UNIVERSITÄT KAISERSLAUTERN
FACHBEREICH INFORMATIK

Arbeitsgruppe Software Engineering
Prof. Dr. H. D. Rombach
Gebäude 57 / 5. Stock

Praktikum Software Engineering I

11. Juli 1999

Aufgabe 10

Zur Erfüllung der Aufgabe 10 sollt Ihr das geänderte System anhand der erstellten Systemtestpläne testen. Dafür müßt Ihr Eure Klassen mit denen anderer Teilsysteme integrieren. Die Integration erfordert eine enge Zusammenarbeit zwischen den Gruppen im Praktikum.

Der Test wird in vier Schritten durchgeführt: 1. Implementierung der benötigten Treiber und Stubs, 2. Integration der Klassen des Systems, 3. Ausführung der Tests und Dokumentation der Testergebnisse, 4. Analyse der Testergebnisse und Dokumentation.

Schritt 1: Implementierung der benötigten Treiber und Stubs:

Das System wird zuerst ohne Anbindung an den Simulator/Testraum und ohne Benutzungsschnittstelle getestet. Deshalb sind für den Systemtest (Integrationstest) Test-Treiber und -Stubs zur Durchführung der verschiedenen Systemtests erforderlich.

Jede Gruppe arbeitet auf einem eigenen Verzeichnis, in dem Treiber und Stubs für die Systemtestfälle des Teilsystems abgelegt werden (~Praktik1/System/systemtest/<<Teilsystem>>/). Es stehen ein erster Treiber (.../systemtest/systemtest.cc) und Stub (.../systemtest/portbaustein.hh) zur Verfügung. Der Treiber umfaßt bereits die Initialisierung des Systems. Die im Stub implementierte Methode erlaubt das Verändern von Sensorwerten während des Testlaufs. Die Funktionalität wird an die Sensor- und Aktuatorklassen vererbt.

Die vorgegebenen Treiber und Stubs müßt Ihr zur Durchführung Eurer Testfälle abändern. Vergleicht hierzu die *Hinweise zum Arbeiten* am Ende des Aufgabenblatts.

Die Funktionalität (Rückgabewerte und Ausgaben), die die Test-Stubs bieten müssen, entnehmt Ihr den Systemtestplänen.

Testtreiber und Stubs schreiben relevante Testergebnisse nach cout. Anhand der ausgegebenen Testergebnisse muß das System validierbar sein.

Schritt 2: Integration der Klassen des Systems

Die Integration des Systems macht die Kooperation aller Gruppen im Praktikum erforderlich. Dafür treffen wir uns am Mittwoch, 14.07.99, ab 16.00 Uhr. Mindestens eine Person aus jeder Gruppe muß anwesend sein. Ziel ist die Erstellung eines linkfähigen Binaries. Wenn es länger dauert, geben wir Pizza und Getränke aus!

Gemeinsam binden wir die Object-Files aller Klassen zu einem linkfähigen Binary zusammen. Dafür wird das existierende Makefile im Verzeichnis ~Praktik1/System/systemtest/makefile verwendet.

Schritt 3: Ausführung der Tests und Dokumentation der Testergebnisse

Führt die Tests durch.

Druckt die Testergebnisse (Postscript-Dateien) aus.

Dokumentiert die Ergebnisse der Tests in der Software-Systemdokumentation im Verzeichnis Validation (.../ProbLoesDok/Validation/Ergebnisse<<Teilsystem.fm>>), indem Ihr die Ausgaben der Testdurchläufe in die entsprechende Datei einhängt.

Schritt 4: Analyse der Testergebnisse und Dokumentation

Vergleicht die Testergebnisse mit den erwarteten Ergebnissen der Testpläne und protokolliert gefundene Fehlverhalten in einer Tabelle (eine Tabelle ist bereits in der Datei .../ProbLoesDok/Validation/Ergebnisse<<Teilsystem.fm>> angelegt):

FV-Nr	**Verweis aus KompTest**	**Schritt im KompTest**	**Beschreibung des Fehlverhaltens**
SV-TR-FVNr1	Systemtest 1	Schritt 5	Das Fenster wird geschlossen anstelle gekippt.

