



Dipl.-Inf. (FH) Thomas Noone

öbuv Sachverständiger für Systeme und Anwendungen der Informationsverarbeitung



Beruflicher Hintergrund

- Geschäftsführer eines Software-Entwicklungsunternehmens.
- Schwerpunkt Adressen-Management, CRM.
- Kunden: ABB bis ZF Friedrichshafen.
- In Berlin: u.a. die OECD, Deutsche Bank, Bundesarchitektenkammer, Oxfam.



Heutiger Hintergrund

- Vortrag der Fachgruppe „Elektronik und EDV“ in Oberursel.
- Thema: Gedanken zur nachträglichen Bewertung von Software.



Software-Metrik

- Man benötigt ein geeignetes Instrument, um anhand des Quellcodes zu erkennen, wie viele Mannstunden für den Code aufgewandt wurden.
- Es gibt verschiedene Verfahren, z.B. die Lines Of Code.



Erkenntnis

- Ein guter Code erzeugt weniger Zeilen.
- Gemessen mit LOC benötigt dieser Code daher weniger Mannstunden.



Beispiel: Muster erkennen

- Jede Tabelle wird von Hand kodiert an das entsprechende 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: Verbesserung

```
FELD.Add("ADRNR");
FELD.Add("STUFE");
FELD.Add("VARIANTE");
FELD.Add("ART");
FELD.Add("AKTUSER");
FELD.Add("STATUS");
FELD.Add("LOKAL");
FELD.Add("PRIVAT");
FELD.Add("AMT");
FELD.Add("SKONTOF");
FELD.Add("SKONTOV");
FELD.Add("RABATTF");
FELD.Add("RABATTV");
FELD.Add("ZIELF");
FELD.Add("ZIELV");

POINTER.Add(&ADRNR);
POINTER.Add(&STUFE);
POINTER.Add(&VARIANTE);
POINTER.Add(&ART);
POINTER.Add(&AKTUSER);
POINTER.Add(&STATUS);
POINTER.Add(&LOKAL);
POINTER.Add(&PRIVAT);
POINTER.Add(&AMT);
POINTER.Add(&SKONTOGEWAHRT);
POINTER.Add(&SKONTOERHALTEN);
POINTER.Add(&RABATTGEWAHRT);
POINTER.Add(&RABATTERHALTEN);
POINTER.Add(&ZIELGEWAHRT);
POINTER.Add(&ZIELERHALTEN);

LAENGE.Add(TAD_ADRNR_LEN);
LAENGE.Add(TAD_STUFE_LEN);
LAENGE.Add(TAD_VARIANTE_LEN);
LAENGE.Add(TAD_ART_LEN);
LAENGE.Add(TAD_AKTUSER_LEN);
LAENGE.Add(TAD_STATUS_LEN);
LAENGE.Add(TAD_LOKAL_LEN);
LAENGE.Add(TAD_PRIVAT_LEN);
LAENGE.Add(TAD_AMT_LEN);
LAENGE.Add(TAD_SKONTOGEWAHRT_LEN);
LAENGE.Add(TAD_SKONTOERHALTEN_LEN);
LAENGE.Add(TAD_RABATTGEWAHRT_LEN);
LAENGE.Add(TAD_RABATTERHALTEN_LEN);
LAENGE.Add(TAD_ZIELGEWAHRT_LEN);
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: 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



Konsequenzen der Bewertung

- Schlechte Entwickler erzeugen teureren Code, da schlechteres Metrik-Ergebnis (ME).
- Dieses Ergebnis soll oft als Basis für eine Entlohnung oder Stammeinlage dienen.
- Schlechter Code erhöht Stammeinlage!



Vorliegendes Problem

- Fakten, soweit mir bekannt:
 - Für einige Softwarekomponenten soll mit LOC belegt werden, dass diese im geplanten Zeitrahmen auch umsetzbar waren.
- Probleme:
 - Die Annahmen (z.B. bezüglich der \emptyset Anzahl Zeilen/Stunde) können angezweifelt werden.
 - Andere Metriken könnten ein negativeres Ergebnis aufzeigen.



Ziel (soweit bekannt)

- Die durch LOC ermittelten Ergebnisse abzusichern.



Erste Gedankenansätze

- LOC sagen nichts darüber aus, wieviel von dem Know-how bereits zur Verfügung stand, welches üblicherweise erst durch „Forschung und Entwicklung“ erarbeitet werden muss.
- Alle LOC werden mit einem „Zeitfaktor“ verrechnet, basierend auf einer Annahme, wieviele Zeilen Code im Schnitt pro Stunde/Manntag geleistet werden.



Erste Gedankenansätze

- LOC als Messverfahren läßt sich nicht ändern.
- Deshalb:
 - Trotz bzw. wegen LOC als Metrik nach Wegen suchen, um „die Braut schöner zu machen“.
- Wie:
 - Die Nachteile des Verfahrens ausnutzen.



1. Schritt

- Aus Lastenheft Entwicklungsbereiche herausarbeiten, mit viel LOCs aber auch mit viel mitgebrachtem Know-how (fertiger Code, Erfahrung, Generatoren etc.).
- Diese Bereiche mit „schnellerem Zeitfaktor“ versehen.



1. Schritt: Ergebnis

- LOC-Ergebnis wird besser.
 - Andere Metriken werden ebenfalls besser.
 - Die Verbesserung basiert auf belegbaren internen Faktoren, die außenstehende Gegengutachter nicht einsehen können.
- ⇒ Wertet deren Sorgfalt im Nachhinein ab.
- ⇒ Verlangt Interviews mit Entwicklern.



2. Schritt

- Alternative Metriken auf ihre Eignung bzw. Nichteignung prüfen.
- Mögliche Folge für Gegengutachten: Diese verwenden u.U. Metriken, die als ungeeignet belegt werden können.



3. Schritt

- Mit geeigneten Metriken Ergebnis des 1. Schritts neu beleuchten.
- Positives Ergebnis: Untermauert LOC-gestützten Standpunkt.
- Negatives Ergebnis:
 - Ist durch 1. Schritt weniger negativ.
 - Mittelwert mit LOC bilden und damit Objektivität und Selbstkritik zeigen.
 - Man kommt Kritikern mit besseren Werten zuvor.