Software-Entwicklungskosten

Einige Gedanken zur
nachträglichen Bewertung

Am Köllenholz 19, 86637 Wertingen

Hintergrund

- Im letzten Jahr erhielt ich gehäuft Anfragen zur nachträglichen Bewertung der Entwicklungskosten von Software.
- In der Regel sollte diese Software in ein Unternehmen eingebracht werden, anstelle von Stammkapital.
- Das Gutachten sollte zur Vorlage beim Finanzamt dienen.

Grundsätzliche Überlegungen

- Unangenehme Zusatzfragen.
- Was kann man bewerten?
- Was kann man nicht bewerten?
- Welche Faktoren beeinflussen das Ganze?

Unangenehme Zusatzfragen

- Vergleich: Welche „marktüblichen“ Kosten wären für die Software anzusetzen?
- Wie ist die Qualität des Sourcecodes (finanziell) zu bewerten?
- Hätte die Wahl einer anderen Programmiersprache / eines besseren Entwicklers, die Entwicklungskosten gesenkt?

Was kann man bewerten?

- Den durchschnittlichen Zeitaufwand für die Entwicklung durch einen geeigneten Entwickler.
- Die am Markt üblichen Stundensätze für einen geeigneten Entwickler (gulp.de).
- Daraus die durchschnittlichen Kosten für die Softwareentwicklung.

Was kann man nicht bewerten?

- Welchen Wert die Software am Markt besitzt (Zu welchem Preis in welchen Mengen lässt sie sich in einem bestimmten Zeitraum absetzen).

- Wie lange ein Entwickler tatsächlich gebraucht hat.

Beeinflussungsfaktoren

- Anzahl der Entwickler
- Projektleitung
- Software-Metriken (Beispiel: LOC)
- Programmiersprache
- Qualifikation der Entwickler

Wir beschränken uns in dieser Präsentation auf einen Entwickler, keine Projektleitung und ignorieren die Planungsphase.

Software-Metrik

- Man benötigt ein geeignetes Instrument, um anhand des Quellcodes zu erkennen, wie viele Mannstunden für den Code aufgewandt werden müssen.
- Es gibt verschiedene Verfahren, z.B. die Lines Of Code.
- Ein geeignetes Verfahren zu finden, ist ein Thema für sich.

Software-Metrik

- Lines Of Code seien nur als ein Beispiel genannt, um im Nachhinein den Entwicklungsaufwand abzuschätzen.
- Für diese Präsentation dienen uns LOC als visuelles Mittel, um die späteren Beispiele beurteilen zu können.
- Welche Metrik(en) die Beste ist, hängt oft davon ab, wie der Code vorliegt und ob es ein „Messwerkzeug“ dafür gibt.

- Die Programmiersprache

- Je einfacher, desto schlechter das Metrik-Ergebnis (idR).
- Je mächtiger, desto besser kann das Metrik-Ergebnis sein (Klassen, Templates).
- Je einfacher, desto günstiger die Kosten einer Entwicklerstunde.
- Ausnahme: Geringes Angebot an Entwicklern, z.B. bei Uralt-Software.
- Die üblichen Durchschnittssätze lassen sich z.B. über www.gulp.de abfragen.

- Der Entwickler

- Beherrscht der Entwickler eine mächtige Programmiersprache, verbessern sich Metrik-Ergebnis, Entwicklungszeit und Folgekosten.
- Je weniger er/sie die Programmiersprache beherrscht, desto schlechter fallen diese aus.
- Ein guter Entwickler in einer mächtigen Programmiersprache verliert seine Vorteile, je einfacher die Programmiersprache für sein Projekt ausfällt.

- Der Entwickler

- Ein guter objektorientierter Entwickler (z.B. C++) kann seine Vorteile mit C, Pascal, Cobol oder Scripting-Sprachen nicht ausspielen.

- Er oder sie muss sich mit „einfachen“ Entwicklern vergleichen lassen, wenn die Programmiersprache eingeschränkt ist.

