LotusScript:
Anti-Patterns & Best-Practices

Dipl. Inform. Gerd Hildebrandt
IT-Consulting
Http://www.gehil.de

Hintergrund

- Mein Arbeits-Schwerpunkt ist die Architektur und Implementierung von komplexen Lotus Notes & Domino Anwendungen
- Ich entwickle seit 19 Jahren Lotus Notes & Domino Anwendungen
- Ich hatte mehrmals die Ehre Notes-Anwendungen konsolidieren zu dürfen, an denen sich zuvor dutzende Entwickler mit sehr 'interessanten' Ideen ausgetobt hatten
- Ich betreue seit Jahren eine Notes-Anwendung (aus vielen NSF's bestehend) mit insgesamt über 250.000 Zeilen LotusScript-Code
- Diese Erfahrungen haben meine Vorgehensweisen bei der LotusScript-Programmierung nachhaltig geprägt

Ausgangssituation

- Unternehmens-Software muss besonderen Anforderung genügen, weil diese:
  - firmen-intern von hunderten (tausenden) Mitarbeitern verwendet wird
  - 24 * 7 Stunden in der Woche im Betrieb ist
  - viele Jahre (Jahrzehnte) lang genutzt wird
  - in dieser Zeit dutzende bis hunderte Male angepasst und erweitert wird
  - zugleich oder im Laufe der zeit unterschiedliche Enwickler daran Arbeiten
- Es gibt viele Strategien und Disziplinen, diesen besonderen Anforderungen gerecht zu werden, ich kenne nur einige davon. Meine Vorgehensweise ist hauptsächlich beeinflusst vom 'Software-Engineering'.
- Im folgenden Abschnitt möchte ich einige Vorgehensweisen aufzeigen, die nach meiner Erfahrung nach nicht mit den Anforderungen an die Entwicklung von Unternehmens-Software vereinbar sind

Motivation: LotusScript Codequalität

- der LotusScript-Code ist von entscheidender Bedeutung für Funktionalität und Qualität einer komplexen Notes-Anwendung (ausser bei reinen Xpages-Anwendungen)
- die Qualität der LotusScript-Codierung hat grossen Einfluss auf die langfristigen Kosten von Wartung und Weiterentwicklung jeder komplexen Notes-Anwendung
- einige zentrale Qualitäts-Merkmale für SourceCode sind:

  - Wartbarkeit (Änderungen sind kostengünstig durchführbarbar)
  - Klarheit (Struktur & Ablauf sind ohne vorhergehende 'Tiefenanalyse' erkennbar)
  - Robustheit (Error-Handling, konfigurierbar, geringe System-Abhängigkeit)
  - Teamfähigkeit (verbreitete Standard-Vorgehensweisen, keine exotischen Tricks)

Teil 1
LotusScript Anti-Patterns

- es gibt unzählige Möglichkeiten, Codierungs-Regeln festzulegen. Bei manchen Unternehmen existieren Programmier-Guidelines von dutzenden Seiten Länge
- nach meiner Erfahrung behindert eine starke Regulierung der Programmierung die Produktivität und auch den Spaß von Entwicklern und Teams
- ich möchte einige verbreitete Vorgehensweisen aufzeigen, die es wegen negativer Konsequenzen für die Code-Qualität zu Vermeiden gilt - sogenannte 'Anti-Patterns'
- die Liste der hier gezeigten Anti-Patterns erhebt keinen Anspruch auf Vollständigkeit, ich zeige hier einige meiner persönlichen 'Spitzenreiter'

LotusScript Anti-Patterns:
1. On Error Resume Next

- mein persönlicher Top-Spitzenreiter - ich nenne dies 'LMAA ErrorHandling'
- verhindert die Kommunikation von Fehlern im LotusScript-Laufzeitsystem
- kann zu folgenreichen Daten-Fehlern führen, wenn z.B. in einer Routine Zwischenwerte (nicht erfolgreich) berechnet und danach mit doc.Save() ungültige Feld-Werte gespeichert werden
- es gibt auch einige sinnvolle Anwendungsfälle für diesen Befehl:

- im ErrorHandler selbst
- für einzelne Befehlszeilen (z.B. GetDocumentByUNID)