Führt jedes gefundene Fehlverhalten auf einen Fehler im Kode (oder Analyse/Entwurf) zurück. Dokumentiert dies mit Hilfe der folgender Tabelle (Datei: .../ProbLoesDok/Validation/Ergebnisse<<Teilsystem.fm>>):

Verweis auf FV-Nr	**F-Nr**	**betroffene Klasse**	**betroffenes Element**	**allgemeine Fehlerbeschreibung**	**Fehlerklasse**	**Prozeß während Fehler gefunden wurde**
SV-TR-FVNr1	SV-TR-FNr-1	Window	Zustandsübergang in der Methode callBack	Beim Zustandsübergang wird FensterSchließen() anstelle FensterKippen() aufgerufen	I	systest-dur

Dokumentiert die aufgetretenen Fehler in den jeweiligen Tabellen (.../ProbLoesDok/Verifikation/Ergebnisse<<Teilsystem>>.fm und .../SoftSysDok/Verifikation/Ergebnisse<<Teilsystem>>.fm und .../SoftKompDok/Verifikation/Ergebnisse<<Teilsystem>>.fm). Beispielsweise in der Datei: .../SoftSysDok/Verifikation/Ergebnisse<<Teilsystem>>.fm

Referenz auf F-Nr	Strategie zur Behebung des Fehlers	Ä.klasse	Behebung des Fehlers durchgeführt (J/N)
SV-TR-FNr1	Zustandsübergang in der Methode callBack: FensterSchliessen() muß aufgerufen werden.	A	N

Es findet keine weitere Praktikumsbesprechung statt. Löscht bitte alle Dateien, die für die Endversion der Systemdokumentation nicht wichtig sind. Am Montag, dem 26.07., werden die Accounts gesperrt.

Wir (die Betreuenden) möchten uns für das gute Klima im Praktikum bedanken. Ihr habt gute Arbeit geleistet, selbst wenn der Aufwand für Euch über dem versprochenen lag. Es war schön (wenn auch teilweise anstrengend ;-)), mit Euch zu arbeiten. Wir hoffen, daß jeder von Euch nicht nur nützliche Einsichten und Erfahrungen für seinen beruflichen Werdegang bekommen hat, sondern auch ein wenig Spaß während des Praktikums hatte.

Hinweise zum Arbeiten mit den Stubs für den Systemtest

Für den Systemtest müssen alle Sensoren und Aktuatoren als Stubs bereitstehen. Als zentrale Anbindung an die konkrete Hardware wurde im Entwurf die Klasse Portbaustein eingeführt. Von dieser Klasse sind alle Sensor- und Aktuatorklassen abgeleitet. Beim Systemtest dient die Klasse Portbaustein zur Realisierung der Stubs. Jede Sensor- und Aktuatorklasse greift letztendlich über die vom Portbaustein geerbten Methoden *CallBitInputPort()*, *CallIntInputPort()*, usw. auf die Hardware zu. Aus diesem Grund ist es möglich, die gesamte Implementierung der Stubs in die Klasse Portbaustein zu verlagern.

Die Implementierung der Stubs geschieht, indem die Methoden *CallBitInputPort()* und *CallIntInputPort()* des Portbausteins so modifiziert werden, daß ein bestimmtes Verhalten der Sensoren erreicht wird. Die Änderungen der Sensorwerte sind beim Systemtest von der Zeit abhängig. Beispielsweise haben die Testfälle folgende Form:

- Der Raum ist leer (Anwesenheitskontakt=0),
- Nach 10 Minuten betritt der Benutzer den Raum (Anwesenheitskontakt=1),
- Nach 4 Minuten verläßt der Benutzer den Raum (Anweseheitskontakt=0)
- ...

