Autorenliste

Kürzel	Name
LMN	Prof. Dr. Thomas Lehmann
BEY	Jannik Beyerstedt
DAN	Daniel Friedrich
KRH	Martin Kroh
LIL	Sascha Marc Lilie

1.2. Versionen

Version	Erstellt	Autor	Kommentar
0.1	02.06.14	LMN	Initiale Version des Templates.
0.2	12.11.14	LMN	Überarbeitung des Templates.
0.3	25.11.14	KRH	Erster Entwurf für die Softwarekomponente
0.4	01.12.14	BEY	Ergänzung um die Artefakte aus Labor 3
0.5	04.12.14	BEY	Korrektur von Anmerkungen von LMN
1.0	20.12.14	KRH, DAN	Tests erfolgreich, Entwicklung abgeschlossen

2. Analyse

Zuerst wurden aus der Aufgabenstellung die Requirements für die Produktionsanlage, also das Gesamtsystem, rekonstruiert. Außerdem wurde ein Klassendiagramm erstellt, um einen Überblick über den vorhandenen Code zu erhalten.

Im nächsten Schritt ging es dann darum die neuen Anforderungen in Requirements, Tests und die entsprechenden Code-Strukturen umzusetzen. Die Fragen, die dabei auftauchten bezogen sich hauptsächlich darauf, wie weit der Code getestet werden soll. Am Anfange des ersten Termins wurde dann jedoch geklärt, dass nur ein Unit-Test für die neu zu entwickelnde Komponente durchgeführt werden soll, aber kein Test für die Gesamte Software.

Daraufhin mussten dann die Höhenlinien mit entsprechenden Toleranzen aus den vorgelegten Messprotokollen herausgearbeitet werden, sowie eine Methode entwickelt werden, wie aus den Höhendaten die verschiedenen Baufehler erkannt werden.
Das Erkennen der Bauteiltypen wurde mit einem Zustandsautomaten gelöst, der parallel zum bestehenden Zustandsautomaten des Systems läuft.

Die neue Komponente wurde dann zuerst mit einem Unit Test in Software getestet, bevor die gesamte Anlage getestet werden sollte.

3. Requirements/Tests

3.1. System Requirements des bestehenden Systems

1. Die Produktionsanlage untersucht Fertigteile auf metallische Verunreinigungen und sortiert Teile mit metallischen Verunreinigungen über eine Rutsche aus. Alle anderen Teile werden bis an das Ende des Förderbandes transportiert.

3.2. Delta-System-Requirements

2. Durch Prüfung der Höhenlinie werden Fertigteile mit Baufehlern erkannt. Wenn ein Baufehler erkannt wurde, wird das Fertigteil an den Anfang des Förderbandes zurück transportiert. Während des Rücktransportes und bis zum Entnehmen des Teils blinkt die rote Lampe der Ampel der Produktionsanlage. Sobald das Fertigteil den Anfang des Förderbandes erreicht, stoppt das Förderband. Nach dem manuellen Entnehmen des Fertigteils geht die Produktionsanlage wieder in den normalen Betrieb über.
   Randbedingung: Die Fertigteile werden immer so auf das Förderband gelegt, dass die Seite mit der vollen Bauhöhe senkrecht zur Beförderungsrichtung steht.
   Es werden drei Arten von Fertigteilen erkannt:

   2.1. Gutteil: Bauteil von 4x4 Einheiten Grundfläche und 2 Einheiten Höhe, Massiv

   2.2. Baufehler Typ 1: Der hintere Baustein fehlt

   2.3. Baufehler Typ 2: Ein Baustein versetzt angebaut

3. entfallen: Inhalt jetzt Teil von Req. 2
4. Wenn ein fehlerhaftes Teil erkannt wurde, wird die Art des Fehlers (siehe Req. 2) in ein Log-File geschrieben, sowie auf der Konsole ausgegeben.
5. entfallen: Inhalt jetzt Teil von Req. 2

3.3. System Tests

A. Die Produktionsanlage wird nacheinander mit Gutteilen, Teilen mit metallischen Verunreinigungen, sowie Fertigteilen mit Produktionsfehlern beladen. Es wird dabei immer nur ein Teil auf den Anfang des Förderbandes gelegt, wenn sich kein anderes Teil mehr auf dem Förderband befindet.
Die Teile mit metallischen Verunreinigungen werden auf die Rutsche gelenkt und somit aussortiert.
Die Teile mit Baufehlern werden wieder an den Anfang transportiert. Währenddessen blinkt die rote Lampe der Ampel. Wenn das Teil am Anfang des Förderbandes zum stehen kommt, wird es vom Bediener entnommen und die Anlange geht in den normalen Betrieb über. Auf der Konsole, sowie in der Log-Datei ist der korrekte Fehlertyp zu lesen.
Gutteile werden bis an das Ende des Förderbandes transportiert.
Dieser Test muss zu 100% erfolgreich sein.
Dieser Test erfüllt die Requirements 1, 2 und 4.