Den Entwickler einschätzen

- Werden offensichtliche Vorteile einer Programmiersprache nicht genutzt?
- Wenn nein: Warum nicht?
  - Fehlendes Know-How
  - Entwicklungsvorgaben
  - Übernahme aus altem Code (z.B. C -> C++)
  - Falsche Prioritäten bei der Umsetzung

Beispiel 1: von C nach C++

```
m_pDB->SetSQL("Select AKTUSER, VARIANTE, STATUS, LOKAL, PRIVAT, ADRESSE, ART, STUFE, FIRMA, FIRMA2,
FIRMA3, ABTEILUNG, GEBAEUDE, BANREDE, ANREDE, BERUF, POSITION, TITEL, TITEL2, VORNAME, NAME, ZUSATZ,
LANDKENNUNG, LAND, STAAT, STRASSE, PLZ, ORT, STRZUSATZ, POSTFACH, PPLZ, PFORT, GPLZ, GKORT, DATUM1,
DATUM2, DATUM3, DATUM4, BANREDE2, ANREDE2, BERUF2, POSITION2, TITEL21, TITEL22, VORNAME2, NAME2,
ZUSATZ2, SEMAPHORE, INUSE, CREATEUSER, MODIFYUSER, CREATEDATE, MODIFYDATE, EXTLOCKS, KOMM1, KOMM2,
KOMM3, KOMM4, KOMM5, KOMM6, KOMM7, KOMM8, KOMM9, KOMM10, KOMM11, KOMM12, KOMM13, KOMM14, KOMM15,
KOMM16, KOMM17, KOMM18, KOMM19, KOMM20, KOMM21, KOMM22, KOMM23, KOMM24, KOMM25, SONSTIGES,
FILIALEVON, BANK1, KONTO1, BLZ1, BANK2, KONTO2, BLZ2, SKONTOF, SKONTOV, RABATTF, RABATTV, ZIELF,
ZIELV, IDNUMMERF, IDNUMMERV, ATTRNR1, ATTRNR2 ");
wsprintf(sqlbuffer, "FROM Adressen, adrkomm, adrbank, adrattr WHERE adressen.AdrNr = %ld and
adrkomm.adrnr = adressen.adrnr and adrbank.adrnr = adressen.adrnr and adrattr.adrnr =
adressen.adrnr", nummer);
m_pDB->AppendSQL(sqlbuffer);
if ((ret = ::SQLAllocStmt(m_pDB->m_hdbc, &hStmt)) != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO)
       return FALSE;
ret = ::SQLExecDirect(hStmt, (unsigned char*)m_pDB->GetSQL(), m_pDB->GetSQLLen());
if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO)
{
       ::SQLBindCol(hStmt,  1, SQL_C_LONG, &(m_pPointer->AKTUSER), 0, &len);
       ::SQLBindCol(hStmt,  2, SQL_C_LONG, &(m_pPointer->VARIANTE), 0, &len);
       ::SQLBindCol(hStmt,  3, SQL_C_LONG, &(m_pPointer->STATUS), 0, &len);
       ::SQLBindCol(hStmt,  4, SQL_C_LONG, &(m_pPointer->LOKAL), 0, &len);
       ::SQLBindCol(hStmt,  5, SQL_C_LONG, &(m_pPointer->PRIVAT), 0, &len);
       ::SQLBindCol(hStmt,  6, SQL_C_CHAR, m_pPointer->GetADRESSE(), TAD_ADRESSE_LEN+1, &len);
       ::SQLBindCol(hStmt,  7, SQL_C_LONG, &(m_pPointer->ART), 0, &len);
       ::SQLBindCol(hStmt,  8, SQL_C_LONG, &(m_pPointer->STUFE), 0, &len);
       ::SQLBindCol(hStmt,  9, SQL_C_CHAR, m_pPointer->GetFIRMA(), TAD_FIRMA_LEN+1, &len);
       ::SQLBindCol(hStmt, 10, SQL_C_CHAR, m_pPointer->GetFIRMA2(), TAD_FIRMA2_LEN+1, &len);
       ::SQLBindCol(hStmt, 11, SQL_C_CHAR, m_pPointer->GetFIRMA3(), TAD_FIRMA3_LEN+1, &len);
```

