—VIER—

ABAP-Infrastruktur und -Werkzeuge

AAA – Alles außer ABAP

Der Schrödinger hat es schon gemerkt: Zum ABAPen reicht es nicht aus, sich die Sprache ABAP selbst reinzuziehen. Denn wo liegen die Programme, welche Programme gibt es eigentlich, womit werden sie bearbeitet? Welche Speicherbereiche werden genutzt? Wie kommen die Programme zu den Anwendern? Und ungefähr weitere 1.000 Fragen. Aber der Schwaiger Roland ist ja (noch) geduldig.

Warum?

… musst du dir dieses Kapitel antun? Ganz einfach, weil ich denke, dass es gut für dich ist. Ich will nur dein Bestes!

Das Besondere an ABAP ist, dass es in einen **technischen Kontext** eingebunden ist, der mindestens **genauso wichtig** ist **wie ABAP selbst**. Da geht es um die **Werkzeuge** zur Entwicklung, um die **Organisation** der Entwicklung, um das **Drumherum** eben. Darum dauert es auch so lange, bis du zum Profi wirst, also unter einem Jahr ist mal nix zu machen.

Pfff, ein Jahr. Ich bin sicher schneller ein Profi.

Du bist natürlich schneller, das habe ich schon gesehen! Trotzdem: Um die Sprache ABAP zu können, benötigst du auch alle Infos zur Erstellung von Programmen. Du legst **Programme** in einem SAP-System an (**Object Navigator** und **Repository Browser**), die in einem speziellen Datenbankbereich (**Repository**) gespeichert werden. Und das nicht nur in Reintext (für Menschen interpretierbar), sondern auch in übersetzter Form (für Maschinen interpretierbar, auch als **Load** bezeichnet). Du bist ordentlich, und SAP ist da ganz auf deiner Seite. **Pakete** dienen dir zur Gruppierung von Programmen. Damit deine Programme auch produktiv einsetzbar sind, kannst du sie in Folgesysteme schicken, da hilft dir das **Transport- und Managementsystem**, wobei du lediglich **Änderungsaufträge** anlegen musst, um den **Transport** durchzuführen.

Das klingt aber nicht sehr spannend. Naja, wenn es sein muss …

Die **Werkzeuge** zur Erstellung der Programme sind wiederum eine ganz andere Geschichte, da gibt es so viele, dass ich beim Nachdenken schon nicht mehr weiß, wie ich dir alles erklären soll. Auf alle Fälle ist die **Königin der Werkzeuge** die **SE80** (man könnte auch sagen der **Object Navigator**, natürlich wäre das dann der König). Als kleiner Prinz mit großen Ambitionen haben sich die ABAP Development Tools (ADT) für Eclipse dazu gesellt. Von der SE80 aus kannst du alle schönen Werkzeuge aufrufen, auch zum Beispiel den **Debugger**. Den wirst du zwar nicht oft benötigen, da du keine Fehler machst, aber zur dynamischen Analyse deiner Programme ist er doch ziemlich praktisch.

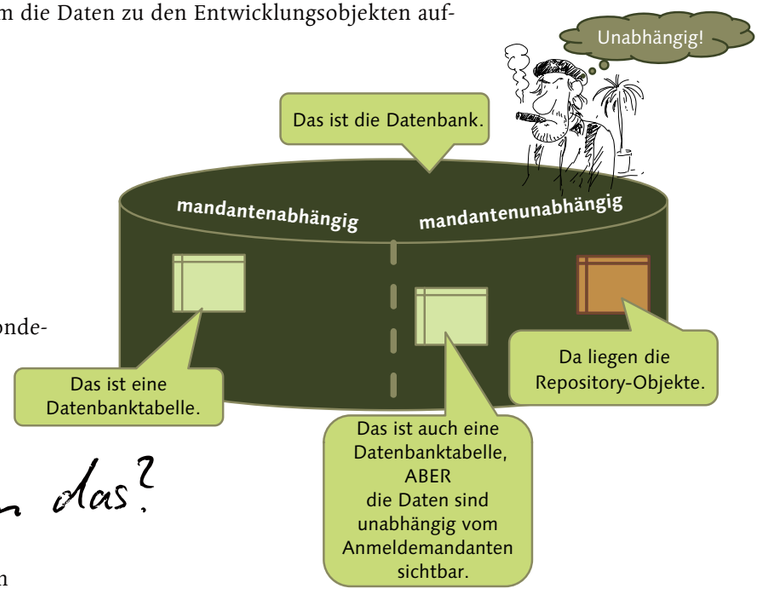
Stimmt, der Debugger wird nicht wichtig für mich. Oder doch?

Wir werden sehen. Einen weiteren Faktor habe ich noch gar nicht erwähnt. Um so richtig erfolgreich im SAP-System entwickeln zu können, ist sicher auch noch das **Applikationswissen** von Vorteil. Unter **Applikationswissen** versteht man das **Fachwissen** und wie es **technisch abgebildet** wird. Also, welche **Geschäftsprozesse** gibt es, welche Funktionalitäten sind vorhanden, wo liegen die Daten, wie kann ich auf die Daten zugreifen, wer darf/darf nicht Prozesse ausführen … Das werde ich hier **NICHT** besprechen. Dafür hat der nette Rheinwerk Verlag Hunderte honorige Autoren, die ihr Wissen über die Applikationen darbieten. Ich konzentriere mich auf **ABAP**.

Fangen wir also mal an, uns langsam dem Rundherum zu nähern. Zu Beginn steht meistens die Frage: **„Wo liegen die Programme, wo werden sie gespeichert?“**

Die Ablage der Entwicklungsobjekte – Repository

Entwicklungsobjekte, so wie Funktionsbausteine, Klassen, Tabellendefinitionen etc., müssen gespeichert werden. Du willst ja nicht immer wieder alles neu entwickeln. Dazu hat sich SAP das **Repository** einfallen lassen, das einen Teil der Datenbank in Anspruch nimmt und pro SAP-System die Daten zu den Entwicklungsobjekten aufnimmt.



Das Repository zwickt dir einen Teil der Datenbank ab. Es liegt in einem speziellen Bereich der Datenbank, nämlich dem mandantenunabhängigen.

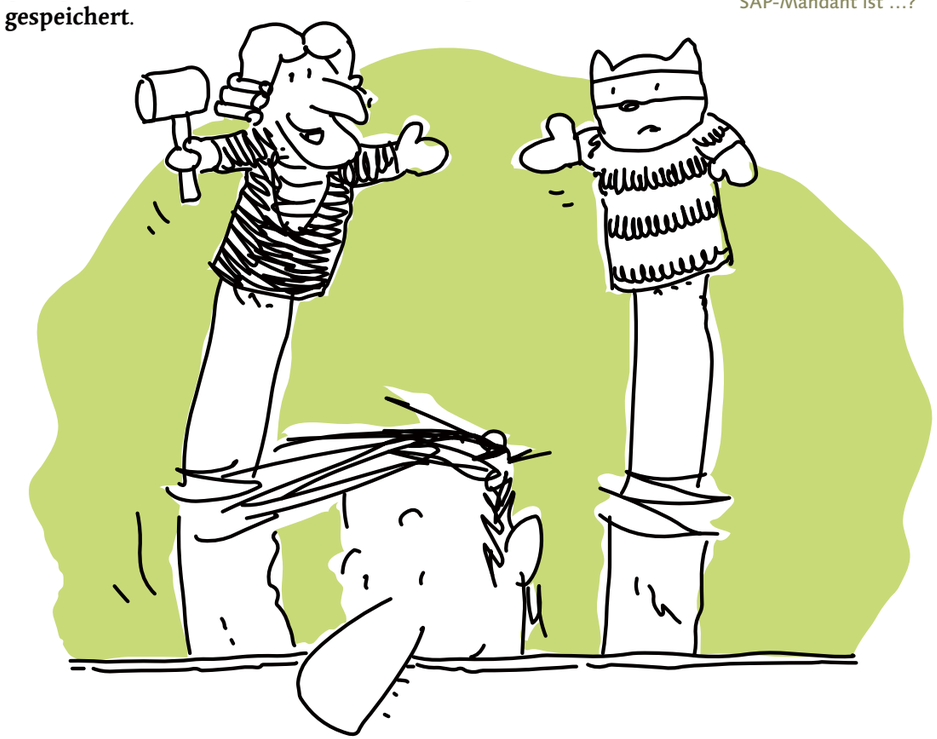
Das Besondere an den **Repository-Objekten** ist, dass sie in einem besonderen Bereich der Datenbank des SAP-Systems gespeichert werden, nämlich im **MANDANTEN-UNABHÄNGIGEN** Teil.

Was ist denn das?

SAP hat ein **Mandantenkonzept** im System abgebildet (Ein kleiner Blick in Kapitel 2 gefällig?). **Je nach Mandant**, an dem man sich anmeldet, sieht man **unterschiedliche Daten**, außer bei Daten,

die aus Tabellen stammen, die **mandantenunabhängig** sind. Da sieht man in jedem Anmeldemandanten die gleichen Daten. **Die Repository-Objekte werden in mandantenunabhängigen Tabellen gespeichert**.

Komm her, du kleiner Mandant! Übrigens: Die englische Übersetzung für SAP-Mandant ist ...?



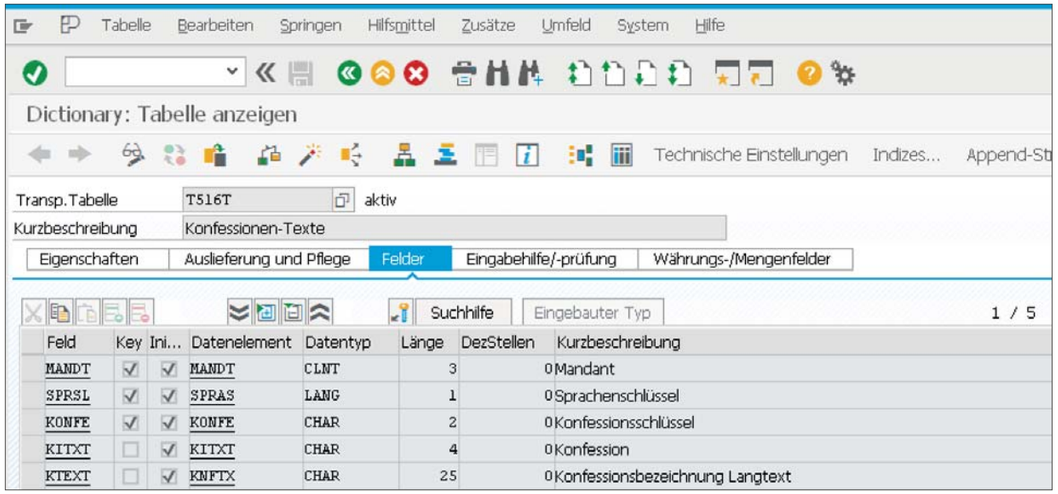
Beispiel: Die Tabelle **REPOSRC**, in der die **Programmquelltexte** gespeichert werden, ist **mandantenunabhängig**. Hingegen ist die Tabelle **USR01** mandantenabhängig und hält die Daten der Anwender.

Sehr interessant. Und, kratzt mich das?

Das wirst du nach der folgenden Deutung erkennen: Die Repository-Objekte sind in allen Mandanten sichtbar, das heißt, du kannst keine Programme schreiben, die nur in bestimmten Mandanten sichtbar sind. Im Gegensatz zu den Anwendern, die für jeden Mandanten angelegt werden müssen.

[Einfache Aufgabe]

Folgende Frage: Ich habe ein Bild einer Datenbanktabellendefinition für dich erstellt. Sind die Inhalte mandantenabhängig oder mandantenunabhängig?



Feld	Key	Ini...	Datenelement	Datentyp	Länge	DezStellen	Kurzbeschreibung
MANDT	✓	✓	MANDT	CLNT	3	0	Mandant
SPRSL	✓	✓	SPRAS	LANG	1	0	Sprachenschlüssel
KONFE	✓	✓	KONFE	CHAR	2	0	Konfessionsschlüssel
KITXT		✓	KITXT	CHAR	4	0	Konfession
KTEXT		✓	KNFTX	CHAR	25	0	Konfessionsbezeichnung Langtext

To be dependent or not to be ...

