arduino Code Testing Pre

Automatisation

engelbert gruber

19.04.2020
Rev. 23.04.2020
1ST GRADE SCIENCE

12:55

INTRODUCTION

Die arduino-Plattform ist gedacht für kleine Programme mit Hardware Interaktionen. Das führt dazu dass niemand mit automatisierten Tests entwickelt. Das wiederum bedeutet dass bei jeder Änderung nur der neue Fall getestet wird und man davon ausgeht, dass alles was vorher funktioniert hat, es auch weiterhin tut.

Tut es nicht.

Tests **müssen** also immer wieder durchgeführt werden. Die Wiederholung führt (bei den meisten Menschen) zu einer erhöhten Fehleranfälligkeit, deshalb wurden Computer entwickelt.

Also lassen wir den Computer testen.

Hier sind die zwei oben genannten Probleme

1. arduino-Programme sind klein, jeder sagt dann Testprogramme sind zu viel Aufwand.
2. Hardware Interaktionen sind nicht einfach zu automatisieren.

Wir versuchen es.

HYPOTHESIS

Die arduino-Programmiersprache ist C++.

Das stimmt nicht wirklich, aber von aussen betrachtet ist das C. Das "++" brauchen wir, weil es in C keine Methoden gibt, der Aufruf "Serial.begin(9600);" nicht möglich ist.

Für die Hardware Interaktionen schreiben wir uns Simulationen.

EINWURF

Damit wir uns nicht selber um die verschiedenen Entwicklungsstufen kümmern müssen, verwenden wir eine Versionsverwaltung. Subversion. Andere gehen auch.

Die Versionsverwaltung legt irgendwo ein Archiv an und wenn man einen Arbeitsschritt hinter sich hat, speichert man diesen dorthin (commit).

Man kann dann jederzeit schauen, was man seit dem letzten commit geändert hat. Mit diff werden die Änderungen in allen Dateien geprüft und angezeigt (ich will mir das nicht merken müssen).

Anlegen des Repositories:

```
svnadmin create /home/engelbert/ko.repo
```

- svnadmin: ist das Programm
- create: der Befehl ein Repository anzulegen
- "`/home/engelbert/ko.repo`" der Ort

Dann wird ein Verzeichnis ko.repo angelegt, darin befindet sich

```
conf  db  format  hooks  locks  README.txt
```

Was auch immer das ist. README.txt kann man anschauen, da steht dass man nichts ändern soll.
Dann erstellt man noch drei Verzeichnisse im Repository:

```
svn mkdir file:///home/engelbert/trunk -m "initial directory
structure. The main development trunk"

svn mkdir file:///home/engelbert/branches -m "initial directory
```

```
structure. Branches are deviations older or experimental."

svn mkdir file:///home/engelbert/tags -m "initial directory
structure. Tags, pointing out important Versions"
```

Wir arbeiten in trunk.

Es gibt auch GUI-Tools für subversion und bei Altium ist das Programm, meines Wissens, inkludiert.

PROCEDURE

Versionsverwaltung

Wir wechseln in unser Arbeitsverzeichnis, "`cd arduino-voltmeter`"

checkout

Und machen eine checkout aller im Repository befindlicher Dateien

```
svn co file:///home/engelbert/ko.repo/trunk/ .
```

Das sind in unserem Fall ... keine.

Wir kopieren unsere erste Datei (von Tobias König) ins Verzeichnis.

```
arduino-voltmeter/arduino-voltmeter.ino
```

add

Fügen dieses Verzeichnis zur Versionsverwaltung hinzu

Eingabe: `svn add arduino-voltmeter`

Programmausgabe:

```
A         arduino-voltmeter
A         arduino-voltmeter/arduino-voltmeter.ino
```

Die beiden Dateien werden nur in die Liste der verwalteteten Dinge übernommen.

diff

Mit "`svn diff`" bekommt man ausgegeben, was lokal anders ist als am Server. Bei uns steht da viel, weil wir das ganze Programm übernommen haben. Der Anfang der

Ausgabe sieht so aus:

```
Index: arduino-voltmeter/arduino-voltmeter.ino
===================================================================
--- arduino-voltmeter/arduino-voltmeter.ino     (nicht existent)
+++ arduino-voltmeter/arduino-voltmeter.ino     (Arbeitskopie)
@@ -0,0 +1,81 @@
+/* Arduino-Voltmeter mit automatischer Messbereichserkennung
+
+ * PBE-Gruber
+ * 17.04.2020
+ * Author: Tobias König
```

Die "`Index:`"-Zeile und "`====`" sind die Überschrift (sage ich einmal so).