3.4. Software Requirements für die neue Komponente

51. entfallen.

52. Der Höhensensor liefert für die möglichen Höhenebenen Werte in den folgenden Wertebereichen:

    52.1. volle und korrekte Bauhöhe: Wert 3082 ± 60

    52.2. mittlere Bauhöhe (Fehler): Wert 3380 ± 60

    52.3. Förderband (kein Teil): Wert 3780 ± 10

    52.4. Alle Werte, die nicht Req. 52.1, 52.2 oder 52.3 entsprechen, sich nicht zulässig.

53. Aus Req. 52 ergeben sich folgende Wechsel der Höhenebenen für die verschiedenen Arten von Fertigteilen (gem Req. 2):

    53.1. Gutteil: Förderband—>volle Höhe—>Förderband

    53.2. Typ 1: Förderband—>volle Höhe—>mittlere Höhe—>Förderband

    53.3. Typ 2: Förderband—>volle Höhe—>mittlere Höhe—>volle Höhe—>Förderband

3.5. Software Tests für die neue Komponente

N. Die Software-Komponente wird per Unit-Test mit Daten, die die drei unterschiedlichen Fertigteil-Arten nach Req. 53 repräsentieren, gespeist. Daraufhin muss über die Schnittstelle result() TRUE ausgegeben werde, wenn das Teil, das durch den Datensatz repräsentiert wird, fehlerhaft ist.
Es wird dabei mit jeweils einem repräsentativen Datensatz getestet.
Außerdem muss in dem Log-File der korrekte, Req. 2 entsprechende, Fehlertyp ausgegeben werden, wenn das Teil einen Fehler aufweist.

4. Design

Zuerst wurde ein Klassendiagramm aus dem vorliegenden Code erstellt, damit man erst einmal einen Überblick über die verschiedenen Komponenten des Software erhält. Welche Klassen gibt es, welche Methoden haben diese, welche Interfaces gibt es und welche Klasse benutzt diese. Antworten auf alle diese Fragen lassen sich sehr gut aus dem Klassendiagramm ablesen.
Außerdem kann an diesem Diagramm gut erkannt werden, welches Interface die neue Komponente implementieren muss und welche Klassen für der Unit-Test benötigt.

Danach wurde der der Steuerung zugrunde liegende Zustandsautomat aus dem Code rekonstruiert. Dieser muss nämlich für die neue Funktionalität erweitert werden, damit das fehlerhafte Bauteil wieder an den Anfang des Förderbandes transportiert wird. Dafür sollte der vorhandene Zustandsautomat bekannt sein, welcher durch ein entsprechendes Diagramm dargestellt wird.
Bei dem Diagrammteil, der das bestehende System abbildet, wurden dieselben Begriffe für Transitions und States wie im Code verwendet. Somit kann schnell eine Verbindung zwischen Code und Diagramm hergestellt werden. Den neuen Teil haben wir zuerst von den Begrifflichkeiten allgemeiner gehalten, da dieser noch keine exakte Implementierung vorgeben soll. Bei der Implementierung wurden alle Diagramme jedoch auf den aktuellen Stand angepasst.

Für die Erkennung der Baufehler wurde ebenfalls ein Zustandsautomat entworfen, der dann entsprechend in Code umgewandelt wurde.

Alle Diagramme sind im Anhang zu finden.

5. Integrationstests

5.1. Implementierung des Software-Tests

Für die Erfüllung des Software-Tests N wurden eine Testklasse „UnitTester“, sowie ein Stub „StubHeightSensor“ erstellt. Der Stub implementiert dabei das Interface „FestoProcessSensors“, das unter anderem für das Übermitteln der Analogwerte des Höhensensors zuständig ist. Der Test wird, wenn die Konstante „TEST“ definiert ist, als erstes in der main-Funktion durchgeführt, indem eine Instanz der Klassen angelegt wird, deren Methode „doTest()“ aufgerufen und die Instanz wieder gelöscht wird. Der Ablauf danach bleibt davon unberührt.