Ganz klar mandantenabhängig. Das **MANDT**-Feld haben wir schon in Kapitel 2 verwendet!

Wer Ordnung hält, hat kein Selbstvertrauen.

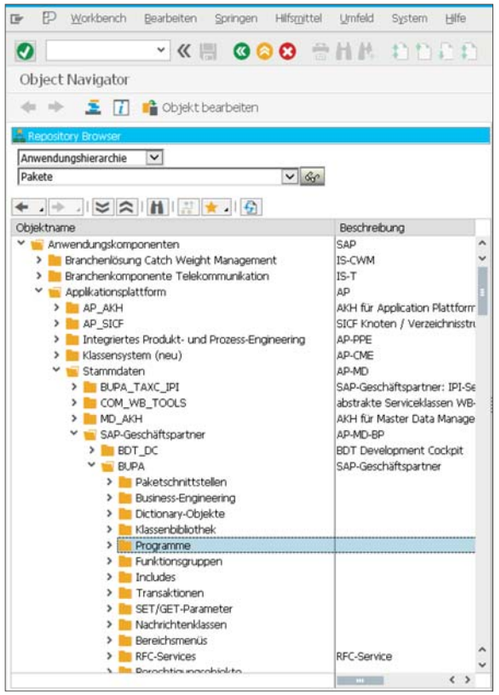
Räum erst mal auf, und erklär's mir dann noch mal.

Im Repository stecken aber **nicht nur deine Objekte**, sondern auch die von **SAP**. Das Repository ist kein Wirrwarr, so wie er in meinem Büro herrscht, sondern hat eine **Ordnung**.

Im Gegensatz zum **Schwaiger-Büro** sieht es im Repository aufgeräumter aus. Das Repository ist nach **Anwendungskomponenten** in einer **Anwendungskomponentenhierarchie** (puh, langes Wort) geordnet.

[Begriffsdefinition]

Die Anwendungskomponenten, auch als Business-Komponenten bezeichnet, dienen zur Strukturierung der SAP-Funktionalität. Ihnen sind Business-Objekte zugeordnet, die betriebswirtschaftliche Daten und Funktionalitäten kapseln.



Das ist ordentlich, fein säuberlich nach Anwendungskomponenten einsortiert.

Okay, noch mal: Die ANWENDUNGSKOMPONENTEN …

… sind, vereinfacht gesagt, Gruppierungsmöglichkeiten, und kompliziert gesagt, ein Modellierungskonstrukt für deine Entwicklungsobjekte, so kann man wiederverwendbare Teile nach fachlichen Aspekten zusammenfassen. Zum Beispiel Financials (FI), Controlling (CO), Sales & Distribution (SD) …

Die **Business-Objekte** sind in SAP schon lange vorhanden, weit länger als die **Objektorientierung** in ABAP. Sie stellen Daten und Funktionen mit Bezug zu einem Geschäftsobjekt (zum Beispiel ein Vertrag, ein Geschäftspartner, eine Bestellung etc.) zur Verfügung und werden auch gerne im **Workflow** verwendet, also in der Schritt-für-Schritt-Abarbeitung von Prozessen. Business-Objekte repräsentieren **Geschäftsobjekte**. Business-Objekte sind aber auch nur ein Beispiel dafür, was einer Anwendungskomponente zugeordnet werden kann. Programme, Funktionsgruppen, ABAP-Klassen etc. – allgemeiner: Entwicklungsobjekte – können ebenfalls einer Anwendungskomponente zugeordnet sein.

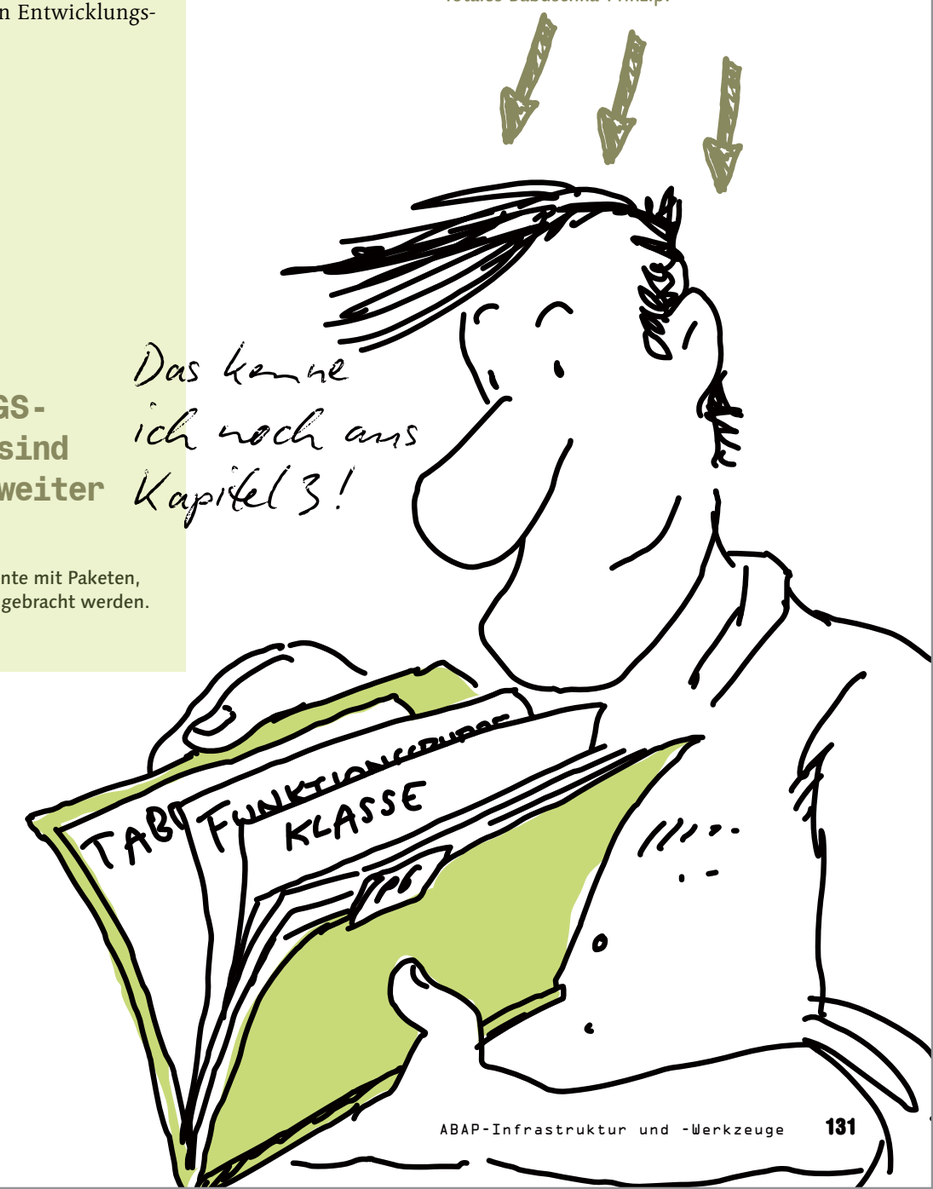
Entwicklungsobjekte sind mandantenunabhängig und werden Paketen und diese den Anwendungskomponenten zugeordnet.

Dem **Paket** sind die **Entwicklungsobjekte** zugeordnet. Obwohl diese Zuordnung **zeitlich unbegrenzt** definiert ist, kannst du ein Entwicklungsobjekt während der Entwicklungszeit von einem Paket zu einem anderen umhängen. Das kommt aber nicht so oft vor, weil du dir ja vorher überlegt hast, in welches Paket du dein Entwicklungsobjekt legen möchtest.

Alles gut verpackt im Paket. Das Paket ist die letzte Kapselungsebene für Entwicklungsobjekte. Jedoch stehen dir drei Paket-Granularitätsebenen zur Verfügung: Das Strukturpaket (das grobgranulare) dient zur Zusammenstellung von Hauptpaketen. Ich vergleiche Strukturpakete gern mit Anwendungskomponenten. Die Hauptpakete sammeln Standardpakete. Die Standardpakete, auch als Nicht-Hauptpakete bezeichnet, sammeln die Entwicklungsobjekte. Totales Babuschka-Prinzip.

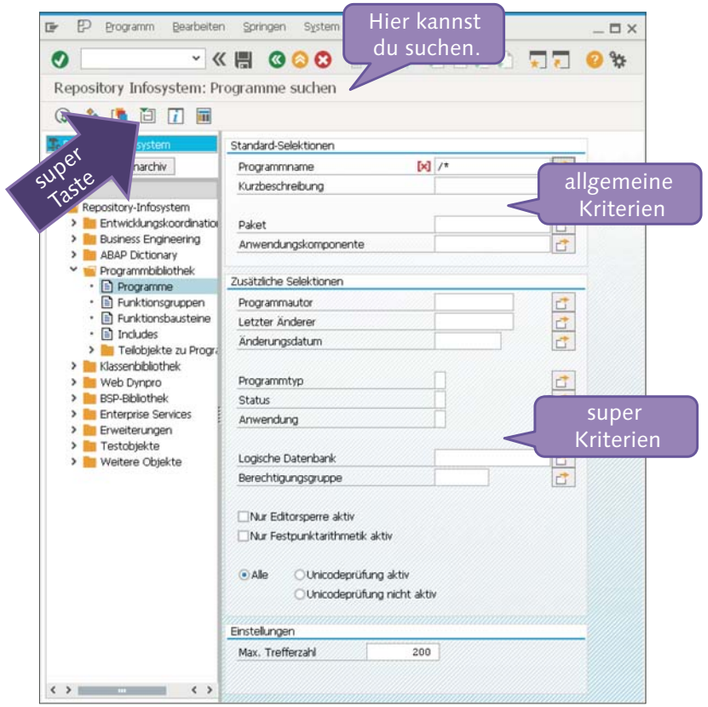
Die ANWENDUNGSKOMPONENTEN sind durch PAKETE weiter unterteilt.

Eine Anwendungskomponente mit Paketen, die aber nicht von der Post gebracht werden.



Wer suchet der findet – Suchen mit dem Repository Infosystem

Damit du im **Repository suchen** kannst, hat sich SAP das **Repository Infosystem** für dich überlegt. Mit der Transaktion **SE84** beginnt das Suchvergnügen.



Ich sehe was, was du nicht siehst …

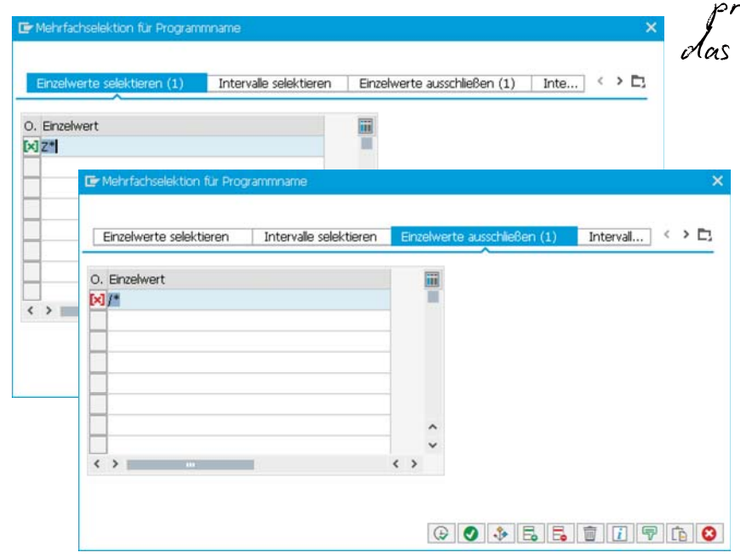
Das **Infosystem** ist schön **nach technischen Aspekten** aufgebaut. Wenn du zum Beispiel nach Programmen suchst, verwendest du die Kategorie **Programmbibliothek** und dort den Punkt **Programme**. Wenn du dann noch die **Super-Taste** (**Alle Selektionen** (⇧ + F9)) drückst, stehen dir sehr viele **feingranulare Suchmöglichkeiten** zur Verfügung, wie zum Beispiel nach dem Programmautor oder dem Zeitpunkt der letzten Änderung.