LotusScript Anti-Patterns:
2. Option Public in ScriptLibraries

- 'Option Public' in einer ScriptLibrary führt dazu, daß alle unter 'Declarations' angegebenen Variablen und alle Routinen der ScriptLibrary nach aussen hin sichtbar werden (ausgenommen als 'Private' deklarierte Variablen / Routinen)
- dies kann zu schwer auffindbaren Fehlern führen, weil jede Routine welche die ScriptLibrary lädt alle in der ScriptLib deklarierten Variablen verändern kann
- beim Debuggen von Routinen welche mehrere ScriptLibs laden werden im Debugger-Bereich 'Variablen' alle Variablen aller ScriptLibs mit angezeigt → sehr unübersichtlich
- ohne 'Option Public' werden alle Modul-Variablen und Routinen auf den Sichbarkeitsbereich innerhalb der Scriptlibrary eingeschränkt (ausgenommen als 'Public' deklarierte)
- → 'Option Public' in LotusScript-Libs grundsätzlich vermeiden - stattdessen die Sichtbarkeit jeder Modul-Variable / Routine einzeln festlegen

LotusScript Anti-Patterns:
2. Option Public in ScriptLibraries

- **'Option Public' ist nur die Spitze des Eisbergs:**

eine der wirksamsten Vorgehensweisen, um komplexe Software nachvollziehbar und wiederverwendbar zu gestalten ist die Sichtbarkeit von Namen (Variablen, Klassen, Typen und Routinen) einzuschränken

- Es ist hilfreich, den Sichtbarkeitsbereich von Variablen möglichst eng einzuschränken (Data-Hiding)
- Data-Hiding ist kein 'theoretischer Spinnkram' sondern erhöht die Code-Qualität wirksam und nachweisbar
- → Best Practices

LotusScript Anti-Patterns:
3. (unnötige) Deklaration als Variant

- die Variable ist dadurch vom Typ 'keineAhnung'
- dadurch wird die automatische Typ-Prüfung durch den Compiler verhindert
- Es gibt nur wenige triftige Gründe, eine Variable als 'Variant' zu deklarieren
- → Best Practices
- oftmals wird Variant aus Unkenntnis oder Faulheit verwendet
- beliebte Falle 'Multi-Deklaration':

```
Dim a,b,c As String
```

→ nur c ist ein String, Variablen a und b sind vom Typ 'Variant' !!

LotusScript Anti-Patterns:
4. kein 'Option Declare' / 'Option Explicit'

- Tatbestand 'unterlassene Hilfeleistung' – keine Hinweise des Compilers auf nicht-deklarierte Variablen
- Folge: der Compiler 'errät' den Typ von nicht-deklarierten Variablen anhand der ersten Verwendung im Code
- Folge: nachfolgende Programmierer rätseln: 'was geht denn hier ab ?'
- zentrale Vorgabe im Domino Designer aktivieren:
  Vorgaben/Domino Designer/LotusScript Editor/'Option Declare' immer einfügen
- Dies hilft aber nicht bei bereits existierendem Code -> nachpflegen !

LotusScript-Antipatterns:
5. zeitgesteuerte Agenten ohne Error-Handling

- genauso unbrauchbar: Error-Handling per 'Print'-Befehl in das Server-Log
- je nach Logging-Konfiguration gehen wichtige Error-Informationen in der Flut von routinemässigen Server-Ausgaben verloren
- Administratoren interessieren sich zumeist nicht für 'Programmierer-Probleme'
- dies verhindert effektiv die Qualitätssicherung von Background-Agenten, weil komplexe Hintergrund-Prozesse nicht 'vorab' fehlerfrei gemacht werden können → erst im Betrieb erkennt man 'unvorhergesehenes'
- fast jede Methode einer Domino-Klasse kann einen Laufzeit-Fehler erzeugen (z.B. wegen Memory-Overflow, Netzwerk-Fehlern oder Festplatten-Problemen)
- Alternative: → Best Practices

LotusScript-Antipatterns: 6. nutzlose Kommentare

- Code-Beispiel:

```
x = x + 1                                          ' Zähler hochzählen
doc = view.GetNextDocument( doc)' nächstes Dokument holen

' ************************************************
' ***************** Deklarationen *****************
' ************************************************
Dim k As Long                                      ' Zähler
```

- auch schön: jede Änderung jedes Entwicklers seit 1992 ist im Code kommentiert

Teil 2
LotusScript Best Practices

- nachdem wir einige Beispiele gesehen haben vom Typ 'so nicht' stellt sich die Frage: 'wie denn nun ?'
- keine theoretische Abhandlung zur strukturierten Programmierung
- einige praktische Vorgehensweisen, welche die Qualität von LotusScript Code verbessern können