Die "`---`" Zeile ist die Version am Server. Wir haben noch nicht "commited" deshalb steht hier "`(nicht existent)`".

Die "+++" Zeile ist die lokale Version.
Die "@@" gibt den Zeilennummern Bereich der Serverversion "-0,0" und der lokalen Version "+1,81".

Danach sind Zeilen die mit "+" beginnen, das sind die von der lokalen Version. Zeilen die in der Serverversion ident sind beginnen mit einem Leerzeichen, die nur in der Serverversion sind mit einem "-", die werden entfernt, die mit "+" kommen hinzu.

Das ist die unified diff Darstellung. Falls es anders aussieht bei euch, das kann man einstellen mit "`svn di -x -u`" beziehungsweise in der svn-Konfiguration.

commit

Der arduino-IDE Syntaxcheck sagt ok. Also committen wir: "`svn ci`" (die Leute sind offensichtlich tippfaul (HTL-Absolventen ?) "ci" steht für commit, commit funktioniert auch, verwende ich aber kaum).

Beim Commit sollte ein Editor aufgehen und man eine kurzen Beschreibung der Änderung eingeben. Hier vielleicht: "Anfangsversion".

Kompilieren am PC

Der cpp sagt:

```
c++ arduino-voltmeter/arduino-voltmeter.ino
/usr/bin/ld:arduino-voltmeter/arduino-voltmeter.ino: file format not
recognized; treating as linker script
/usr/bin/ld:arduino-voltmeter/arduino-voltmeter.ino:7: syntax error
collect2: error: ld returned 1 exit status
```

Der Compiler weiß nicht was eine ino-Datei ist.

Wir machen eine Datei test.cpp:

```
// test rahmen für ein arduino Programm
#include "arduino-voltmeter/arduino-voltmeter.ino"

int main() {
   return 1;
}
```

Und dann geht es, der Compiler meldet geschätzte 25 Fehler in der inkludierten Datei.

(svn add test.cpp , noch kein commit, es geht noch nicht).

Errors

1. Der erste Fehler der ausgegeben wird (beim ersten anfangen, weil spätere Fehler vielleicht aus dem ersten folgen).

   ```
   In file included from test.cpp:5:
   arduino-voltmeter/arduino-voltmeter.ino:7:1: error: ‘byte’ does not name
   a type
         7 | byte inputpin = A0;
           | ^~~~
   ```

   Wir fügen vor dem include die Zeile:

   ```
   #define byte unsigned char
   ```

   ein (der Präprozessor ersetzt “byte” durch “unsigned char” (that is C, character char mit und ohne Vorzeichen :-))

2. Nun ist der erste Fehler:

   ```
   arduino-voltmeter/arduino-voltmeter.ino:7:17: error: ‘A0’ was not
   declared in this scope
   ```

```
7 | byte inputpin = A0;
```

A0 ist die Nummer des Anschlusspins am Arduino. Vor dem include der ino-Datei

```
const int A0 = 0;
const int A1 = 1;
const int A2 = 2;
```

INPUT, OUTPUT sind auch nicht deklariert.

```
const int INPUT = 0;
const int OUTPUT = 1;
```

3. Jetzt fehlen noch : pinMode, Serial.begin, analogWrite, analogRead, Serial.println und delay.

   Komisch: analogWrite ?

   Wir machen ein paar Funktionen:

```
void pinMode(int pin, int mode) {
}
void analogWrite(int pin, int value) {
}
int analogRead(int pin) {
   return -1;
}
void delay(int milliseconds) {
}
```

   Und eine Klasse:

```
class SerialClass {
   void begin(int baudrate) {
   }
   void println(char ln[]) {
   }
};
```

   Und legen ein Object an:

```
SerialClass Serial;
```

Wir brauchen noch die Methode print in der Klasse SerialClass.

```
        void print(char ln[]) {
        }
```

Print und println sollten die Zeile als Konstante deklariert bekommen.

```
   void print(const char ln[]) {
   }
   void println(const char ln[]) {
   }
```

Print und println für float

```
   void print(float value) {
   }
   void println(float value) {
   }
```

und die Methoden sind noch private.

```
      class SerialClass {
      public:
         void begin(int baudrate) {
         }
```

