

Softwareengineering für langlebige Systeme – Übung 5

AUFGABE 1 (Updates) (5BP):

Die Firma Aer Calidus nutzt für Kundenkontakte eine Adressdatenbank. Da die Datenbank alt ist und nicht mehr modernen Anforderungen genügt, soll das System durch eine neue Version ersetzt werden. Dabei sind die Daten zu migrieren.

Da das alte System nicht alle benötigten Daten unterstützt sind, in der Vergangenheit Felder anders genutzt worden, als vorgesehen:

- Es existieren keine Felder für Email, Webseite oder ähnliches. Diese Daten wurden in die Felder Pager, Telex oder Bemerkung eingetragen.
- Es kann pro Kunde nur eine Adresse eingetragen werden. Daher wurden weitere Adressen in weiteren Datensätzen zu dem Kunden gehalten. Die verschiedenen Rollen der Adressen wurden im Bemerkungsfeld gepflegt. Kontaktdaten (Email, Telefon) wurden mit in den entsprechenden Datensätzen gehalten.

Bitte schreiben Sie ein Tool oder Script, welches die alte Datenbank in die neue überführt.

```
create table alt_adressen (  
  id          integer,  
  name        varchar(20),  
  vorname     varchar(40),  
  anrede      varchar(10),  
  strasse     varchar(70),  
  plz         integer(5),  
  ort         varchar(40),  
  telefon     varchar(20),  
  telefax    varchar(20),  
  telex       varchar(20),  
  pager       varchar(20),  
  autotelefon varchar(20),  
  bemerkung   varchar(20)  
);  
  
create table new_contact (  
  client      integer,  
  typ         varchar(20),  
  contactid   varchar(20),  
  remark      varchar(40)  
);  
  
create table new_client (  
  client      integer,  
  address1    varchar(40),  
  address2    varchar(40),  
  address3    varchar(40),  
  zip         varchar(10),  
  city        varchar(40),  
  country     varchar(40),  
  adresstyp   varchar(20)  
);  
  
create table new_client (  
  id          integer,  
  surname     varchar(40),  
  middlename  varchar(40),  
  name        varchar(40),  
  titel       varchar(10),  
  anrede      varchar(10),  
  remark      varchar(20)  
);
```

AUFGABE 2 (Reengineering) (5BP):

Analysieren Sie das Programm.

1. Um welche Sprache bzw. welches Sprachderivat handelt es sich?
2. Für welche Plattform ist das Programm?
3. Wie wird dieses Programm genutzt? Sind spezielle Tools, Hardware oder ähnliches erforderlich?
4. Analysieren Sie die einzelnen Funktionalitäten des Programms. Stellen Sie das gegebene Programm als Modell dar.
5. Extrahieren Sie bitte die Anforderungen, die das Programm erfüllt.

```

#include <DmxSimple.h>
#define maxGroups 5
#define maxGroupmembers 5
#define maxChan 125
#define maxProgs 5

#define sign(i) ((i) < 0?-1:1)

int value = 0;
int channel = 0;
int group = 0;
int groups[maxGroups][maxGroupmembers];
boolean isGroupMode = false;
boolean runProgram[maxProgs];
int programGroup[maxProgs];
int pc[maxProgs];
int farb[] = { 0, 0, 255,
              0, 255, 0,
              255, 0, 0,
              255, 255, 0,
              255, 0, 255,
              0, 255, 255};

int r3 = 0,rz3 = 0,g3 = 0,gz3 =0,b3 = 0,bz3 = 0,y3 = 0,yz3 = 0;
int i4 = 0;
int farb4[] = { 0, 0, 0, 255,
               0, 0, 255, 0,
               0, 255, 0, 0,
               255, 0, 0, 0,
               255, 255, 0, 0,
               0,255, 255, 0,
               0, 0,255, 255,
               255, 0, 0, 255,
               255, 255, 255, 0,
               0, 255, 255, 255,
               255, 0, 255, 255,
               255, 255, 0, 255,
               255, 255, 255, 255};

int chnselect(int c) {
    int result = 0;
    switch (c) {
        case 'w':
        case 'R':
            result = 0;
            break;
        case 'r':
        case 'G':
            result = 1;
            break;
        case 'g':
        case 'B':
            result = 2;
            break;
        case 'b':
        case 'Y':
            result = 3;
            break;
    }
    return result;
}

void setvalue(int basechn, int command, int value) {
    DmxSimple.write(basechn + chnselect(command), value);
}

void addToGroup(int lgroup,int channel) {
    if(lgroup < maxGroups) {
        int j = 0;