Best Practices:
1. Data Hiding

- 'Divide et impere' - teile und (be)herrsche
- Data-Hiding ist kein 'theoretischer Spinnkram' sondern erhöht die Code-Qualität wirksam und nachweisbar
- Kernfrage: bis wohin wirkt sich die Änderung der aktuellen Codezeile aus ?
- Anstatt von überall auf alles zuzugreifen müssen Werte als Parameter übergeben oder in Objekten zusammengefasst werden

Best Practices:
1. Data Hiding - Sichtbarkeit von Namen

1. Globale Sichtbarkeit: 'Container' (Maske,Ansicht,Agent,DBScript)

2. Sichtbarkeitsbereich 'Modul' (z.B. Button innerhalb Maske)

3. Sichtbarkeitsbereich 'Routine' (Function / Subroutine)

1. Sichtbarkeitsbereich 'ScriptLibrary'

2. Sichtbarkeitsbereich 'Klasse'

3. Sichtbarkeitsbereich 'Methode'

4. Sichtbarkeitsbereich 'Routine'

2. Sichtbarkeitsbereich 'Modul' (z.B. Button in Maske)

3. Sichtbarkeitsbereich 'Scriptlibrary'

4. Sichtbarkeitsbereich 'Klasse'

5. Sichtbarkeitsbereich 'Methode'

Best Practices:
1. Data Hiding

- Beispiel ScriptLibrary-Vorlage:

```
' Option Public – auskommentiert !!
Option Declare

Private session As NotesSession  ' Modul-Variablen deklarieren
Private thisDb As NotesDatabase
Private viewFuerAlle As NotesView

Sub Initialize                   ' Modul-Variablen initialisieren
  Set session = New NotesSession
  Set thisDb = session.CurrentDatabase
  Set viewFuerAlle = thisDb.GetView("LookupAdrByCompany")
End Sub

Public Sub UiVerbSubstantiv( uiDoc As NotesUiDocument ) ...

Private Function TuEtwasMitIrgendwem( doc As NotesDocument ) As String ...
```

Best Practices:
2. Error Logging überall

- Wirklich überall ? Mindestens an diesen Stellen:

1. in jeder Public Routine einer ScriptLibrary
2. in jedem Button/Action-Handler einer Maske / Ansicht etc.
3. in jedem Agent 'Initialize'
4. in allen Masken-Events
5. in allen DatenbankScript-Events

- Warum ?
  - Benutzer klicken oft Fehlermeldung weg ohne den Fehler zu melden ('keine Zeit')
  - Agenten loggen sonst 'ins Leere'

- jeder bekannt gewordene Laufzeit-Fehler ist ein Gewinn für die Code-Qualität !

Best Practices:
2. Error Logging überall

- → Empfehlung: 'minimal-invasives' Error-Logging
- wenn es aufwändig ist den Error-Logging-Code einzusetzen, dann lassen Programmierer das Logging gern mal weg ('zu viel Arbeit')
- Wenn der Error-Logging Code zu großen Anteil des Programm-Code beansprucht, dann leidet darunter die Klarheit und Wartbarkeit
- Beispiel für einen 'minimal-invasiven' Error-Handler (weniger geht nicht):

```
Public Sub HierEinBeispiel()
  On Error Goto ErrorHandler
  ...
ErrorHandler:
  Call LogRuntimeError(“”“)
  Exit Sub
End Sub
```

Best Practices:
2. Error Logging überall

- → Empfehlung: Error-Logging per Mail-Router in zentrale Mailin-DB
  - sonst dauert das Erzeugen des Logging-Eintrags zu lang (z.B. international)
  - asynchrone Zustellung per Router ist robuster als synchroner Schreibzugriff
- Ich stelle gern meine ScriptLibrary 'SimpleLogging' kostenlos zur Verfügung:

  → Mail mit Betreff 'SimpleLogging' an: gerd@gehil.de

Best Practices:
3. Datentyp 'Variant' vermeiden

- Es gibt nur wenige Ausnahmen:
  1. bei Verwendung von Array-Pointern (→ nächster Punkt)
  2. falls die Methode einer Notes-Klasse einen Variant als Resultat liefert
  3. falls eine Routine verschiedene Typen als Parameter bekommen muss (!?!)
  4. sonst noch was ?
- Bei jeder fremden, komplexen Notes-Anwendung finde ich 'unnötige' Variants
- Es ist immer ein Gewinn genau zu wissen welcher Datentyp notwendig ist