Es kompiliert ohne Fehler und Warnungen

Arduino simluation 1

Wir extrahieren den ganzen Block aus der test.cpp

```
      // arduino Simulation for c++

      #define byte unsigned char
      const int A0 = 0;
      const int A1 = 1;
      const int A2 = 2;

      const int INPUT = 0;
```

```
const int OUTPUT = 1;

void pinMode(int pin, int mode) {
}
void analogWrite(int pin, int value) {
}
int analogRead(int pin) {
   return -1;
}
void delay(int milliseconds) {
}

class SerialClass {
Public:
   void begin(int baudrate) {
   }
   void print(float value) {
   }
   void println(float value) {
   }
   void print(const char ln[]) {
   }
   void println(const char ln[]) {
   }
};

SerialClass Serial;
```

Ändern test.cpp:

```
// test rahmen für ein arduino Programm

#include "arduino-simu.h"
#include "arduino-voltmeter/arduino-voltmeter.ino"

int main() {
   return 1;
}
```

Und ein test-run.sh

```
#!/bin/sh

c++ test.cpp -o test.exe
./test.exe
```

Speichern (“commit”) das jetzt einmal, es kompiliert ohne Meldung, im Repository

```
svn add arduino-simu.h
svn add test-run.sh
svn ci -m “compile without complain”
```

KLEINIGKEITEN IM CODE

analogWrite macht keinen Sinn in dem arduino-voltmeter. Wir ändern das zu digitalWrite und fügen :

```
void digitalWrite(int pin, int value) {
}
```

Zur arduino-simu.h dazu.

Damit wir sehen ob der Compile-Vorgang funktioniert hat löschen wir in test-run.sh vor dem Aufruf des Kompilers die test.exe (rm test.exe, unter windows del text.ese) und probieren es aus (das # vor dem c++ unter windows REM):

```
#!/bin/sh

rm test.exe
#c++ test.cpp -o test.exe
./test.exe
```

Der Aufruf von test-run-sh gibt folgendes aus:

```
./test-run.sh: 5: ./test.exe: not found
```

Und wenn wir das “#” for c++ entfernen, kommt keine Meldung mehr (TODO ich will ein OK).

svn di -x -u

Zeigt die Änderungen an

```
Index: arduino-simu.h
===================================================================
--- arduino-simu.h       (Revision 2)
+++ arduino-simu.h       (Arbeitskopie)
@@ -15,6 +15,8 @@
 int analogRead(int pin) {
      return -1;
 }
+void digitalWrite(int pin, int value) {
+}
 void delay(int milliseconds) {
 }


Index: arduino-voltmeter/arduino-voltmeter.ino
===================================================================
--- arduino-voltmeter/arduino-voltmeter.ino      (Revision 1)
+++ arduino-voltmeter/arduino-voltmeter.ino      (Arbeitskopie)
@@ -21,8 +21,8 @@
   pinMode(s1, OUTPUT);
   pinMode(s2, OUTPUT);

-  analogWrite(s1, 0);
-  analogWrite(s2, 0);
+  digitalWrite(s1, 0);
+  digitalWrite(s2, 0);
   float in = analogRead(inputpin); //Obwohl analogRead() immer Integer
      //zurück gibt, wird float verwendet, um in der Rechnung
Gleikommazahlen als
      //Ergebnis zu bekommen.
Index: test-run.sh
===================================================================
--- test-run.sh (Revision 2)
+++ test-run.sh (Arbeitskopie)
@@ -1,5 +1,6 @@
 #!/bin/sh

+rm test.exe
 c++ test.cpp -o test.exe
```

```
./test.exe
```

Die Zeilen die mit “+” beginnen sind dazugekommen, die die mit “-” beginnen wurden entfernt.

Wir machen zwei Commits.

1. Die Änderung in test-run.sh
   `svn ci test-run.sh -m "remove test.exe before compile"`
2. `den Rest in arduino-simu und voltmeter.`
   `svn ci -m "digitalWrite instead analogWrite"`

Small fixes

1. Im arduino-voltmeter steht “Gleikomma” ändern zu Gleitkomma und commit.
   `svn ci -m "typo"`
2. In testrun.sh steht dreimal test.exe, das stinkt (siehe References)

```
#!/bin/sh
EXE=test.exe
rm $EXE
c++ test.cpp -o $EXE
./$EXE
```

svn ci -m “remove duplicate”

Wir sind bei Revision 6.