Beim Komponententest konnte die Steuerung der Stubs über die Aufrufanzahl bestimmter Funktionen erreicht werden. Für den Systemtest reicht dies jedoch nicht mehr aus; die Abläufe im System sind zu komplex, um genaue Aufrufanzahlen festzulegen. Aus diesem Grund muß zu einer zeitabhängigen Steuerung der Stubs mittels eines Timers übergegangen werden.

In jedem Testfall hat ein Sensor - über die Zeit betrachtet - ein ganz spezielles Verhalten (siehe obiger Testfall):

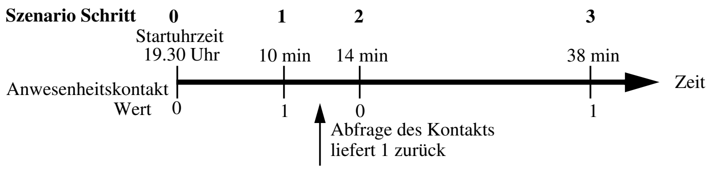

Abbildung 1: Gefordertes Verhalten des Anwesenheitskontakts für Testfall 1

Die gesamte Zeit läßt sich in Intervalle einteilen, in denen ein bestimmter Sensorwert zurückgeliefert wird. Die Zeitintervalle sind dabei relativ zu einer bestimten Startuhrzeit angegeben.

- [0 min, 10 min [: Wert 0
- [10 min, 14 min [: Wert 1
- [14 min, 38 min [: Wert 0
- ...

Für die Implementierung der Stubs kann Abbildung 1 äquivalent in ein Array übertragen werden. Die Definition des Arrays sieht folgendermaßen aus:

```
#define MAXTESTFALL 4       // maximale Anzahl von Testfaellen
#define MAXSCHRITTE 10      // maximale Schritte

struct arraytype {
    int Minuten;    // minutenanzahl relativ zu StartMinute.
    int Wert;       // Sensorwert, der zu diesem Zeitpunkt angenommen wird
  };


arraytype Anwesenheitskontakt[MAXTESTFALL][MAXSCHRITTE];

Anwesenheitskontakt[1][0].Minuten=0;
Anwesenheitskontakt[1][0].Wert=0;
Anwesenheitskontakt[1][1].Minuten=10;
Anwesenheitskontakt[1][1].Wert=1;
Anwesenheitskontakt[1][2].Minuten=14;
Anwesenheitskontakt[1][2].Wert=0;
Anwesenheitskontakt[1][3].Minuten=38;
Anwesenheitskontakt[1][3].Wert=1;

...
```

Zur Definition des Arrays wurden zwei Konstanten `MAXTESTFALL` und `MAXSCHRITTE` eingeführt. Die erste Array-Dimension legt die Abhängigkeit der Sensorwerte von der Testfallnummer fest. In jedem Testfall haben die Sensoren ein unterschiedliches Verhalten (und damit entspricht MAXTESTFALL der Anzahl der durchzuführenden Testfälle). Die zweite Array-Dimension teil das Verhalten des Sensors in die einzelnen Szenario-Schritte (siehe Abbildung 1). Für jeden Szenario-Schritt ist der Zeitpunkt der Wertänderung und der Sensorwert selbst anzugeben.

Wenn zu einem beliebigen Zeitpunkt ein Sensor abgefragt wird, kann der Rückgabewert mit Hilfe des Arrays bestimmt werden. Der folgende Kode zeigt einen Ausschnitt aus der *CallBitInputPort()*-Methode:

Zunächst deklarieren die Zeilen 1 bis 5 verschiedene Hilfsvariablen. Dabei wird `Testfall` vom Treiber mit der aktuellen Testfallnummer belegt. Um den Systemtest mit einer beliebigen Uhrzeit starten zu können, benötigt man die Variable `Startminute`.

```
1:  Time Uhrzeit;
2:  static int AnwesenheitsSchritt;
3:  int result;
4:  int Startminute = 0;
5:  extern int Testfall;
```

Zur Berechnung des Sensorwertes ist die aktuelle Uhrzeit nötig. Diese bestimmt Zeile 7. Dann beginnt die eigentliche Berechnung.

```
6:
7:  Uhrzeit = Zeit_ptr->getActualTime();
8:
```

Der erste Aufruf des Stubs muß einen Startwert zurückgeben (Zeile 10 bis 13). Dieser Punkt entspricht Schritt 0 in Abbildung 1.