Best Practices:
4. Array-Pointer verwenden

- nicht dokumentiertes LotusScript Feature: 'Array-Copy' per Zuweisung
- wozu ? Weil in den Notes-Klassen unzählige Arrays vorkommen
- Beispiele:
  - NotesItem.Values
  - NotesDocument.LockHolders
  - NotesACL.Entries
- Um diese Arrays verarbeiten zu können ist es vorteilhaft eine Kopie im Speicher anzulegen

Best Practices:
4. Array-Pointer verwenden

- Code-Beispiel:

```
Dim multiValueArray As Variant
multiValueArray = doc.GetItemValue("$UpdatedBy")
```

- ein dynamisches Array wird als Kopie von NotesItem.Values angelegt !!
- alle Array-Funktionen von LotusScript lassen sich anwenden:

```
ArrayGetIndex(), ArrayAppend(), ArrayUnique(), FullTrim(), Implode() …
```

- damit lassen sich umständliche Programm-Schleifen effizient und robust ersetzen:

```
If Not IsNull(ArrayGetIndex(multiArray,"Gerd Hildebrandt/ghi/DE")) Then...
...
MsgBox Implode( multiArray,";" )
...
multiArray = ArrayAppend( multiArray, session.UserName )
```

Best Practices:
5. List As String

- das 'meist-übersehene' Feature von LotusScript
- assoziatives Array – Element-Zugriff nicht über einen numerischen Index sondern über einen Text-Schlüssel (Tag, Selector)
- Element-Operationen Hinzufügen, Finden und Löschen sind viel effizienter als per Array-Schleifen nachprogrammiert
- Code wird robuster und kompakter, weil ein Teil der Arbeit von der Liste erledigt wird anstatt ausprogrammiert und qualitätsgesichert werden zu müssen
- Liste kann auch von einer Notes-Klasse sein (z.B. List As NotesDatabase)
- Liste kann auch von einer eigenen Klasse sein
- Code wird lesbarer (wenn man List-Befehle versteht)

Best Practices:
5. List As String

• Code-Beispiele:

```
Dim departmentRecipients List As String

...

departmentRecipients(“Einkauf“) = “Hans Kaufma/ghi/DE“' hinzufügen

If IsElement(departmentRecipients(“Support“)) Then …      ' prüfen

Call mailDoc.Send(False,departmentRecipients(“Einkauf“))  ' auslesen

Erase departmentRecipients(“Karneval“)                    ' löschen

ForAll recipient In departmentRecipients …                ' alle verarbeiten
```

Best Practices:
6. Vorbedingungen prüfen (lean)

• In einer Routine beansprucht oft die Prüfung der Parameter und anderer Vorbedingungen erheblichen Anteil am gesamten Code. Dabei gerät oft die eigentliche Aufgabe (Kern-Logik) aus dem Blick

• Ich verwende daher eine eigene Assert() Routine (Name stammt aus C / C++)

• Code-Beispiel:

```
Private Function GetConfigurationValue( key As String ) As String
    Call Assert( key <> “” )          ' Abbruch bei unzulässigem Parameter
    Call Assert( konfigurationView )' Abbruch bei fehlender Ressource
```

• Wenn der Assert Parameter False, Nothing, Empty, Null, 0 oder “” ist, wird die Programmausführung abgebrochen und ein Log-Eintrag generiert mit der Information warum / wo der Abbruch passiert ist: „Assert fehlgeschlagen in Code-Zeile 2 von Routine 'GetConfigurationValue'“

• Die Routine Assert() ist verfügbar in der 'SimpleLogging' ScriptLibrary

Best Practices:
noch ein paar Tipps

- eigene 'schlanke' Klassen definieren (Keep-it-Simple)
- UI- und Backend-Code trennen (getrennte ScriptLibraries)
- Routinen sehr kurz halten (ca. eine Bildschirmseite), sonst aufteilen
- Kommentare minimieren (stimmen sowieso meist nicht dem Code-Stand überein) und stattdessen Namen 'hochgradig' sprechend vergeben ('ExportNetherlandCustomersToExcel')