[Einfache Aufgabe]
Such doch mal nach den Programmen, die du entwickelt hast. Die beginnen alle mit ZSCH. Dazu trägst du deinen Benutzernamen (zum Beispiel BCUSER) im Feld **Programmautor** und die Zeichenkette ZSCH* im Feld **Programmnamen** ein. Starte die Suche mit der Drucktaste **Ausführen** (F8). Je länger die Liste, desto größer die Spuren, die du im SAP-System hinterlassen hast.

Wo wir nun schon dabei sind, kann ich dir die Verwendung **einer Select-Option** zeigen. Das ist ein Wunderding. Mit der kannst du sooooo **komplexe Suchbedingungen** eingeben, dass sich dein Neocortex nochmals faltet. **Mein Wunsch ist, dass wir nur jene Programme suchen, die mit Z beginnen, aber kein _ an der zweiten Stelle haben**.
Dazu klickst du auf den Pfeil (**Mehrfachselektion**) rechts neben dem Feld **Programmnamen**. Die Karteireiter des erscheinenden Dialogs bieten dir die Möglichkeit, **Suchkriterien** (manche unter uns sagen auch **Selektionskriterien**) einzugeben. Mit **Platzhaltern** und dem ganzen Pipapo.
Ungewöhnlich ist, dass du **einschließende** (das soll im Ergebnis drinnen sein) und **ausschließende** (das soll sicher nicht dabei sein) **Bedingungen** pflegen kannst. Für meine Aufgabenstellung „‚**Z**' dabei, ‚**_**' nicht dabei" würden wir also auf dem Karteireiter **Einzelwerte selektieren** in der ersten Zeile den Wert „**Z***" und auf dem Karteireiter **Einzelwerte ausschließen** den Wert „**Z_***" eingeben. Mit der Taste **Übernehmen** (F8) bestätigst du die Eingabe für die Suchverwendung.

[Notiz]
Die Liste der Platzhalter für die Textsuche lautet: * und +, wobei * für eine beliebige Zeichenkette und + für exakt ein Zeichen steht. Das #-Zeichen, auch als Escape- oder Fluchtsymbol bezeichnet, hat eine Sonderfunktion. Damit kannst du nach Sonderzeichen wie ! (Ausrufezeichen), * (Stern) oder nach dem #-Zeichen selbst suchen, indem du dem gesuchten Zeichen ein # voranstellst. Beispiel: bei der Suche nach * also #*.

Rein damit und raus damit! Du kannst Einzelwerte und Intervalle in die Selektion mit aufnehmen oder von dieser ausschließen.

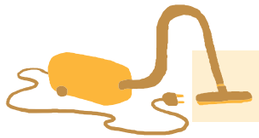


Ist SAP ein Zeichenprogramm? Jetzt ist das Kästchen beim Pfeil grün gefüllt?

Und das ist gut so, denn das System hat die Werte in die Selektionskriterien übernommen, und mit dem vollen Kästchen wird dir angezeigt, dass da mehr als ein Wert hinterlegt ist.

[Schwierige Aufgabe]
Such doch mal alle von dir erstellten Datenbanktabellen, die du im aktuellen Jahr angelegt hast.

Suchen und Finden – Repository Infosystem



[Einfache Aufgabe]
Probiere das mal aus. Suche in der Anwendungskomponente, deren Namen mit BC beginnt, nach Datenbanktabellen, die mit SCARR beginnen.

Das war einfach: Zuerst die SE84, dann den Knoten ABAP Dictionary öffnen und auf das Blatt Datenbanktabelle doppelklicken. Tabellenname SCARR* und Anwendungskomponente BC* eingeben und – Schwups –, schon ist die Treffermenge mit einem Eintrag für SCARR da.

Du Scherzkeks, da gibt es ja nur einen Treffer!

Ach so, das wusste ich nicht. ☺ Gut gemacht, dann noch eine Suche mit mehreren Treffern.

[Schwierige Aufgabe]
Suche nach „Professional User Transaction" – **Transaktionen**, die irgendwo ABAP in der Bezeichnung haben.

Das ist schon gefinkelter. Ich fühl mich richtig gefordert. Zuerst hab ich die SE84 aufgerufen, den Knoten Weitere Objekte geöffnet (das war fies), Doppelklick auf das Blatt Transaktionen, die Super-Taste Alle Selektionen (⇧ + F9) gedrückt (das war noch fieser) und dann im Feld Transaktionscode *ABAP* eingegeben und im Block Zusätzliche Selektionen Klassifikation. Professional User Transaction ausgewählt. Pffff.
Da kommen tatsächlich einige Ergebnisse. Spannend ist die Transaktion ABAPDOCU, die hab ich gleich ausprobiert. Lauter schöne Beispiele für die Programmierung, das gefällt mir!

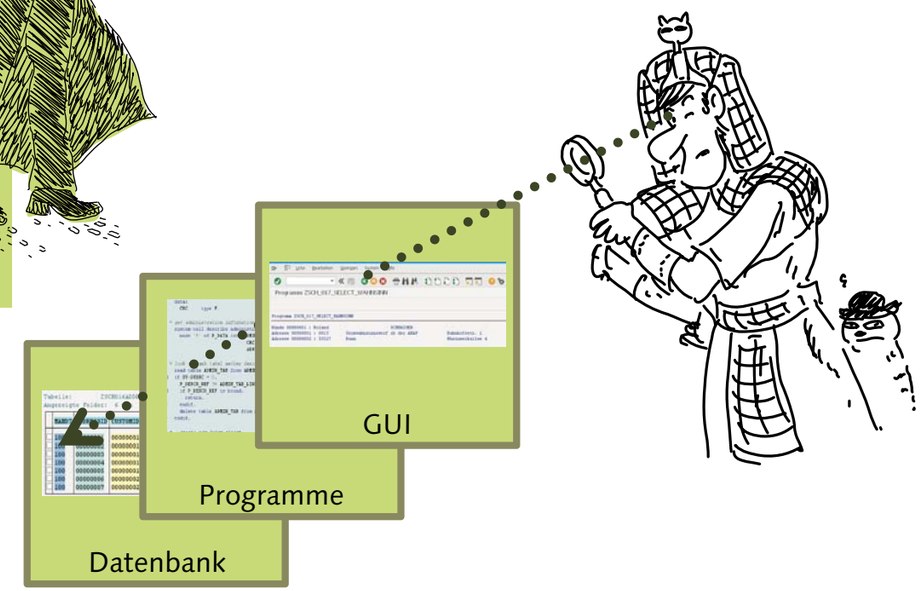
Und mich freut, dass du glücklich bist.
Das ist ein schöner Tag für mich.

Geschichtet, aber nicht gefaltet – ABAP-Infrastruktur

Jetzt komme ich noch zu der Frage: Wo läuft was ab?
Was ist der Unterschied zwischen dir vor dem SAP GUI und einem Anwender vor dem SAP GUI?

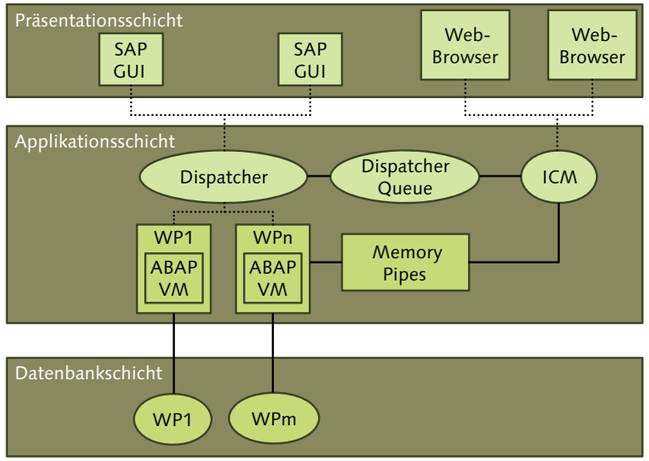
Ich sehe cooler aus!

Auf alle Fälle, UND du hast im Gegensatz zum Anwender den **Röntgenblick**. Dieser kann nur beurteilen, was er an der Oberfläche sieht, und weiß im Allgemeinen nicht, wie das da auf der Oberfläche zustande kommt.
Mit Röntgenblick meine ich, dass du durch das GUI hindurchsiehst, das dahinterliegende Programm erkennst und bis zur Datenbank guckst. Das GUI, die Programme und die Datenbank sind wie die **Schichten** einer **Torte** organisiert.



Der Röntgenblick durch die Systemschichten.

Jede Schicht trägt zur erfolgreichen Abarbeitung der Teile deines Programms bei.
Ich könnte sogar so weit gehen, dass ich **Programmteile** den **Infrastrukturschichten** zuordne. Kannst du dich noch an das Programm `zsch_03_durchblick` aus Kapitel 3 erinnern? Das werde ich jetzt als Anschauungsstück verwenden.



Schichten schlichten,
Schichten sichten.

In der **Präsentationsschicht** – das ist da, wo die Anwender sitzen und in einen Bildschirm hineinschauen – können ein **SAP GUI** und/oder ein **Webbrowser** installiert sein. Dort kann der interessierte Betrachter die Eingabe und die Ausgabe der Programme beobachten.


```
*&---------------------------------------------------------------------*
*& Report  ZSCH_03_DURCHBLICK
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  zsch_03_durchblick.
* TABLES Struktur für Dynpro Daten
TABLES: zsch03project.

* Controls
DATA: gr_container TYPE REF TO cl_gui_custom_container,
      gr_picture TYPE REF TO cl_gui_picture.

* Hier springt die Laufzeitumgebung rein
INITIALIZATION.

* Hier springt die Laufzeitumgebung rein
START-OF-SELECTION.

                                                    INTO gs_project

* Jetzt auch mit logischer Kontrolle
  IF sy-subrc = 0.
* und auf der Liste ausgeben
    WRITE: / gs_project.
  ELSE.
                                    hts gefunden für Projekt = ', pa_proj.

* Hier springt die Laufzeitumgebung rein
AT LINE-SELECTION.
```

** Parameter für das Projekt*
PARAMETERS: pa_proj TYPE zsch03project-projekt OBLIGATORY.

** und in der Liste ausgeben*
WRITE: / gs_project.

** Dynpro aufrufen*
CALL SCREEN 9100.

PARAMETERS, WRITE und CALL SCREEN beeinflussen die Darstellung auf der Präsentationsschicht.

Im Programm gibt es spezialisierte Anweisungen, die das GUI erzeugen, wie zum Beispiel **`WRITE`**, **`PARAMETERS`** oder auch **`CALL SCREEN`**.
Wo werden die Programme ausgeführt? Auf der nächsten Ebene, der **Applikationsschicht**, befindet/n sich der/die **Applikationsserver**. Konzentrieren wir uns mal auf einen Applikationsserver. Der sogenannte **Dispatcher** (Verteiler) übernimmt die Anfrage aus der Präsentationsschicht in den Applikationsserver und reicht diese an einen

Arbeitsprozess (Work Process) oder, falls alle belegt sind, an die **Dispatcher Queue** weiter. Ab in die Warteschlange, aber bitte streng nach FIFO: First in, First out, oder frei übersetzt: **Wer zuerst kommt, mahlt zuerst**.

Der Dispatcher hat den Überblick. Eine Anfrage zum Arbeitsprozess oder in die Schlange, das sind die essenziellen Fragen in der Welt des …


```
*&---------------------------------------------------------------------*
*& Report  ZSCH_03_DURCHBLICK
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  zsch_03_durchblick.
* TABLES Struktur für Dynpro Daten
TABLES: zsch03project.
* Parameter für das Projekt
PARAMETERS: pa_proj TYPE zsch03project-projekt OBLIGATORY.
* Die Variable zum Befüllen
DATA: gs_project TYPE zsch03project.

* Controls
DATA: gr_container TYPE REF TO cl_gui_custom_container,
      gr_picture TYPE REF TO cl_gui_picture.

* Hier springt die Laufzeitumgebung rein

* Jetzt auch mit logischer Kontrolle
  IF sy-subrc = 0.
* und auf der Liste ausgeben
    WRITE: / gs_project.
  ELSE.
* der arme Benutzer
    WRITE: / 'Och schade, nichts gefunden für Projekt = ', pa_proj.
  ENDIF.

* Hier springt die Laufzeitumgebung rein
AT LINE-SELECTION.
```

** Die Variable zum Befüllen*
DATA: gs_project TYPE zsch03project.

** Controls*
DATA: gr_container TYPE REF TO cl_gui_custom_container,
gr_picture TYPE REF TO cl_gui_picture.

Die **Programme** laufen in den **Arbeitsprozessen**. Man könnte auch sagen, ein Arbeitsprozess ist die **Laufzeitumgebung** für das Programm. Programme erhalten Ressourcen, um zum Beispiel Daten in Variablen zu speichern.