Als Testdaten wurden die gegebenen Höhen-Logdateien processlog 1 bis 3 verwendet, die jeweils einen der zu erkennenden Bauteiltypen darstellen.
Der Konstruktor der Testklasse erstellt zuerst eine Instanz des „StubHeightSensor“ und übergibt diese per dependency injection der unit under Test „heightProfileCheck“.
Die Methode „doTest()“ der Testklasse simuliert dann alle drei Bauteiltypen nacheinander, indem jeweils eine der Logdateien Zeile für Zeile eingelesen wird und die Höhenwerte an den Stub übergeben werden. Danach wird die unit under Test, wie im Normalbetrieb, über die Methode „evalCycle()“ aufgerufen, damit die Werte des Höhensensors ausgewertet werden, und das Ergebnis über die Methode „result()“ abgefragt.

Da das Interface result() den Wert TRUE ausgibt, wenn ein Fehler erkannt wurde, darf bei einem guten Teil das Ergebnis nie TRUE sein. Bei den beiden anderen Fehlertypen muss es genau ein Mal TRUE sein. Die jeweils geprüft und bei einem Fehler eine Warnmeldung auf der Konsole ausgegeben.

5.2. Durchführung des Hardware-Tests

Das Hardware-Software-System wurde gemäß Test A getestet:

Test-Nr.	Beschreibung	Erwartetes Ergebnis	Erfolg?
A1	Einlegen eines Gutteils: Bauteil von 4x4 Einheiten Grundfläche und 2 Einheiten Höhe, Massiv	Teil wird bis an das Ende des Förderbands transportiert.	Ja
A2	Einlegen eines Teils mit Baufehler Typ 1: Ein Baustein fehlt	Teil wird an den Bandanfang zurück transportiert. Rote Lampe blinkt Der Fehlertyp wird auf der Konsole ausgegeben.	Ja
A3	Einlegen eines Teils mit Baufehler Typ 2: Ein Baustein versetzt angebaut	Teile werden an den Bandanfang zurück transportiert. Rote Lampe blinkt Der Fehlertyp wird auf der Konsole ausgegeben.	Ja
A4	Einlegen eines Teils mit metallischer Verunreinigung	Teil wird auf die Rutsche aussortiert.	Ja

6. Fazit

Das Praktikum hat einen guten Einblick gegeben, was es bedeutet, ein bestehendes System im Rahmen der begrenzten Zeit zu erweitern und Programmieraufgaben in einer Gruppe zu verteilen.

Dabei hat sich herausgestellt, dass eine strukturierte Arbeitsweise am Ende deutlich vorteilhafter ist, als einfach mit dem Programmieren loszulegen. Man sollte vorher das vorliegende System, sowohl als gesamte Einheit von Software und Hardware, als auch nur die Software, als Erstes gut dokumentieren. Dies ist vor allem für die Kommunikation zwischen den Gruppenmitgliedern, als auch für die Verteilung der Aufgaben wichtig.

Uns haben insbesondere der Zustandsautomat sowie das Klassendiagramm geholfen, ganz abgesehen davon, dass wir uns auf diese beiden Diagramme beschränkt haben, da mit diesen Diagrammen sowohl die Schnittstellen zwischen den Klassen deutlich wurde, als auch das Verhalten des Systems. Die Diagramme sind außerdem hilfreich gewesen, um eine gemeinsame Wortwahl zu finden.

Es stelle sich aber trotzdem heraus, dass die Aufteilung der Arbeit in gleiche Teile nicht ganz einfach ist.

Beim Testen mit dem Hardware-System ist aufgefallen, dass das System gegebenenfalls auch von der Hardware-Seite umgebaut werden muss. Außerdem sind wir auf ein Problem gestoßen, welches das Umschalten des Motors von dem Rechtslauf auf den Linkslauf betrifft. Dies war nämlich nicht einfach damit getan, dass die Methode „driveLeft()“ aufgerufen wurde. Somit mussten die abstrakteren Zugriffsfunktionen auf die Anlage noch auf die neuen Gegebenheiten angepasst werden. Also auch, wenn der gegebene Code vollständig sein sollte, hat sich herausgestellt, dass dies natürlich nur für das vorher bestehende System gilt, jedoch nicht unbedingt auch für das erweiterte System. Der hier aufgetretene Fehler ist nämlich im ursprünglichen System nie aufgetreten, da die Anlage nie den Linkslauf benutzt hat.
Hierbei wäre sicherlich eine ausführliche Dokumentation der bestehenden Klassen und Methoden von Vorteil gewesen.

7. Anhang

State Machine aus der Klasse „FSM“ (PDF)

State Machine aus der Klasse „heightProfileCheck“ (PDF)

Klassendiagramm (PDF)

Quellcode (Ordner mit allen Dateien)