```

```
    for(;j < maxGroupmembers && groups[lgroup][j] != 0; j++) {
    };
    if(j < maxGroupmembers) {
        groups[lgroup][j] = channel;
        Serial.print("Channel:");
        Serial.print(channel);
        Serial.print("added to group");
        Serial.println(lgroup);
    }
}
}
```

```
void setup() {
    for(int i = 0; i < maxGroups; i++) {
        for(int j = 0; j < maxGroupmembers; j++) {
            groups[i][j] = 0;
        };
    }
    for(int i = 0; i < maxProgs; i++) {
        runProgram[i] = false;
        programGroup[i] = 0;
        pc[i] = 0;
    }
    DmxSimple.usePin(3);
    Serial.begin(9600);
    Serial.println("ready");
}
```

```
void execProgram(int i) {
    switch (i) {
    case 0:
        if(pc[0] < 500) {
            setGroupValue(programGroup[0], 'r', 255);
        }
        else if(pc[0] < 2000) {
            setGroupValue(programGroup[0], 'r', 0);
        }
        else {
            pc[0] = 0;
        };
        pc[0]++;
        break;
    case 1:
        if(pc[1] < 10) {
            setGroupValue(programGroup[1], 'b', 255);
        }
        else if(pc[1] < 60) {
            setGroupValue(programGroup[1], 'b', 0);
        }
        else {
            pc[1] = 0;
        };
        pc[1]++;
        break;
    case 2:
        if(pc[2] == 1) {
            if(r3 == rz3) {
                rz3 = random(16)*16;
            };
            if(g3 == gz3) {
                gz3 = random(16)*16;
            }
            if(b3 == bz3) {
                bz3 = random(16)*16;
            }
            if(y3 == yz3) {
```

```

        yz3 = random(16)*16;
    }
    b3 += sign(bz3 - b3);
    r3 += sign(rz3 - r3);
    g3 += sign(gz3 - g3);
    y3 += sign(yz3 - y3);
    setGroupValue(programGroup[2], 'R', r3);
    setGroupValue(programGroup[2], 'G', g3);
    setGroupValue(programGroup[2], 'B', b3);
    setGroupValue(programGroup[2], 'Y', y3);
}
else if(pc[2] > 1000) {
    pc[2] = 0 ;
};
pc[2]++;
break;
case 3:
if(pc[3] % 5000 == 1) {
for(int i = 0; i < maxGroups; i++) {
    setGroupValue(i, 'R', farb[3*i+pc[3]/5000]);
    setGroupValue(i, 'G', farb[3*i+pc[3]/5000+1]);
    setGroupValue(i, 'B', farb[3*i+pc[3]/5000+2]);
}
}
else if(pc[3] > 5000*maxGroups) {
    pc[3] = 0 ;
};
pc[3]++;
break;
case 4:
if(pc[4] == 1) {

    if(r3 == rz3 && g3 == gz3 && b3 == bz3 && y3 == yz3) {
        i4 = constrain(i4+4,0,51);
        rz3 = farb4[i4];
        gz3 = farb4[i4+1];
        bz3 = farb4[i4+2];
        yz3 = farb4[i4+3];
    }
    b3 += sign(bz3 - b3);
    r3 += sign(rz3 - r3);
    g3 += sign(gz3 - g3);
    y3 += sign(yz3 - y3);
    setGroupValue(programGroup[2], 'R', r3);
    setGroupValue(programGroup[2], 'G', g3);
    setGroupValue(programGroup[2], 'B', b3);
    setGroupValue(programGroup[2], 'Y', y3);
}
else if(pc[4] > 1000) {
    pc[4] = 0 ;
};
pc[4]++;
break;
}
}

void setGroupValue(int group, int c, int value) {
    for(int j = 0; j < maxGroupmembers; j++) {
        if(groups[group][j] == 0) {
            break;
        }
        setvalue(groups[group][j], c, value);
    }
}

void loop() {
    for(int i = 0; i < maxProgs; i++) {
        if(runProgram[i]) {

```

```
        execProgram(i);
    }
}

void serialEvent() {
    int c;

    //while(!Serial.available());
    c = Serial.read();
    if ((c>='0') && (c<='9')) {
        value = 10*value + c - '0';
    }
    else {
        switch (c) {
            case 'c':
                channel = value;
                isGroupMode = false;
                break;
            case 'Y':
            case 'B':
            case 'G':
            case 'R':
            case 'b':
            case 'g':
            case 'r':
            case 'w':
                if(isGroupMode) {
                    setGroupValue(group, c, value);
                }
                else {
                    setvalue(channel, c, value);
                };
                break;
            case 'a':
                if(value < maxGroups) {
                    addToGroup(value, channel);
                };
                break;
            case 's':
                if(value < maxGroups) {
                    group = value;
                    isGroupMode = true;
                };
                break;
            case 'p':
                if(value < maxProgs) {
                    runProgram[value] = true;
                    programGroup[value] = group;
                }
                break;
            case 'q':
                if(value < maxProgs) {
                    runProgram[value] = false;
                    programGroup[value] = 0;
                }
                break;
            case 'x':
                if(value < maxGroups) {
                    for(int j = 0; j < maxGroupmembers; j++) {
                        groups[value][j] = 0;
                    };
                };
                break;
            case 'z':
                for(int i = 0; i < maxChan; i++) {
                    DmxSimple.write(i, 0);
                };
                break;
        }
    }
}
```

```
case '1':
    Serial.println("Status:\nGroups:");
    for(int i = 0; i < maxGroups; i++) {
        Serial.print("Group");
        Serial.print(i);
        Serial.print("Member:");
        for(int j = 0; j < maxGroupmembers; j++){
            Serial.print(groups[i][j]);
            Serial.print(",");
        }
        Serial.println("");
    }
    Serial.println("\nProgramme:");
    for(int i = 0; i < maxProgs; i++) {
        Serial.print("Programm");
        Serial.print(i);
        Serial.print("run:");
        Serial.print(runProgram[i]);
        Serial.print("Group:");
        Serial.println(programGroup[i]);
    }
}
value = 0;
}
```