Die ABAP VM ist eine sogenannte **virtuelle Maschine**. Ein Begriff, der zum Beispiel auch in Java verwendet wird und andeutet, dass die Programme in einer isolierten, sicheren und gesicherten Umgebung ablaufen.

Die Ressourcennutzung auf der Applikationsschicht.

```
*&---------------------------------------------------------------------*
*& Report  ZSCH_03_DURCHBLICK
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  zsch_03_durchblick.
* TABLES Struktur für Dynpro Daten
TABLES: zsch03project.
* Parameter für das Projekt
PARAMETERS: pa_proj TYPE zsch03project-projekt OBLIGATORY.
* Die Variable zum Befüllen
DATA: gs_project TYPE zsch03project.

* Controls
DATA: gr_container TYPE REF TO cl_gui_custom_container,
      gr_picture TYPE REF TO cl_gui_picture.

START-OF-SELECTION.
* Das mächtige WRITE zaubert eine Zeile auf die Liste
  WRITE: / 'Durchblick 2.0'.
* Einzelsatz lesen
  SELECT SINGLE * FROM zsch03project INTO gs_project
    WHERE projekt = pa_proj.
* Jetzt auch mit logischer Kontrolle
  IF sy-subrc = 0.
* und auf der Liste ausgeben
    WRITE: / gs_project.
  ELSE.
* der arme Benutzer
    WRITE: / 'Och schade, nichts gefunden für Projekt = ', pa_proj.
  ENDIF.

* Hier springt die Laufzeitumgebung rein
AT LINE-SELECTION.
```

** Einzelsatz lesen*
SELECT SINGLE * FROM zsch03project INTO gs_project
WHERE projekt = pa_proj.

Kommunikation mit der Datenbank.

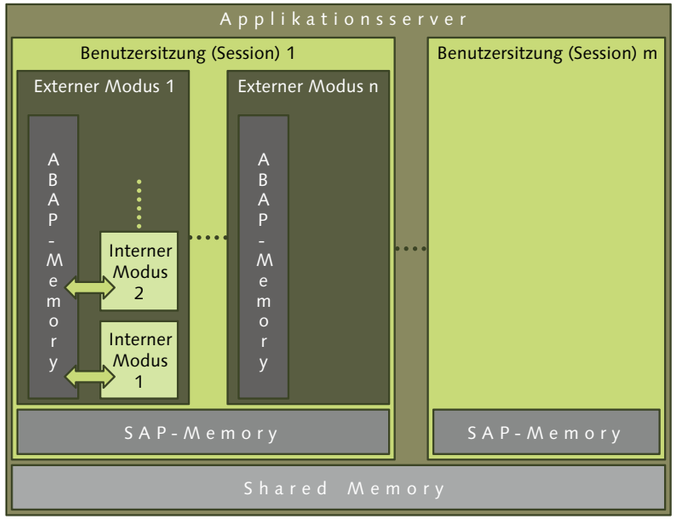
Mit der **DATA**-Anweisung nutzt du die Ressourcen, die dir durch den Arbeitsprozess zur Verfügung gestellt werden. Des Weiteren werden durch die Programmlogik natürlich die Arbeitsprozessressourcen kräftig beansprucht.
Falls Daten von der Datenbank benötigt werden, wird auf die Datenbank zugegriffen. Jedem **Arbeitsprozess** ist eine **Datenbankverbindung** fest zugeordnet. Eine feste Zuordnung von Anwendern zu Arbeitsprozessen gibt es nicht. Das hilft bei der Skalierung der Benutzeranfragen!
Eine längere Benutzersitzung kann der Reihe nach unterschiedliche Workprozesse nutzen. Diese Info wird noch bei der Besprechung der SAP-Transaktionen wichtig.

[Belohnung]
Damit du noch von Schichten träumst, könntest du dir eine ordentliche Portion Tiramisu gönnen, mit einem Espresso. Wohl bekomms!

Wohin mit dem Anwender – Speicherorganisation extern und intern

Alles wieder okay.
Der Zuckerhaushalt
wurde wieder optimiert.
Bitte weiter im Programm.

Nachdem wir das Schichtenmodell durchhaben, kommen wir zur **Speicherorganisation** und der **Benutzersitzung**. Zu (d)einer **Benutzersitzung** wird ein sogenannter **externer Modus** (Hauptmodus) geöffnet, dem das **ABAP-Memory** zugeordnet ist, in das du Daten ablegen kannst.



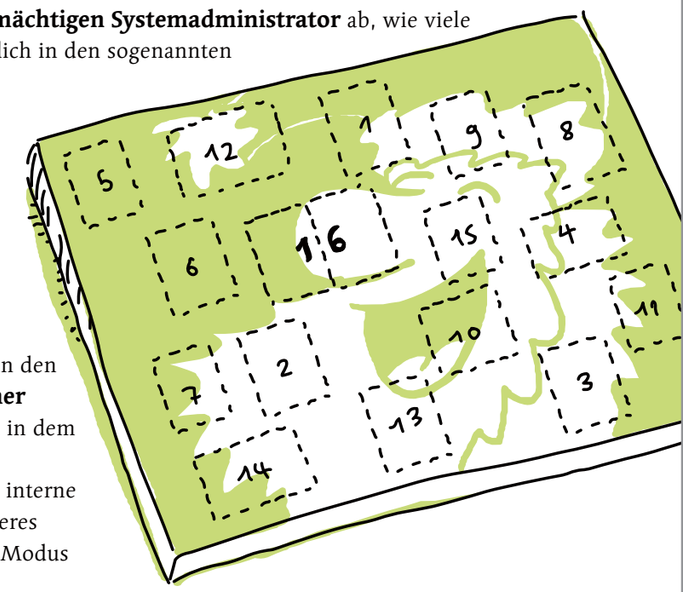
Speicherkammer. Für jeden Benutzer werden Daten zu seiner Sitzung (Session) verwaltet.

Externes Modus klingt ja, als wenn das außerhalb des SAP-Systems wäre.

[Hintergrundinfo]
Stell dir für einen externen Modus einfach ein Fenster des SAP GUI vor.

Möchtest du weitere Fenster (externe Modi) öffnen, verwendest du **`/o+<Transaktionscode>`** im **Transaktionscode-Feld** oder drückst die Drucktaste **Neuen Modus erzeugen** in der Drucktastenleiste. Du kannst seit Release 7.0 maximal 16 Fenster, das heißt externe Modi, öffnen. Vorher waren es maximal sechs Fenster. Aber wie immer hängt es vom **allmächtigen Systemadministrator** ab, wie viele Fenster du aufmachen kannst. Der stellt das nämlich in den sogenannten **Profilparametern** ein.

Was, 16 Fenster? Unglaublich.



Wenn du nun ein Programm aufrufst, wird im externen Modus ein **interner Modus** angelegt, in den das gerufene Programm geladen wird. Ein **interner Modus** besitzt einen **eigenen Speicherbereich**, in dem alle verwendeten **Objekte** des Programms leben.
In einem externen Modus können maximal neun interne Modi existieren. Also falls ein Programm ein anderes Programm aufruft, wird dafür ein neuer interner Modus im selben externen Modus erzeugt.
Die Programme können ihre Daten in das **ABAP-Memory** ablegen und von dort Daten lesen. Auf 64-Bit-Plattformen kann ein interner Modus theoretisch bis zu 4 TB Speicher anfordern. Auf 32-Bit-Plattformen liegt die theoretische Obergrenze bei 4 GB. Wobei das System natürlich durch den physikalisch installierten Hauptspeicher beschränkt ist. Das **SAP-Memory** dient zur Ablage elementarer Daten über externe Modi hinweg, aber immer nur für einen Benutzer.

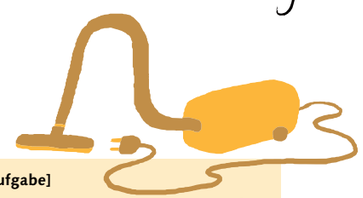
Werkzeugkiste – Entwicklungswerkzeuge

Du hast es gut!

Ich? Warum?

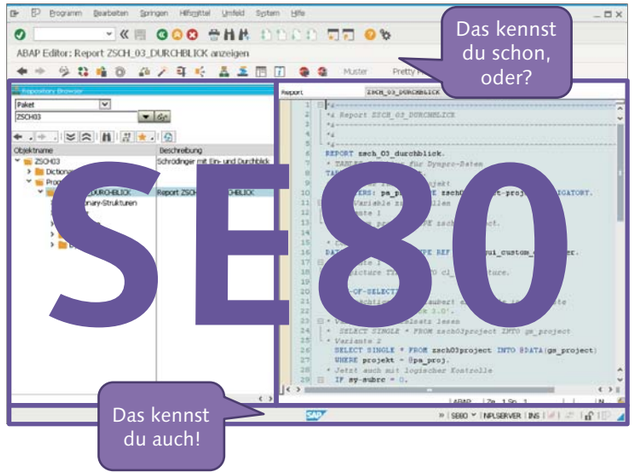
Das will ich dir sagen: Es reicht, dass du dir eine Transaktion merkst: SE80. Kein Suchen mehr, nichts, was du verlegen kannst. Die Frage: „Schatzi, hast du meinen Editor gesehen, ich kann ihn nicht finden“ kannst du aus deinem Sprachrepertoire streichen. Mit dieser wunderbaren, unglaublichen Transaktion kannst du jedes Werkzeug starten, das du für die Entwicklung benötigst! Dein Universalschlüssel zur allumfassenden Entwicklung.

Endlich selbstständig!



[Einfache Aufgabe]
Probiere es doch einfach aus. Ruf die SE80 auf, und mach dich mit ihr vertraut. Spiel einfach herum. Du kannst nichts zerstören.

Object Navigator – Der Regisseur



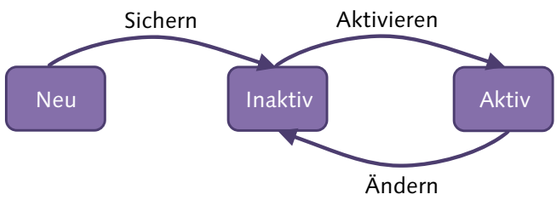
SE80, das Füllhorn der Entwicklung.

Der **Object Navigator** ist deine Schaltzentrale, dein Schweizermesser, für Repository-Objekte: **Anlegen**, **Ändern**, **Löschen**, **Umbenennen**, **Aktivieren**, **Kopieren**, und vieles mehr wird dir zur Verfügung gestellt.