Bei jedem weiteren Aufruf der *CallBitInputPort()*-Methode werden nun die Intervallgrenzen abgeprüft. Im Anwesenheitskontakt-Array ist der nächste Änderungszeitpunkt des Sensorwertes festgehalten. Zeilen 16 und 17 prüfen, ob diese Intervallgrenze schon erreicht wurde. Wenn dies der Fall war, wird in Zeile 19 die Variable `AnwesenheitsSchritt` um eins erhöht. Die kommenden Sensorabfragen benutzen somit das nächste Zeitintervall.

Ist die Intervallgrenze noch nicht erreicht, wird der bisher gültige Sensorwert zurückgegeben. Eine Erhöhung von `AnwesenheitsSchritt` findet nicht statt.

```
9:  // Beim ersten Aufruf den InitialWert zurueckgeben
10: if (AnwesenheitsSchritt == 0) {
11:    result = Anwesenheitskontakt[Testfall][0].Wert;
12:    AnwesenheitsSchritt++;
13: } // if
14: // ansonsten ueberpruefen, ob obere Intervallgrenze erreicht ist
15: else {
16:    if ((Uhrzeit.toDuration() - Startminute) >=
17:        Anwesenheitskontakt[Testfall][AnwesenheitsSchritt].Minuten ) {
18:        result= Anwesenheitskontakt[Testfall][AnwesenheitsSchritt].Wert;
19:        AnwesenheitsSchritt++;
20:    } // if
21:    else
22:      result = Anwesenheitskontakt[Testfall][AnwesenheitsSchritt-1].Wert;
23: } // else
24: return result;
```

Abbildung 2: Algorithmus zur Berechnung des Rückgabewertes

Wichtig:

Jeder Testfall setzt eine bestimmte Ausgangssituation voraus, d.h. es werden Annahmen über die Rückgabewerte aller Sensoren gemacht. Selbst wenn sich ein Sensorwert in einem Testfall nicht ändert, muß in dem Stub mindestens ein Intervall mit Rückgabewert definiert sein. Ansonsten funktioniert der vorgestellte Algorithmus zur Berechnung des Rückgabewerts nicht.

Für jeden Systemtest wird ein bestimmter Startzustand vorausgesetzt. Dies kann z.B. bedeuten, daß die Anwesenheit einer Person im Raum gefordert wird (d.h. Anwesenheitskontakt = 1). Wird das Klimasteuerungssystem initialisiert, so stellt sich dieses auf einen Initialzustand ein, der fest kodiert ist (Initialzustände der State-Diagramme). Es ist daher wahrscheinlich, daß es Unterschiede zwischen dem geforderten Startzustand für den Systemtest und dem Initialzustand des Steuerungssystems gibt. Der Initialzustand des PresenceController lautet z.B. „NiemandDa“. Wird Anwesenheit einer Persom im Raum als Startzustand eines Systemtests gefordert, unterscheiden sich diese Zustände. Für die Stubs bedeutet dies, daß es nötig sein kann, zunächst eine Folge von Sensorwerten anzugeben, die das Steuerungssystem in den gewünschten Startzustand für den Systemtest bringt. Wenn eine solche „Einleitungsfolge“ nötig ist, müssen die Reaktionsintervalle des Systems berücksichtigt werden. So erkennt das System z.B. erst nach 10 Minuten die Anwesenheit einer Person im Zimmer.