16:00

TESTEN

17:30

Was passiert wenn wir im main() von test.cpp loop() aufrufen ?

Kurzes Zwischenspiel: Wir benennen test-run.sh in runtest.sh um … weil ich so will:

```
svn rename test-run.sh runtest.sh
svn ci -m “rename test-run to runtest”
```

Dann :

```
int main() {
```

```
    loop();
    return 1;
}
```

Runtest hängt ohne Ausgabe.

Aktivieren wir den Simulator etwas:

```
void println(const char ln[]) {
    std::cout << ln << std::endl;
}
```

“std::out <<” ist etwas zur Ausgabe/Konsole schicken, hier “ln”.

“std::endl” ist das Zeilenende, end of ln.

Und oben im arduino-simu.h

```
#include <iostream>
using namespace std;
```

Dann wird auf der Konsole “V” ausgegeben. Das Programm mit Strg-C abbrechen.

Füllen wir alle print-Methoden:

```
void print(float value) {
    std::cout << value;
}
void println(float value) {
    std::cout << value << std::endl;
}
void print(const char ln[]) {
    std::cout << ln;
}
void println(const char ln[]) {
    std::cout << ln << std::endl;
}
```

Dann wird bei runtest permanent

```
-1MB = 5V | U = -0.00488281V
```

Ausgegeben. Das heisst wir sind hier

```
  else if (in <= 204)  //MB = 5V
  {
    pinMode(s1, INPUT);
    pinMode(s2, INPUT);
    while (analogRead(inputpin) < 1023)
    {
     Serial.print("MB = 5V | ");
     in = analogRead(inputpin);
     u = in * 5 / 1024;
     Serial.print("U = ");
     Serial.print(u);
     Serial.println("V");
     Serial.print(in);
     delay(1000); //damit nicht alles "durchrauscht"
    }
  }
```

Weil analogRead -1 zurückgibt.

Ein commit der Änderung am Simulator:

```
svn ci arduino-simu.h “add print output”
```

Um den Test beeinflussen zu können müssen wir bestimmen was analogRead zurückgibt. Im arduino-simu

```
int simu_analogIn;
int analogRead(int pin) {
   return simu_analogIn;
}
```

In test.cpp:

```
int main() {
   simu_analogIn = 100;
   loop();
```

Dann gibt runtest aus:

```
100MB = 5V | U = 0.488281V
```

Wenn wir da heraus wollen muss analogRead 1023 zurückgeben, beim vierten mal.

Im arduino-simu

```
int simu_analogIn;
int simu_cnt = 0;
int analogRead(int pin) {
   Simu_cnt++;
   If (simu_cnt >=3)
      return 1023;
   return simu_analogIn;
}
```

Dann hängt runtest nicht mehr in der Schleife sondern gibt nur folgendes aus:

```
100
MB = 5V | U = 4.99512V
1023
```

Das ist das was wir überprüfen müssen …

svn ci -m “analogRead simulation”

18:14 Pause

18:41

Wir können den MB /Messbereich erkennen weil das Programm, beide Pins auf INPUT setzt:

```
pinMode(s1, INPUT);
pinMode(s2, INPUT);
```

S1 und s2 sind aber nur die Nummern der Pins nicht die Werte. Das müssen wir erst einbauen. Arduino-simu:

```
int simu_pinMode[16];
void pinMode(int pin, int mode) {
   simu_pinMode[pin] = mode;
}
```

Und im test.cpp:

```
simu_pinMode[1] = 99;  // invalid
simu_analogIn = 100;
loop();
if (simu_pinMode[1] == 0) {
  std::cout << "OK" << std::endl;
  return 0;
}
std::cout << "ERROR" << std::endl;
```

runtest:

```
100
MB = 5V | U = 4.99512V
1023OK
```

Und wenn wir im arduino-simu auskommentieren:

```
int simu_pinMode[16];
void pinMode(int pin, int mode) {
//   simu_pinMode[pin] = mode;
}
```

runtest:

```
100
MB = 5V | U = 4.99512V
1023ERROR
```

Das heisst wir haben den Code besser verstehen gelernt.

svn ci (ohne die “//” im arduino-simu.h).

19:05 PAUSE bis ...

REFERENCES

1. https://en.wikipedia.org/wiki/Duplicate_code