Beispiel 1: Verbesserung

```
int i = 1;
if ((ret = ::SQLAllocStmt(m_pDB->m_hdbc, &hStmt)) != SQL_SUCCESS && ret !=
SQL_SUCCESS_WITH_INFO)
       return FALSE;
ret = ::SQLExecDirect(hStmt, (unsigned char*)m_pDB->GetSQL(), m_pDB->GetSQLLen());
if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO)
{
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->AKTUSER), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->VARIANTE), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->STATUS), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->LOKAL), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->PRIVAT), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_CHAR, m_pPointer->GetADRESSE(), TAD_ADRESSE_LEN+1, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->ART), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->STUFE), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_CHAR, m_pPointer->GetFIRMA(), TAD_FIRMA_LEN+1, &len);
::SQLBindCol(hStmt, i++, SQL_C_CHAR, m_pPointer->GetFIRMA2(), TAD_FIRMA2_LEN+1, &len);
::SQLBindCol(hStmt, i++, SQL_C_CHAR, m_pPointer->GetFIRMA3(), TAD_FIRMA3_LEN+1, &len);

                                        :
                                        :
```

Einschätzung: Erstes Fazit

- Die Denkweise lässt das Vermissen eines übergreifenden Verständnisses für den gesamten C++-Code erwarten. Beispiele:
  - ++ kommt in jeder For-Schleife vor!
  - Char statt CString, mit allen Konsequenzen.
  - SQLBindCol: Loslegen statt Planen.
- Wie sieht es dann erst mit Fehler-behandlung, Stabilität etc. aus?