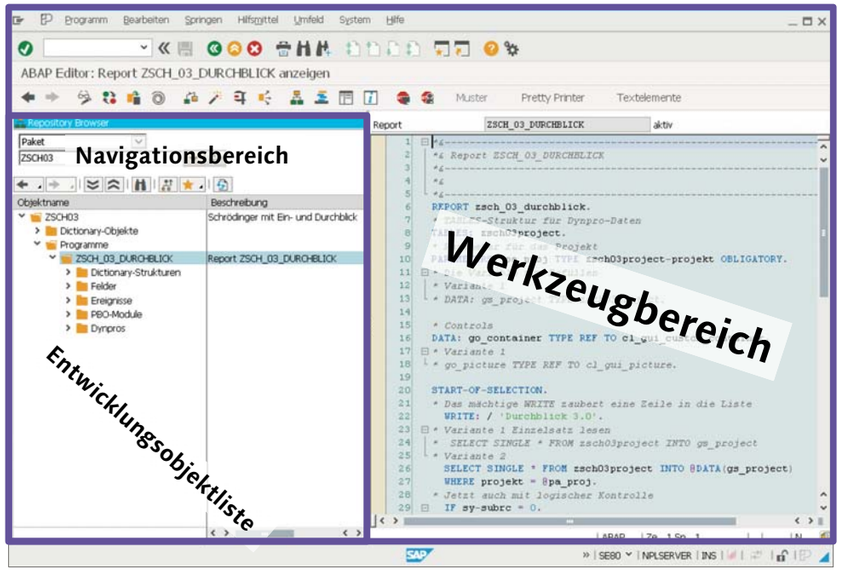
[Achtung]
Es wird Zeit, über **Zustände** zu sprechen. Ein Entwicklungsobjekt im SAP-System hat prinzipiell drei Zustände: **neu**, **in-aktiv** und **aktiv**. Das siehst du immer im Werkzeug, rechts neben dem technischen Namen des Objektes. **Neu** ist das Objekt, wenn es angelegt, aber noch nicht gespeichert ist. **Inaktiv** ist es, wenn es gespeichert ist, aber noch nicht aktiviert wurde. Und **aktiv** ist es, wenn es aktiviert wurde. Du kannst inaktive Objekte testen, andere Teilnehmer im System jedoch nicht. Die können nur aktive Objekte testen. Ein System kann somit eine inaktive und eine aktive Version besitzen. Wird die inaktive Version aktiviert, ist die inaktive futsch und nur mehr die aktive Version übrig.

Der SE80-Wunderwuzi.

Aktiv – Inaktiv – Aktiv – Inaktiv.
Klingt nach einem Double-Binding-Problem, sind aber die zwei Zustände, die ein Entwicklungsobjekt annehmen kann.



Wenn du die SE80 startest, erscheint die SE80 zunächst einmal sehr minimalistisch. Der Object Navigator ist in zwei Teile unterteilt, den **Navigationsbereich** mit der **Entwicklungsobjektliste** und den **Werkzeugbereich**.



Der **Navigationsbereich** hat im oberen Bereich einige **Drucktasten** (zum Beispiel **MIME Repository**, **Repository Browser**, **Repository Infosystem**), mit denen du spezielle Browser einstellen kannst, die unterhalb der Drucktasten angezeigt werden.

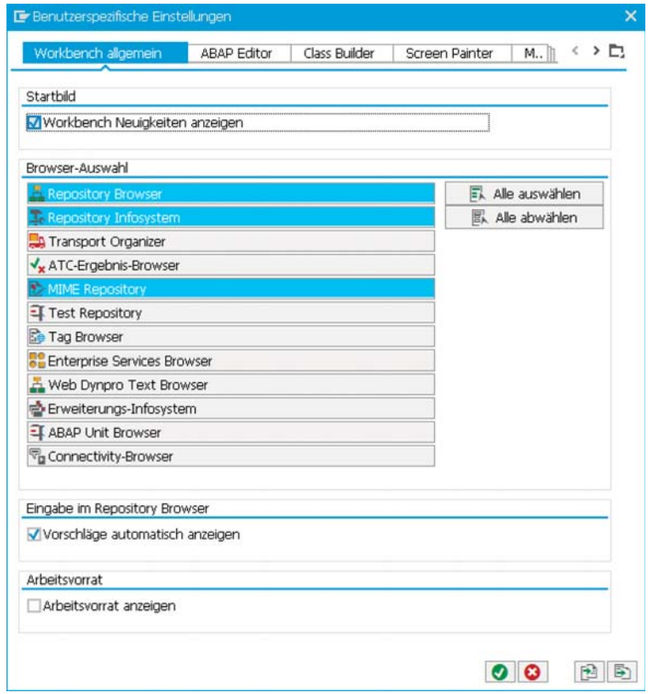
Bereiche. Die Hauptregionen des Object Navigators.

Das Repository Infosystem erkenne ich wieder. Da hast du doch so fiese Aufgaben dazu gestellt.

Nur zu deinem Besten! Wenn du zum Beispiel die Repository-Objekte sehen willst (das wird in 99 % der Fälle so sein, darum ist das auch die Standardeinstellung), drückst du die Drucktaste **Repository Browser**, und schon erscheint unterhalb der Drucktasten der dir bereits bekannte Repository Browser.

Du kannst noch zusätzliche Funktionen in der SE80 einblenden. Wenn du den Menüeintrag **Hilfsmittel • Einstellungen** gewählt hast – probiere doch mal! –, erscheint das folgende mächtige Einstellungsfenster.

Damit holst du deine Browser wieder zurück oder schickst sie in die Dunkelheit – benutzerspezifische Einstellungen.



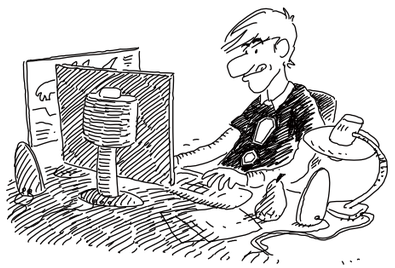
Auf dem Karteireiter **Workbench allgemein** kannst du deine gewünschten Browser durch Setzen der Checkbox für die Anzeige auswählen. Da ist ja der **Repository Browser**!

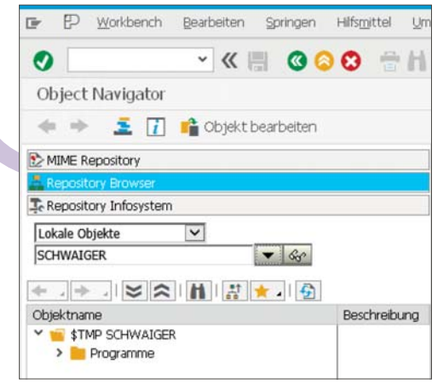
Einfach anklicken, und schon sind die Funktionen im Navigationsbereich sichtbar!

[Einfache Aufgabe]
Wir werden immer wieder mal auf die Einstellungen zu sprechen kommen. Sieh dir doch mal die ganzen Karteireiter an, und im Speziellen den Karteireiter **ABAP Editor**. Suche die Einstellung für **Groß-/Kleinkonvertierung**, und setze sie auf **Schlüsselwort groß**.

Repository Browser ausnutzen

Ich hab ihn wieder! Weiter geht's!





Der Repository Browser bietet dir Suchfelder und eine Liste mit Entwicklungsobjekten zu den Suchkriterien. Natürlich nur, wenn er etwas findet!

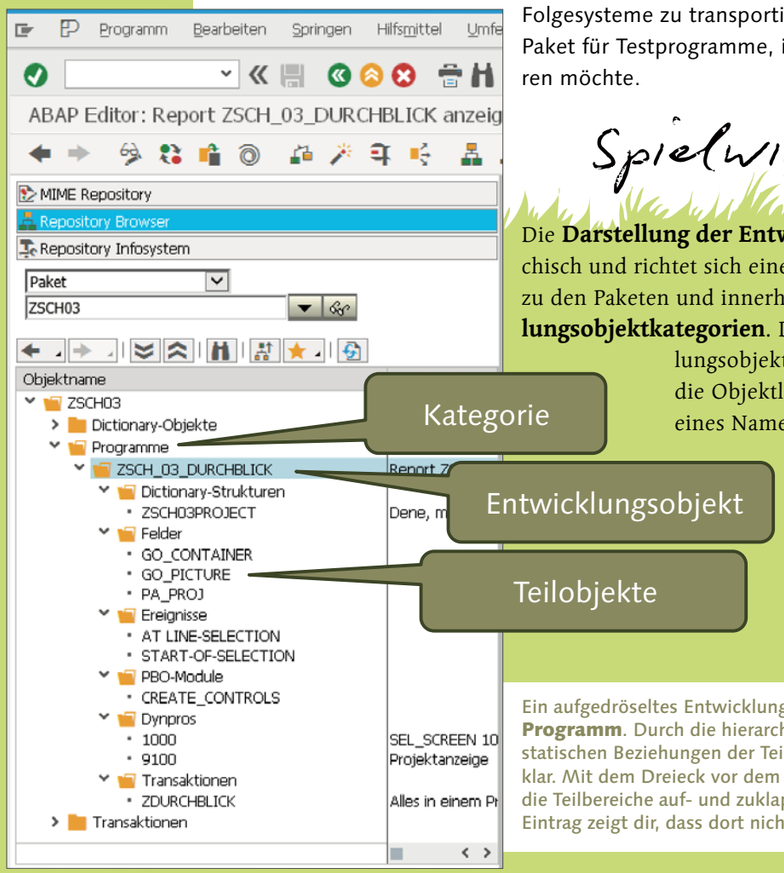
[Einfache Aufgabe]
Probier's mal aus! Suche deine lokalen Objekte im System.

*Was nicht schwer, einfach in der Drop-down-Liste den Eintrag **Lokale Objekte** auswählen, dann wird schon mein Benutzername vorgeschlagen, und die Brille drücken. Hast du noch mehr Brause?*

Repository Browser browsen

Unterhalb der Drucktastenleiste (kannst du auch liebevoll Druckerei nennen) befindet sich **die Browser-Sicht**. Wenn du in deinem Lieblings-Browser **Repository Browser** bist, kannst du dort den Einstieg der Suche im Repository auswählen (Objektlistenauswahl) – zum Beispiel **Paket** – und im Eingabefeld darunter den **Namen des Objektes** angeben – zum Beispiel **$TMP** – und die **Brille** (Anzeigen) oder die **Enter**-Taste drücken oder mit dem **Dreieckerl** suchen. Wenn ich dir alles verständlich erklärt habe und du gesucht hast, dann sollte dir das System eine Liste der zugehörigen Objekte anzeigen.

[Hintergrundinfo]
Das Paket **$TMP** ist dafür gedacht, sogenannte lokale Objekte zu sammeln. Das sind Objekte, die nicht in Folgesysteme zu transportieren sind. Ich verwende das Paket für Testprogramme, in denen ich etwas ausprobieren möchte.



Spielwiese

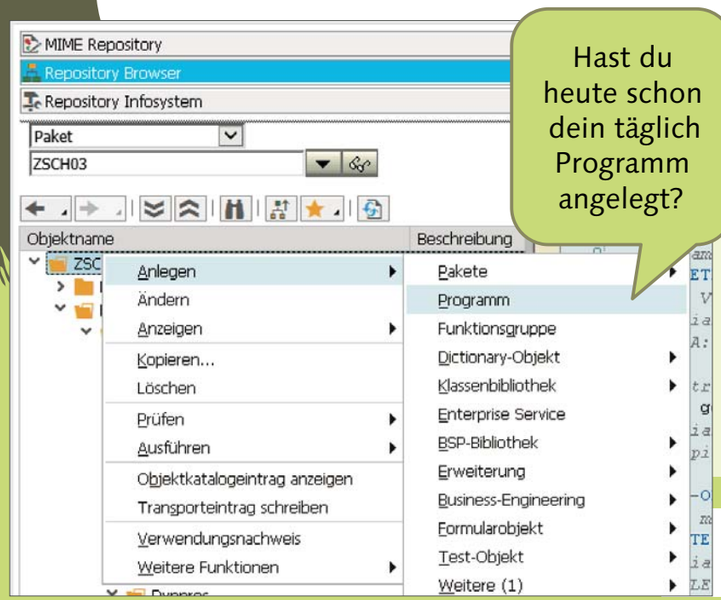
Die **Darstellung der Entwicklungsobjekte** ist hierarchisch und richtet sich einerseits nach der Zuordnung zu den Paketen und innerhalb eines Pakets nach **Entwicklungsobjektkategorien**. Die Auswahl, welche Entwicklungsobjekte darzustellen sind, wird durch die Objektlistenauswahl und die Angabe eines Namens passend zum selektierten Objekt durchgeführt.

Ein aufgedröseltes Entwicklungsobjekt aus der Kategorie **Programm**. Durch die hierarchische Darstellung werden die statischen Beziehungen der Teilobjekte zum Entwicklungsobjekt klar. Mit dem Dreieck vor dem Verzeichnissymbol kannst du die Teilbereiche auf- und zuklappen. Ein kleiner Punkt vor einem Eintrag zeigt dir, dass dort nichts mehr auf- oder zuzuklappen ist.

