Inhalt

0 Einleitung 2

1 Struktur der Aufgabenanalyse 2

2 HIPO: Hierarchy Input Processing Output 3

3 Was hat das mit API zu tun? 4

4 Layer 5

0 Einleitung

Unser Diskussion am 06.07.2007 in der 4D Werkstatt hat mich wieder einmal auf einen Gedankenpfad aufmerksam gemacht, den ich seit längerem verfolge, immer wieder konsequent anwenden wollte, aber aus Bequemlichkeit immer wieder beiseite gelegt habe.

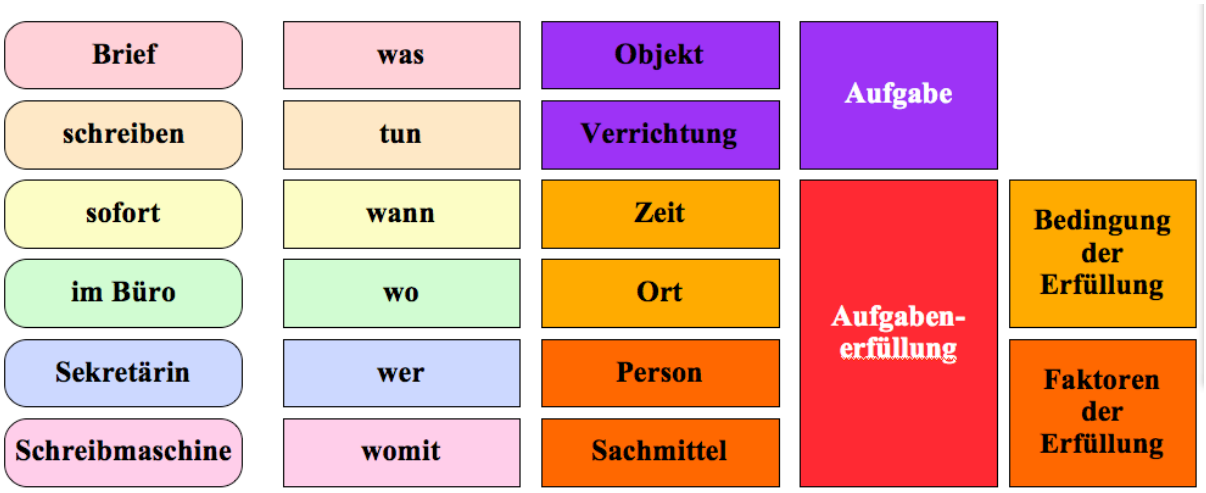
Mein Gedankengang in der Übersicht:

- Ich betrachte Arbeitsabläufe durch meinen beruflichen Werdegang aus Sicht des Organisators bzw. Organisationsanalytikers
- Dort gibt es seit längerem bewährte Verfahren der Organisationsanalyse
- Unsere Diskussion über API hat mich daran erinnert, dass eine solche Methode bzw. Funktion eigentlich nichts andres ist, als die strukturelle Shell eines Arbeitsprozesses
- Wieso also nicht die Regeln der Organisationsanalyse auf die API anwenden?

Deswegen hier mal kurz und prägnant meine Idee den Kollegen zum Verriss dargeboten ;-)

1 Struktur der Aufgabenanalyse

Eine (neben anderen) aus meiner Sicht bewährte Struktur der Aufgabenanalyse sieht folgendermaßen aus (das ist ein nicht aktualisiertes Bildchen aus einem meiner Standardseminare):



Erläuterung:

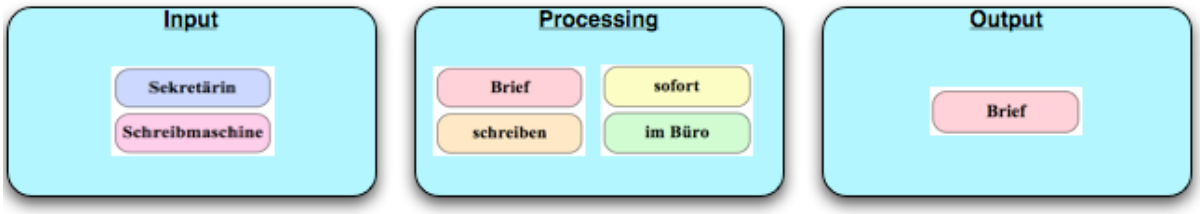
- Eine Aufgabe erfordert IMMER mindestens die Beschreibung des Objektes und der Verrichtung (Ortwin hat das als Subjekt und Verb beschrieben)
- Meistens muss noch die Art der Aufgabenerfüllung beschrieben werden. Dabei unterscheidet man zwischen
  - Bedingungen für die Aufgabenerfüllung, das sind Zeit und Ort (wobei Zeit sowohl einen Zeitpunkt, als auch eine Zeitdauer betreffen kann) und den
  - Faktoren für die Aufgabenerfüllung, also wer erledigt die Aufgabe mit welchen Sachmitteln

Ich will das zunächst mal einfach so stehen lassen und mit einem weiteren Strukturaspekt von Aufgaben befassen, der diesmal aus der Systemanalyse kommt:

2 HIPO: Hierarchy Input Processing Output

Das Teilchen ist zwar uralt und inzwischen in Theorie (und Praxis?) von anderen Methoden (ERM, EMK, UML) abgelöst. Aus meiner Sicht hat HIPO einen entscheidenden Vorteil: einfach, übersichtlich, allgemein verständlich und problemlos auch außerhalb der Softwareentwicklung und Systemdesign einsetzbar.