Beispiel 2: Muster erkennen

```
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->AKTUSER), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->VARIANTE), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->STATUS), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->LOKAL), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->PRIVAT), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_CHAR, m_pPointer->GetADRESSE(), TAD_ADRESSE_LEN+1, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->ART), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_LONG, &(m_pPointer->STUFE), 0, &len);
::SQLBindCol(hStmt, i++, SQL_C_CHAR, m_pPointer->GetFIRMA(), TAD_FIRMA_LEN+1, &len);
::SQLBindCol(hStmt, i++, SQL_C_CHAR, m_pPointer->GetFIRMA2(), TAD_FIRMA2_LEN+1, &len);
::SQLBindCol(hStmt, i++, SQL_C_CHAR, m_pPointer->GetFIRMA3(), TAD_FIRMA3_LEN+1, &len);
::SQLBindCol(hStmt, i++, SQL_C_CHAR, m_pPointer->GetABTEILUNG(), TAD_ABTEILUNG_LEN+1, &len);
::SQLBindCol(hStmt, i++, SQL_C_CHAR, m_pPointer->GetGEBAEUDE(), TAD_GEBAEUDE_LEN+1, &len);
```

```
strcpy(sqlbuffer,"INSERT INTO DARSTELLUNG
(USERNR,LFHEIGHT,LFWIDTH,LFESCAPEMENT,LFORIENTATION,LFWEIGHT,LFITALIC,LFUNDERLINE,LFSTRIKEOUT,LFCHARSET,LFOUTPRECISION,LFCLIPPRECISION,LFQUALITY,
LFPITCHANDFAMILY,LFFACENAME,LFCOLOR,BFHEIGHT,BFWIDTH,BFESCAPEMENT,BFORIENTATION,BFWEIGHT,BFITALIC,BFUNDERLINE,BFSTRIKEOUT,BFCHARSET,BFOUTPRECISIO
N,BFCLIPPRECISION,BFQUALITY,BFPITCHANDFAMILY,BFFACENAME,BFCOLOR,CREATEUSER,CREATEDATE,MODIFYUSER,MODIFYDATE,DNR,BEZ,BLINKS,BRECHTS,BREITELINKS,SE
MAPHORE,INUSE,FELDNR1,FELDNR2,FELDNR3,FELDNR4,FELDNR5,FELDNR6,FELDNR7,FELDNR8,FELDNR9,FELDNR10,FELDNR11,FELDNR12,FELDNR13,FELDNR14,FELDNR15,FELDN
R16,FELDNR17,FELDNR18,FELDNR19,FELDNR20,FELDNR21,FELDNR22,FELDNR23,FELDNR24,FELDNR25,BREITE1,BREITE2,BREITE3,BREITE4,BREITE5,BREITE6,BREITE7,BREI
TE8,BREITE9,BREITE10,BREITE11,BREITE12,BREITE13,BREITE14,BREITE15,BREITE16,BREITE17,BREITE18,BREITE19,BREITE20,BREITE21,BREITE22,BREITE23,BREITE2
4,BREITE25,MAKEGLOBAL,FENSTERTEILER) VALUES(");

temp.Format("%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,'%s',%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,'%s',%d,'%s','%s','%s','%s',%d,'%s','%s','%
s',%d,'%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s',
%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d)",m_pPointer->QGetUSERNR(),m_pPointer->QGetLFHEIGHT(),
      m_pPointer->QGetLFWIDTH(),m_pPointer->QGetLFESCAPEMENT(),m_pPointer->QGetLFORIENTATION(),m_pPointer->QGetLFWEIGHT(),m_pPointer-
>QGetLFITALIC(),m_pPointer->QGetLFUNDERLINE(),      m_pPointer->QGetLFSTRIKEOUT(),m_pPointer->QGetLFCHARSET(),m_pPointer->QGetLFOUTPRECISION(),
      m_pPointer->QGetLFCLIPPRECISION(),m_pPointer->QGetLFQUALITY(),m_pPointer->QGetLFPITCHANDFAMILY(),m_pPointer->QGetLFFACENAME(),m_pPointer-
>QGetLFCOLOR(),m_pPointer->QGetBLFHEIGHT(),m_pPointer->QGetBLFWIDTH(),m_pPointer->QGetBLFESCAPEMENT(),m_pPointer-
>QGetBLFORIENTATION(),m_pPointer->QGetBLFWEIGHT(),m_pPointer->QGetBLFITALIC(),m_pPointer->QGetBLFUNDERLINE(),  m_pPointer-
>QGetBLFSTRIKEOUT(),m_pPointer->QGetBLFCHARSET(),m_pPointer->QGetBLFOUTPRECISION(),  m_pPointer->QGetBLFCLIPPRECISION(),m_pPointer-
>QGetBLFQUALITY(),m_pPointer->QGetBLFPITCHANDFAMILY(),m_pPointer->QGetBLFFACENAME(),m_pPointer->QGetBLFCOLOR(),m_pPointer-
>QGetCREATEUSER(),m_pPointer->QGetCREATEDATE(),m_pPointer->QGetMODIFYUSER(),m_pPointer->QGetMODIFYDATE(),m_pPointer->QGetNR(),m_pPointer-
>QGetBEZ(),m_pPointer->QGetBLINKS(),m_pPointer->QGetBRECHTS(),m_pPointer->QGetBREITELINKS(),m_pPointer->QGetSEMAPHORE(),m_pPointer->QGetINUSE(),
            m_pPointer->QGetFELDNR1(), m_pPointer->QGetFELDNR2(), m_pPointer->QGetFELDNR3(), m_pPointer->QGetFELDNR4(), m_pPointer-
>QGetFELDNR5(), m_pPointer->QGetFELDNR6(), m_pPointer->QGetFELDNR7(), m_pPointer->QGetFELDNR8(), m_pPointer->QGetFELDNR9(), m_pPointer-
>QGetFELDNR10(), m_pPointer->QGetFELDNR11(), m_pPointer->QGetFELDNR12(), m_pPointer->QGetFELDNR13(), m_pPointer->QGetFELDNR14(), m_pPointer-
>QGetFELDNR15(), m_pPointer->QGetFELDNR16(), m_pPointer->QGetFELDNR17(), m_pPointer->QGetFELDNR18(), m_pPointer->QGetFELDNR19(), m_pPointer-
>QGetFELDNR20(), m_pPointer->QGetFELDNR21(), m_pPointer->QGetFELDNR22(), m_pPointer->QGetFELDNR23(), m_pPointer->QGetFELDNR24(), m_pPointer-
>QGetFELDNR25(), m_pPointer->QGetBREITE1(), m_pPointer->QGetBREITE2(), m_pPointer->QGetBREITE3(), m_pPointer->QGetBREITE4(), m_pPointer-
>QGetBREITE5(), m_pPointer->QGetBREITE6(), m_pPointer->QGetBREITE7(), m_pPointer->QGetBREITE8(), m_pPointer->QGetBREITE9(), m_pPointer-
>QGetBREITE10(), m_pPointer->QGetBREITE11(), m_pPointer->QGetBREITE12(), m_pPointer->QGetBREITE13(), m_pPointer->QGetBREITE14(), m_pPointer-
>QGetBREITE15(), m_pPointer->QGetBREITE16(), m_pPointer->QGetBREITE17(), m_pPointer->QGetBREITE18(), m_pPointer->QGetBREITE19(), m_pPointer-
>QGetBREITE20(), m_pPointer->QGetBREITE21(), m_pPointer->QGetBREITE22(), m_pPointer->QGetBREITE23(), m_pPointer->QGetBREITE24(), m_pPointer-
>QGetBREITE25(), m_pPointer->QGetMAKEGLOBAL(), m_pPointer->QGetFENSTERTEILER() );
```