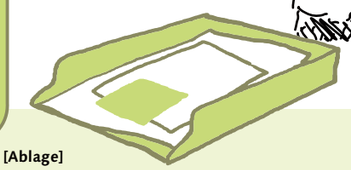
Die Operationen, die für ein Entwicklungsobjekt und den Teilobjekten in der Objektliste zur Verfügung stehen, kannst du einfach über das **Kontextmenü** (rechte Maustaste) aufrufen.

[Einfache Aufgabe]
In der Drucktastenleiste gibt es auch einen Button, der Operationen anbietet. Versuche mal, ihn zu finden.

Gefunden! Die Drucktaste Anderes Objekt (⇧ + F5).



Hast du heute schon dein täglich Programm angelegt?



[Ablage]
Falls dir die **Breite der Anzeige** im **Navigationsbereich** nicht ausreichend oder zu groß ist, kannst du die Breite durch das **Verschieben** der **Grenze des Navigationsbereiches** verändern.

Leg! Jetzt! An!
(mit scharfem Unterton, wie aus der Werbung bekannt)

[Einfache Aufgabe]
Rahmen mit linker Maustaste anklicken, halten und nach links oder rechts verschieben.
Schieb zehnmal schnell hin und her.
Der Rekord liegt bei 5 Sekunden, 31 Hundertstel.

Die Spaltenbreiten in der Objektliste kannst du ebenfalls einfach verändern: am Spaltenrand **click – move – leave.** Übrigens kannst du den **Navigationsbereich** komplett **ausblenden**. Dafür gibt es wieder eine Drucktaste in der Drucktastenleiste (**Vollbild ein/aus** (⇧ + F12)).

Welche war das nur, hmmm? Und wie kann man den Navigationsbereich wieder einblenden?

Jetzt kommt der eigentlich wichtige, essenzielle und alles verändernde **höchst informative Kernabschnitt** dieses Kapitels. Willst du vorher noch eine theatralische Pause einlegen?

Niemals!

Also gut. Bitte jedes Wort einzeln lesen und kein Speedreading.

Wenn du **doppelt auf ein Objekt** klickst, werden die Details zum Objekt rechts vom Navigationsbereich, im **Werkzeugbereich**, mit dem passenden **Entwicklungswerkzeug** dargestellt. Also zum Beispiel für ein Programm der Programmtext. Im Werkzeugbereich stehen die Funktionen für das Objekt über das Kontextmenü oder die **Oberfläche** (Menüleiste, Symbolleiste, Drucktastenleiste) zur Verfügung.

Das war's?

[Ablage]
Ja, das war's. Mit dem Doppelklick – du wirst auch **Vorwärtsnavigation** dafür hören – kannst du von jedem Objekt in der Objektliste zum passenden Werkzeug springen, ohne zu wissen, welches Objekt du tatsächlich benötigst. Einfach genial!

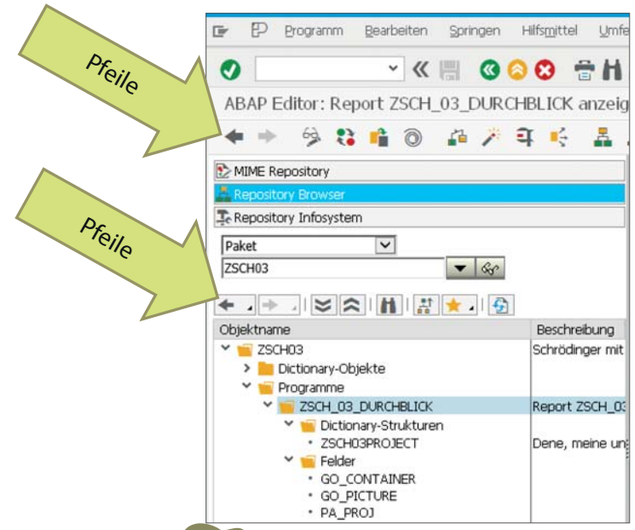
[Einfache Aufgabe]
Probiere den Doppelklick bitte aus. Für das Programm. Für ein Feld. Für eine DDIC-Struktur. Für ein Dynpro. Du wirst immer woanders landen.

DDIC?

Ach, DDIC hab ich ja noch gar nicht erklärt. Das Data Dictionary – dafür steht die Abkürzung – ist eine alternative Bezeichnung fürs ABAP Dictionary. DDIC war die bevorzugte Bezeichnung, bevor das Java Dictionary ins Spiel kam.

Sowohl für den Navigationsbereich als auch für den Werkzeugbereich stehen **Navigationspfeile** (die blauen) zur Verfügung. Damit kannst du wie in einem Webbrowser in der **Navigationshistorie** vor- und zurückwandern.

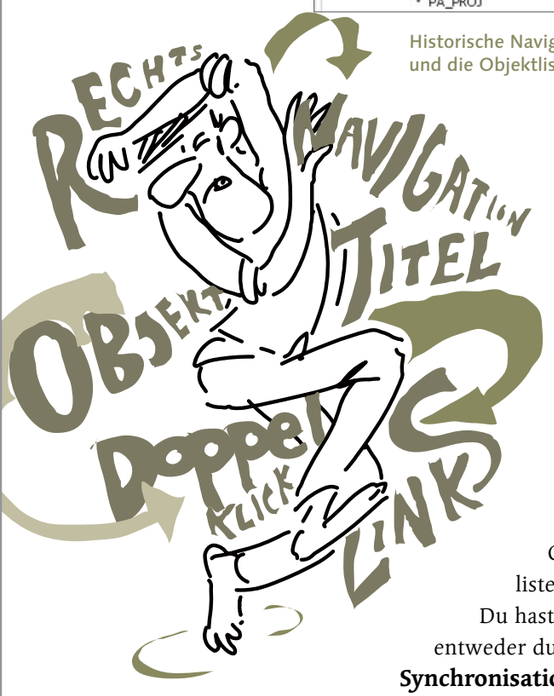
[Achtung]
Aber Achtung: Die Pfeile in der Drucktastenleiste betreffen das Werkzeug, und die Pfeile im Repository Browser betreffen die Objektlisten. Das sind zwei unterschiedliche Dinge!



Historische Navigation für das Werkzeug und die Objektliste. Unabhängige Navigation inklusive!

Synchron oder aus dem Tritt – Objekt-Browser-Synchronisation

Durch diese unabhängige Navigation kann es natürlich vorkommen, dass der **Inhalt des Navigationsbereichs** und der **Inhalt des Werkzeugbereichs** nicht zusammenpassen.



[Achtung]
Aufpassen. Hier kann es schon mal vorkommen, dass du nur in die Objektliste schaust, der Annahme bist, dass du ein bestimmtes Objekt bearbeitest und sich nach geraumer Zeit herausstellt – weil du den Titel im Werkzeugbereich nicht kontrolliert hast –, dass du ungewollt ein anderes Objekt bearbeitet hast!

Passiert mir nie. Ich kann doch lesen!

Offensichtlich, und wir werden sehen!

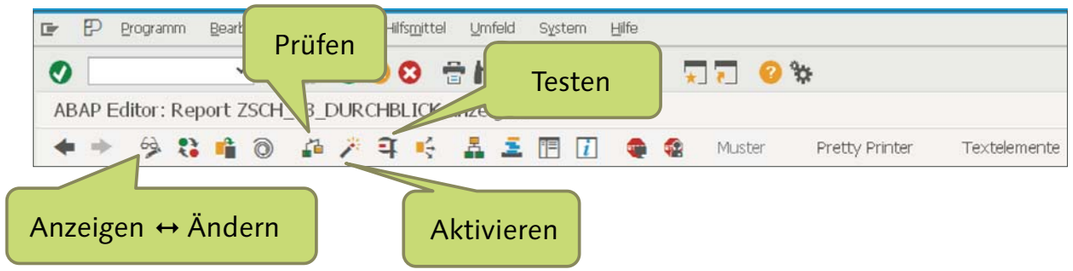
Glücklicherweise kannst du den Werkzeugbereich und die Objektliste synchronisieren.

Du hast zwei Richtungen, um eine **Synchronisation** durchzuführen, entweder du klickst auf ein Objekt aus der Objektliste (**Links-nach-rechts-Synchronisation**), oder du drückst die Drucktaste **Objektliste Anzeigen** (Strg + ⇧ + F5) und machst darüber die **Rechts-nach-links-Synchronisation**. Schon sind die zwei Bereiche wieder in harmonischem Einklang.

Eines vorweg, bevor wir uns die üblichen Verdächtigen der Entwicklungswerkzeuge ansehen.

Die Gemeinschaft der vier Tasten.

Im Werkzeugbereich sind vier Tasten in der Drucktastenleiste von zentraler Bedeutung: die **Anzeigen ↔ Ändern**-Taste (Strg + F1), die **Prüfen**-Taste (Strg + F2), die **Aktivieren**-Taste (Strg + F3) und die **Testen/Ausführen**-Taste (F8).



Mit der Taste Anzeigen ↔ Ändern kannst du den Bearbeitungsmodus ändern. Mit Prüfen wird die syntaktische Korrektheit geprüft. Mit Aktivieren änderst du den Zustand des Objektes von inaktiv auf aktiv. Und mit Testen kannst du das angezeigte Objekt ausführen und ausprobieren.

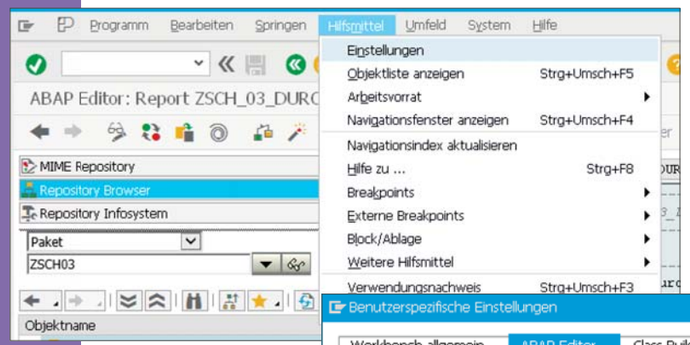
Und zum Schluss noch ein Anwendungsszenario: Die **Anlage eines neuen Objektes** erfolgt entweder über die **Vorwärtsnavigation** aus dem Navigationsbereich heraus (Namen eingeben und mit der **Enter**-Taste bestätigen, den Rest erledigt das System) oder über die Drucktaste **Anderes Objekt** (⇧ + F5) in der Drucktastenleiste, da musst du mehr nachdenken.

Für die unterschiedlichsten Entwicklungsobjektkategorien stehen unter **Erweiterte Optionen** Karteireiter zur Verfügung, die dir auch ermöglichen, Teilobjekte zu bearbeiten. Wenn du zum Beispiel ein Programm anlegen möchtest, gibst du den Namen des Programms im Feld (bitte ausfüllen) ..... ein und drückst danach die Anlegen-Taste (F5). Das ist die Drucktaste mit dem weißen Blatt. Das Symbol kannst du dir auch merken, da es sehr oft vorkommt.

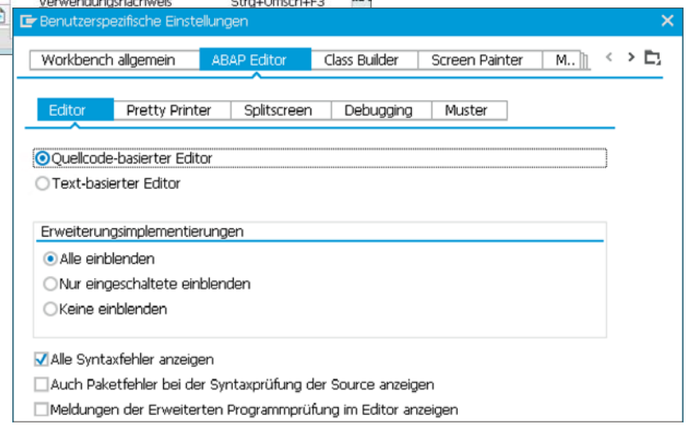
ABAP Editor – Die Schreibmaschine

Das wird wohl dein bester Freund werden, der **ABAP Editor**. Falls mal jemand **SE38** zu ihm sagt, hat der sich als jemand geoutet, der seit mindestens 100 Jahren mit dem SAP-System programmiert.