Das Prinzip ist ganz einfach (angewandt auf mein obiges Beispiel der Aufgabenanalyse):



„Hierarchy“ bedeutet in diesem Zusammenhang: Ich analysiere rückwärts, also ich suche den erforderlichen Output, analysiere, welche Prozesse für die Erzeugung des Outputs notwendig sind und daraus ergeben sich die Inputs. Das ist eine für die Analyse wichtige Vorgehensweise (in der Praxis wird oft in genau umgekehrter Reihenfolge gearbeitet: ich sammle erst mal Informationen, überlege was ich damit machen kann und erzeuge dann einen Output, den vielleicht keiner braucht).

3 Was hat das mit API zu tun?

Wenn ich die beiden Theorien zusammen packe, dann sehe ich eine Möglichkeit zu einer strukturell einheitlichen Beschreibung der Parametrisierung eines API. Nun bin ich mal gespannt, ob sich meine – aus der Sicht des Betriebswirts und Organisators kommende – Vorstellung mit dem deckt, was aus Programmierersicht definiert wurde oder wird.

Ich würde folgendes API strukturieren:

Errorcode:=API(Aufgabe{;Objekt;Verrichtung{;Inputs{;Bedingungen}}})

Dabei ist:

- Errorcode die Fehlernummer (das habe ich gestern gelernt)
- Aufgabe: der auszuführende Job (am Beispiel ein Begriff für „Brief schreiben“)
- Objekt: Das zu erstellende Objekt (Pointer?)
- Verrichtung: die konkrete Tätigkeit (Pointer?)
- Inputs: Pointer auf die Eingangsdaten (Werte, Variablen, etc.)
- Bedingungen: zeitliche, örtliche oder sonstige Rahmenbedingung für die Aufgabe

Wichtig sind die optionalen Parameter: Aufgabe muss immer, Objekt und Verrichtung kann, Inputs und Bedingungen ebenfalls.

Konkretes Beispiel:

$vt_Was:=“Erzeuge PDF“
$vz_Time:=“2007-07-07T18:00:00“
$vl_Error:=XX_handler(„Korrespondenz“;->Brief;->$vt_Was;->[Tabelle]Blob;->$vz_Time)

„Brief“ wäre dann ein PDF-Dokument. In [Tabelle]Blob könnte z.B. ein Write-Dokument oder sonst was stehen.

Zu bedenken ist dabei ergänzend Folgendes:

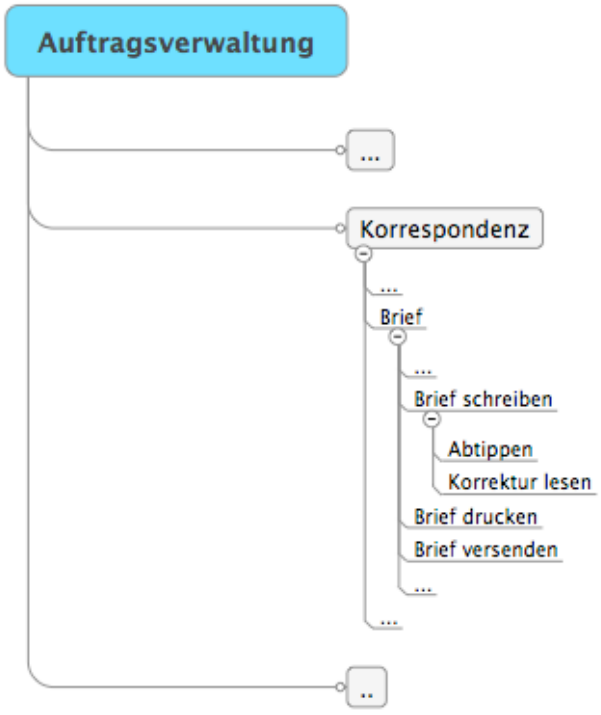
- Die Parameter Inputs und Bedingungen müssen Arrays sein (zumindest muss in geeigneter Form die Möglichkeit geschaffen werden, Mehrfacheinträge als Parameter zu übergeben
- Die Parameter können in sich eine (hierarchische) Struktur haben, und damit komme ich zu den diskutierten „Layers“

4 Layer

Die jeweiligen Ebenen (Layer) können bei den Betrachtungen eine entscheidende Rolle spielen:

- Das Objekt kann z.B. ein einzelner Brief, ein ganzer Satz Serienbriefe oder in anderen Aufgaben nur ein Tabellenfeld, ein Datensatz oder eine ganze Tabelle sein.
- Auch die Inputs und Bedingungen können mehrere „Layer“ haben: z.B. eine einzelne konkrete Sekretärin, ein Schreibbüro oder ganz abstrakt irgend eine beliebige Stelle

Die Layer lassen sich aus meiner Sicht nicht allgemeingültig definieren. Möglicherweise muss man einen Weg finden, die jeweilige Hierarchie durch eigene Parameter oder (einfacher) durch entsprechende Definitionen der Aufgabe darzustellen. Dieses scheint mir auch plausibler zu sein:



Ich will es nicht auf die Spitze treiben, aber das Beispiel zeigt, dass ich das selbe API mit unterschiedlichen „Aufgaben“ in einer unterschiedlichen Hierarchiestufe (Layer) bedienen kann.

Dabei hängt es aus meiner Sicht nur vom konkreten Fall ab, ob der Layer horizontal, vertikal oder sonst wie gegliedert ist. Das hängt nur von dem ab, was der Autor des API jeweils anbietet und dieses muss wiederum von den Anforderungen des „Kunden“ bestimmt werden.