Beispiel 2: Muster erkennen

- Jede Tabelle wird von Hand kodiert an das entsprechendem Objekt gebunden.
- Die SQL-Befehle Select, Insert und Update werden ebenfalls pro Tabelle auskodiert.
- Die Folge:
  - Sehr hohes Fehlerpotential
  - Schlechtes Metrik-Ergebnis
  - Versteckte Kosten auch nach der Entwicklung

Beispiel 2: Verbesserung

```
FELD.Add("ADRNR");                  POINTER.Add(&ADRNR);                  LAENGE.Add(TAD_ADRNR_LEN);
FELD.Add("STUFE");                  POINTER.Add(&STUFE);                  LAENGE.Add(TAD_STUFE_LEN);
FELD.Add("VARIANTE");               POINTER.Add(&VARIANTE);               LAENGE.Add(TAD_VARIANTE_LEN);
FELD.Add("ART");                    POINTER.Add(&ART);                    LAENGE.Add(TAD_ART_LEN);
FELD.Add("AKTUSER");                POINTER.Add(&AKTUSER);                LAENGE.Add(TAD_AKTUSER_LEN);
FELD.Add("STATUS");                 POINTER.Add(&STATUS);                 LAENGE.Add(TAD_STATUS_LEN);
FELD.Add("LOKAL");                  POINTER.Add(&LOKAL);                  LAENGE.Add(TAD_LOKAL_LEN);
FELD.Add("PRIVAT");                 POINTER.Add(&PRIVAT);                 LAENGE.Add(TAD_PRIVAT_LEN);
FELD.Add("AMT");                    POINTER.Add(&AMT);                    LAENGE.Add(TAD_AMT_LEN);
FELD.Add("SKONTOF");                POINTER.Add(&SKONTOGEWAEHRT);         LAENGE.Add(TAD_SKONTOGEWAEHRT_LEN);
FELD.Add("SKONTOV");                POINTER.Add(&SKONTOERHALTEN);         LAENGE.Add(TAD_SKONTOERHALTEN_LEN);
FELD.Add("RABATTF");                POINTER.Add(&RABATTGEWAEHRT);         LAENGE.Add(TAD_RABATTGEWAEHRT_LEN);
FELD.Add("RABATTV");                POINTER.Add(&RABATTERHALTEN);         LAENGE.Add(TAD_RABATTERHALTEN_LEN);
FELD.Add("ZIELF");                  POINTER.Add(&ZIELGEWAEHRT);           LAENGE.Add(TAD_ZIELGEWAEHRT_LEN);
FELD.Add("ZIELV");                  POINTER.Add(&ZIELERHALTEN);           LAENGE.Add(TAD_ZIELERHALTEN_LEN);
```

```
void CAdrCommonObj::Bind()
{
       for(i=0; i<=FELD.GetUpperBound(); i++)
       {
          if(LAENGE.GetAt(i) > 0)
              ::SQLBindCol(hStmt, i+1, SQL_C_CHAR, ((CString*)POINTER.GetAt(i))->GetBufferSetLength(LAENGE.GetAt(i)+1), LAENGE.GetAt(i)+1, &len);
          else
              ::SQLBindCol(hStmt, i+1, SQL_C_LONG, (unsigned long*)POINTER.GetAt(i), 0, &len);
       }
}
```

```
CString CAdrCommonObj::PrepAddSQL(long NR)
{
       //Die Values müssen erzeugt und angefügt werden
       CString tmp;
       temp = templ = "";
       SetNR(NR);
       for(i=0; i<=FELD.GetUpperBound(); i++)
       {
              if(LAENGE.GetAt(i) > 0)
              {
                     tmp = qtrim_it(*(CString*)(POINTER.GetAt(i)));
                     templ = templ +", '" + tmp.GetBufferSetLength(tmp.GetLength()) + "'";
              }
              else
              {
                     tmp.Format("%ld", *(unsigned long*)POINTER.GetAt(i));
                     templ = templ + ", " + tmp.GetBufferSetLength(tmp.GetLength());
              }
       }
       temp.Format(m_AddSQL, templ);
       temp.Replace("(, ", "(");

       return temp;
}
```