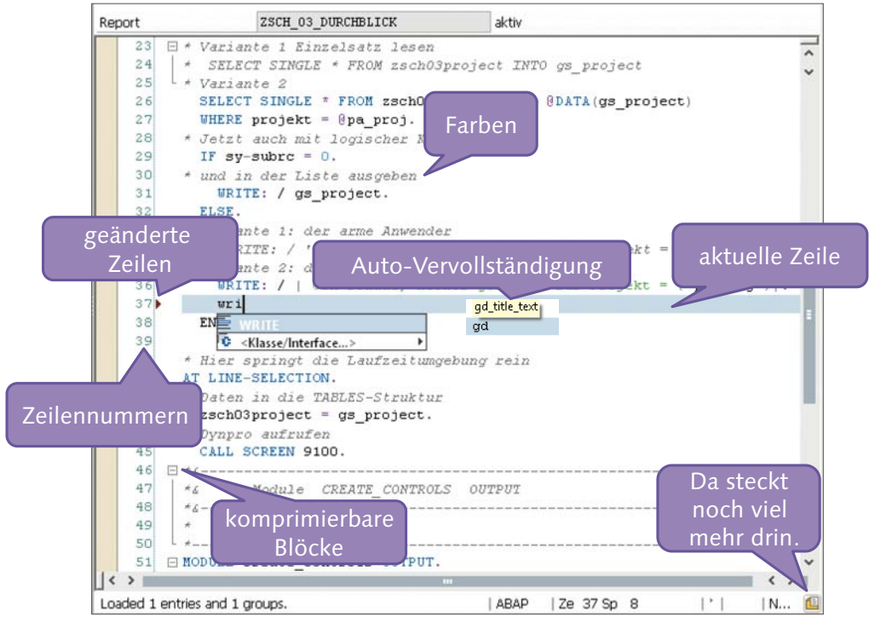
Der Editor kommt je nach SAP-Release in unterschiedlichen/mehreren Versionen daher. Du kannst ganz einfach feststellen, woran du bist: **Hilfsmittel • Einstellungen • ABAP Editor**.



Welchen Editor hätten Sie denn gerne?

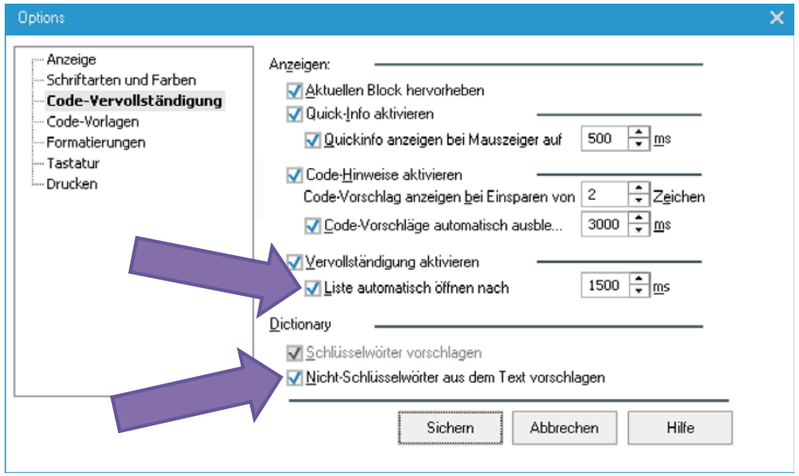


Natürlich verwenden wir den **Quellcode-basierten Editor**, der kann naturgemäß am meisten, übrigens ab SAP NetWeaver 7.0. Außerdem ist er schön bunt, hebt zum Beispiel Schlüsselwörter farbig hervor, kontrolliert automatisch die Syntax usw.



Eine Auswahl der schärfsten Features.

Du kannst Farben definieren, Schriftarten einstellen, Blöcke komprimieren/expandieren, die Zeilennummern ablesen, auf die Vervollständigung hoffen (wenn dir etwas vom System vorgeschlagen wird, drücke die **Tab**-Taste, um die Vervollständigung durchzuführen) und eigene Einstellungen vornehmen. Dazu musst du den Button rechts unten im Editor anklicken, dann erscheint das **Einstellungsfenster**.

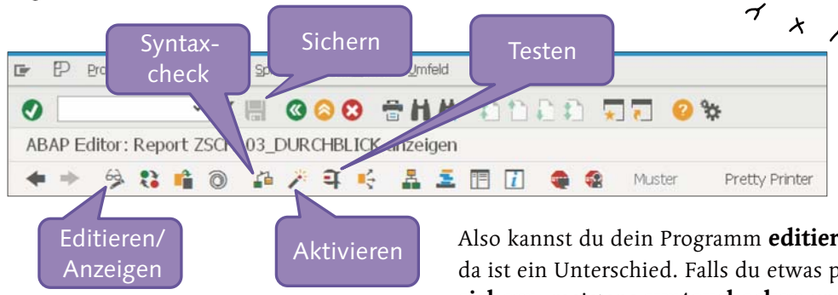


Vervollständige dich.

Das **Einstellungsfenster** bietet dir ziemlich viele Einstellungen, wie du sehen kannst. Für den Moment sind die beiden Checkboxen **Liste automatisch öffnen nach** und **Nicht-Schlüsselwörter aus dem Text vorschlagen** aus meiner Sicht die wichtigsten, um die Verwendung der Vervollständigung optimal nutzen zu können. Also setzen! Nicht dich, die Checkboxen!

Gesetzt! Im Einstellungsfenster sehe ich aber schon noch ein paar nette Features. Zum Beispiel Code-Vorlagen.

Das stimmt, die kannst du sogar selbst erstellen. Aber das ist eine andere Geschichte. Wichtiger für die Arbeit mit dem Editor sind die **Funktionen**, die dafür zur Verfügung stehen und auf der Oberfläche angeboten werden.



Funktionen über Funktionen.

Also kannst du dein Programm **editieren** und es **anzeigen**. Ja, da ist ein Unterschied. Falls du etwas programmierst, wirst du es **sichern**, meistens **syntaxchecken** und **testen**. Falls du es für gut befindest, kannst du es noch **aktivieren**, damit die anderen Anwender im System deine geniale Funktionalität ebenfalls verwenden können.

Die Tasten **Muster** und **Pretty Printer** wirst du auch oft verwenden, jetzt, wo ich es dir sage. Mit der Drucktaste **Muster** kannst du dir Coding-Teile produzieren lassen.

Hat das nicht die Code-Vervollständigung auch gemacht?

Du hast Recht. Ich würde sagen, Muster ist der Opa der Code-Vervollständigung. Aber sehr nützlich, da es doch einige Unterschiede gibt.

Die **Pretty Printer**-Taste ist zum Behübschen deiner Implementierung. **Einrückungen** und **Groß- und Kleinkonvertierung** sind ihre Stärken. Ich stelle gerne mit dem Menüeintrag **Hilfsmittel • Einstellungen • ABAP Editor • Pretty Printer** die Schreibweise **Schlüsselwort groß** ein, damit ich nach dem Drücken der Drucktaste **Pretty Printer** sofort sehe, ob ich mich vertippt habe (das nennt sich **Poorman's Syntax Check**).

[Einfache Aufgabe]

Leg das neue lokale Programm `ZSCH_03_EDICHECK` an. Nutze dazu das Kontextmenü auf dem Paket `$TMP`. Lasse es im Werkzeugbereich anzeigen. Füge die Anweisung `WRITE "Hi Edi".` ein. Sichere das Programm. Prüfe das Programm. Korrigiere das Programm. Pretty printe das Programm. Aktiviere das Programm. Teste das Programm.

[Belohnung]

Zeit für eine Kreativpause. Überlege dir einen Reim mit den folgenden Wörtern: Paket, Sichern, Programm, Korrigieren, Aktivieren, Testen, und irgendwo sollte „Ich bin ein Checker" vorkommen.

Viel Spaß.

Debugger – Der Kammerjäger

Der **Debugger** (Entwanzer) ist dein Freund für die **dynamische Analyse**. Seinen Bruder hast du ja bereits kennengelernt: den **Syntaxchecker**. Der ist aber ziemlich statisch.
Der Unterschied zwischen den ungleichen Brüdern aus der Familie Analyse ist der folgende: Die **statische Analyse** (Bruder Syntaxchecker) stellt sicher, dass keine Syntaxfehler vorhanden sind und das Programm ausführbar ist. Leider können dann aber logischerweise immer noch inhaltliche Fehler im Programm enthalten sein, wie zum Beispiel falsche Berechnungen etc.
Dafür ist nur der Bruder Debugger (**dynamische Analyse**) zuständig. Mit ihm kannst du das Programm inhaltlich analysieren.
Und da das total wichtig ist, sehen wir uns das jetzt mal kurz an.

Das Vorgehen zur Analyse ist das folgende:

1. Debugger starten
2. Programm mit Debugger-Unterstützung analysieren

Eine Möglichkeit, den Debugger zu starten, ist die Eingabe von „**/h**" im Transaktionscode-Feld und die Bestätigung mit der **Enter**-Taste. Das **h** kommt von **hoppeln**, also wie ein Hase durch das Programm springen.

[Hintergrundinfo]

Der Start mit **/h** ist besonders dann geeignet, wenn du ab einem gewissen Zeitpunkt während der Programmausführung in das Debugging wechseln möchtest.

Für dich als Debugging-Neuling ist die folgende Startvariante

die Einstiegsdroge:

Der Debugger-Start wird durch das **Setzen eines Breakpoints im Programm** ausgelöst. Setze dazu einen Breakpoint, der auch als **unbedingter Haltepunkt** bezeichnet wird, im Programm, und starte es. Der Debugger wird beim Erreichen des Breakpoints automatisch gestartet.

Noch mal schön langsam, bitte schön.

Debugger entschleunigt

Ich habe da mal ein kleines Programm vorbereitet, das uns die Zeit in zwei Stunden berechnet. Aufgrund deiner Erfahrung aus Kapitel 3 sollte das Programm einfach zu verstehen sein, dennoch werde ich meinen Senf dazugeben.

*1 Der Standard-Ereignisblock, der vom Laufzeitsystem angesprungen wird. Hier startet die Abarbeitung in unserem Programm.

*2 Mit **WRITE** geben wir eine Zeile aus, und zwar einen Text und den Inhalt des Feldes **sy-uzeit**, also die aktuelle Zeit. Ich habe zur Ausgabe wieder die String Templates verwendet und für die Aufbereitung der Zeit ein Ausgabeformat gesetzt. Die Zeit wird damit so ausgegeben, wie der User es für sich in den Benutzerparametern eingestellt hat.

```
REPORT zsch_03_inzweistundenistes.
* Party on
START-OF-SELECTION. *1
* Jetzt
  WRITE: | Jetzt ist es: { sy-uzeit TIME = USER }|. *2
* In zwei Stunden ist es ...
* Zeitarithmetik
  DATA(gd_inzweistunden) = CONV t( sy-uzeit + 2 ). *3 und *4
* und in zwei Stunden
  WRITE: / | Und in zwei Stunden ist es: { gd_inzweistunden
  TIME = USER }|. *5
```

*4 Jetzt kommt eine richtig komplexe Berechnung. Ich zähle zur aktuellen Systemzeit zwei Stunden dazu und speichere das Ergebnis in der Variablen **gd_inzweistunden**.

*3 Mit der Inline-Deklaration **DATA (gd_inzweistunden)** deklarierst du die Variable **gd_inzweistunden**, die mit einem „Zeittyp" deklariert ist. Der wird durch die Anweisung **CONV t( )** vorgegeben. Den Typ t kennst du als eingebauten ABAP-Typ für Zeit. Du kannst da aber beinahe jeden anderen Typ verwenden. **sy-uzeit** ist ein wichtiges Systemfeld, das dir die Zeit am Applikationsserver liefert. Es hat das Format Stunden:Minuten:Sekunden, jeweils zweistellig.

*5 <Füge bitte hier die Erläuterung ein> .....