Beispiel 2: Verbesserung

- Jede Tabelle wird über Arrays definiert.
- Jedes Tabellenobjekt erbt die nur einmal vorhandene Funktionalität zur automatischen Bindung und Erzeugung von SQL-Befehlen.
- Die Folge:
  - Kaum Fehlerpotential
  - Gutes Metrik-Ergebnis
  - Keine versteckten Kosten

Beurteilung der Beispiele

- Beispiel 1 zeigt C++-Programmierung auf C/Pascal/Cobol-Niveau.
- Das Potential der Sprache wurde nicht nur im Bezug auf die Vererbung verschenkt.
- Jeder gute Entwickler hätte auch mit C oder Pascal eine Vereinfachung angestrebt.

Konsequenz für die Kosten

- Die Software sollte trotz C++ nicht mit dem Stundensatz von guten C++-Entwicklern bewertet werden.
- Trotz geringerer Stundensätze erhöht sich die Entwicklungszeit. Die Entwicklungs-kosten werden nicht unbedingt gesenkt.
- Die Wartungs- und Folgekosten für Fehler-behebung werden höher ausfallen.

Konsequenz für die Bewertung

- Ein guter Entwickler spart Zeit während der Entwicklung, sowie Folgekosten für Fehlerbehebung und Wartung.
- Für die Beurteilung der Entwicklungs-kosten im Nachhinein wird jedoch oft nur der Ist-Zustand vorgegeben.
- Trotz schlechterer Metrik-Ergebnisse werden Entwicklungskosten durch einen günstigeren Stundensatz angeglichen.

Möglichkeiten der Ist-Bewertung

- A. Keine Unterscheidung beim Entwickler
  - Schlechte Entwickler erzeugen teueren Code, da schlechteres Metrik-Ergebnis (ME).
  - Schlechter Code erhöht Stammkapital!?
- B. Unterscheidung beim Entwickler
  - Weniger Unterschied in den Kosten.
  - Schlechteres ME x geringerer Stundensatz ~ besseres ME x höherer Stundensatz.

Welche Folgen?

- Bei A: Nur im Vergleich mit einer Neuentwicklung durch guten Entwickler erkennt man den Kostennachteil von schlechter Software.

- Bei B: Nur unter Berücksichtigung von Entwicklungszeit und Folgekosten erkennt man den Nachteil von schlechter Software.

Wozu das Ganze?

- Erst wenn man zu den Entwicklungs-kosten den Zeitgewinn und die Folge-kosten hinzu nimmt, offenbart sich ein Qualitätsunterschied in Euro.
- Ohne Berücksichtigung der Qualität muss man sein Gutachten bei nachträglicher, erweiterter Betrachtung durch Dritte diesbezüglich u.U. rechtfertigen.

Offene Fragen

- Soll man diese Unterscheidungen immer treffen oder nur, wenn die Entwicklungs-kosten im Zusammenhang mit den Kosten einer Neuentwicklung zu sehen sind?
- Hat die Qualität überhaupt eine Bedeutung für die Entwicklungskosten oder nur die Metrik?

Mein Standpunkt

- Ich pflege das CYA-Prinzip:
  - Was geschieht, wenn z.B. das Finanzamt die Entwicklungskosten im Gutachten als zu hoch einschätzt?
  - Dann möchte ich zweifelsfrei festgehalten haben, dass die Entwicklungskosten im Gutachten nicht notwendigerweise denen einer Neuentwicklung unter optimalen Bedingungen entsprechen müssen!