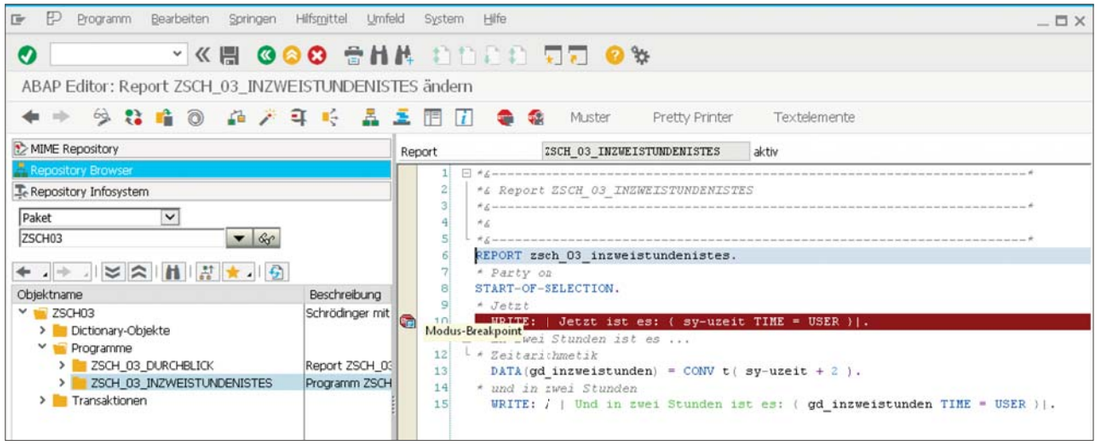
Wunderbares Programm.

Sichern, prüfen, aktivieren, testen. Wie sieht dein Ergebnis aus?

Da stimmt doch etwas nicht? Zwei Stunden nach 17:42:31 sollte doch 19:42:31 sein und nicht 17:42:33? Was ist hier falsch?

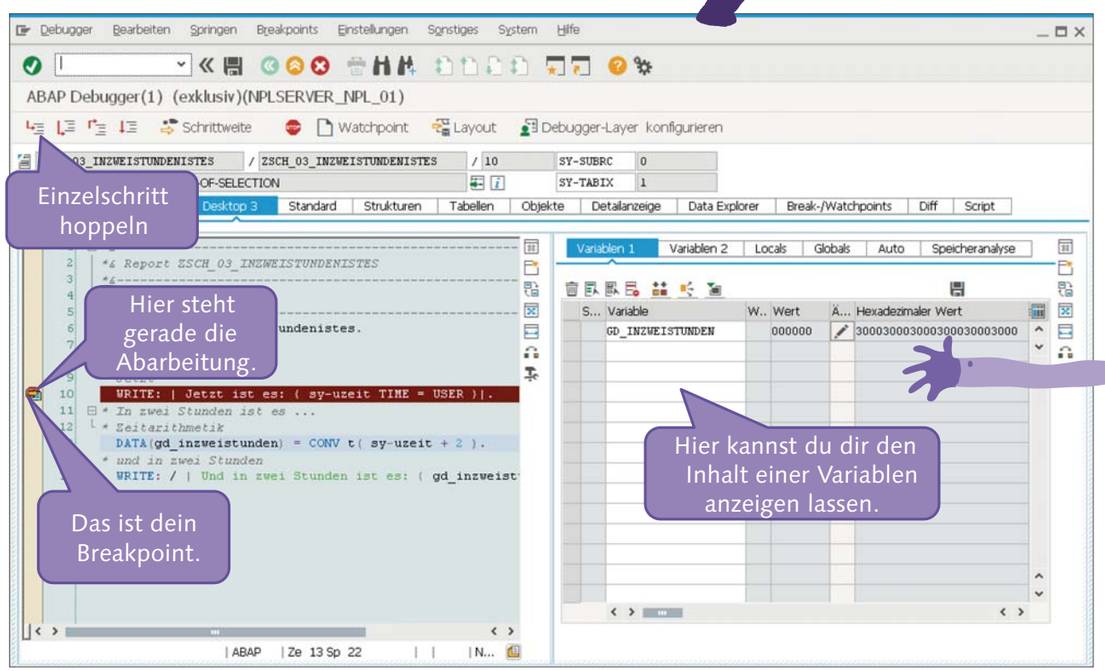
[Fehler/Müll]

Nachdem du das Programm sicher 1.000 Mal getestet hast, stellst du fest, dass die 2 nicht zu den Stunden, sondern zu den Sekunden hinzugezählt werden. Glauben oder nicht, oder doch lieber analysieren?



Im ABAP Editor klickst du ganz einfach in die braune Spalte in der Zeile, ab der du gerne analysieren möchtest. Durch den Klick wird ein Symbol eingefügt. Das Breakpoint-Symbol, das dir anzeigt: Hier ist mal ein Zwischenhalt.

Jetzt wird es ernst. Teste das Programm!



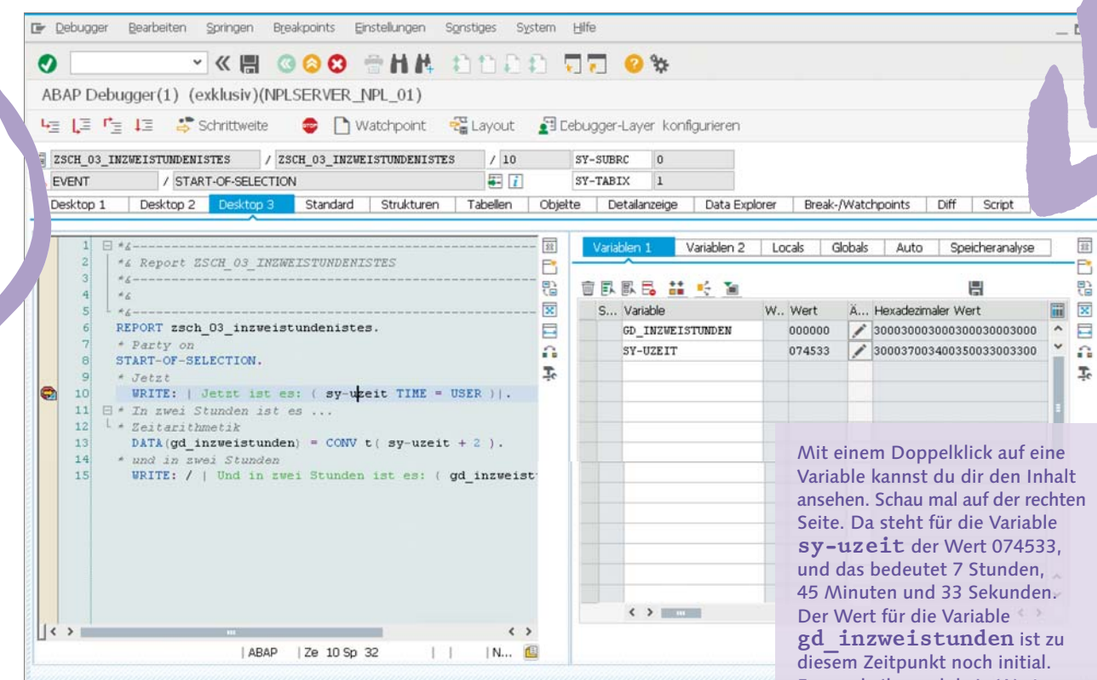
Der Debugger sieht komplizierter aus, als er ist. Für den Moment reichen uns einige wenige Funktionen: Einzelschrittnavigation und die Inhaltsanzeige von Variablen.

Mit dem **gelben Pfeil** in der braunen Spalte wird dir angezeigt, wo die Abarbeitung des Programms gerade steht. Wenn du die Taste **Einzelschritt** (F5) drückst, führt das System die aktuelle Zeile aus und springt zur nächsten auszuführenden Zeile.

[Einfache Aufgabe]
Einen Sprung, bitte!

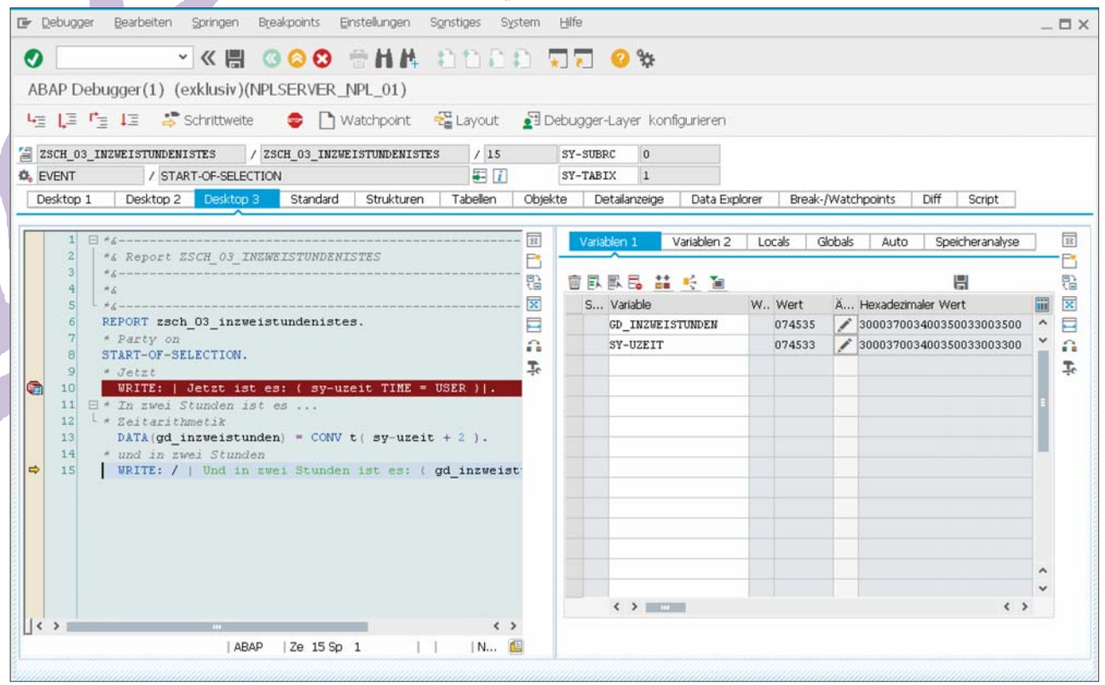
Jetzt bin ich beim **WRITE** in Zeile 10 angekommen.

Sehr gut, klick doch mal doppelt auf das Feld **sy-uzeit**.



Mit einem Doppelklick auf eine Variable kannst du dir den Inhalt ansehen. Schau mal auf der rechten Seite. Da steht für die Variable **sy-uzeit** der Wert 074533, und das bedeutet 7 Stunden, 45 Minuten und 33 Sekunden. Der Wert für die Variable **gd_inzweistunden** ist zu diesem Zeitpunkt noch initial. Es wurde ihr noch kein Wert zugewiesen.

Hey, das ist ja cool. Da kann ich mir im Zeitraffer die Programmausführung ansehen. Ich bin gleich zweimal weitergesprungen. Dadurch ist die hochwissenschaftliche Berechnung ausgeführt worden.



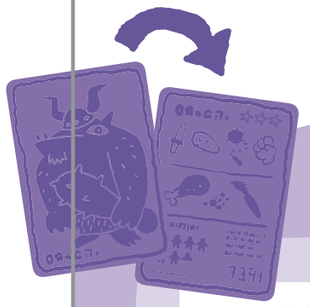
Da ist er, der Übeltäter. Es werden tatsächlich nur zwei Sekunden dazugerechnet.

Da ist der Fehler.

Die dynamische Analyse hat es gezeigt und aufgedeckt. Wenn du tatsächlich zwei Stunden addieren möchtest, musst du was tun?

Der Roland Schwaiger hat uns wieder mal reingelegt. Mithilfe des Debuggers wurde es entlarvt. Berechnungen mit Zeit geschehen auf Basis von Sekunden. Ha!

Wenn ich zwei Stunden addieren möchte, muss ich die zwei Stunden in Sekunden umrechnen. Das mache ich aber nicht, ist langweilig.



War natürlich unabsichtlich. Eine kurze Zusammenfassung:

- Das **Hoppeln** durch den Quelltext im Debugger kannst du unter anderem in **Einzelschritten** durchführen.
- Einzelne Objekte kannst du durch einen Doppelklick in die **Detailanzeige** übernehmen und dort die **Inhalte analysieren und ändern**.

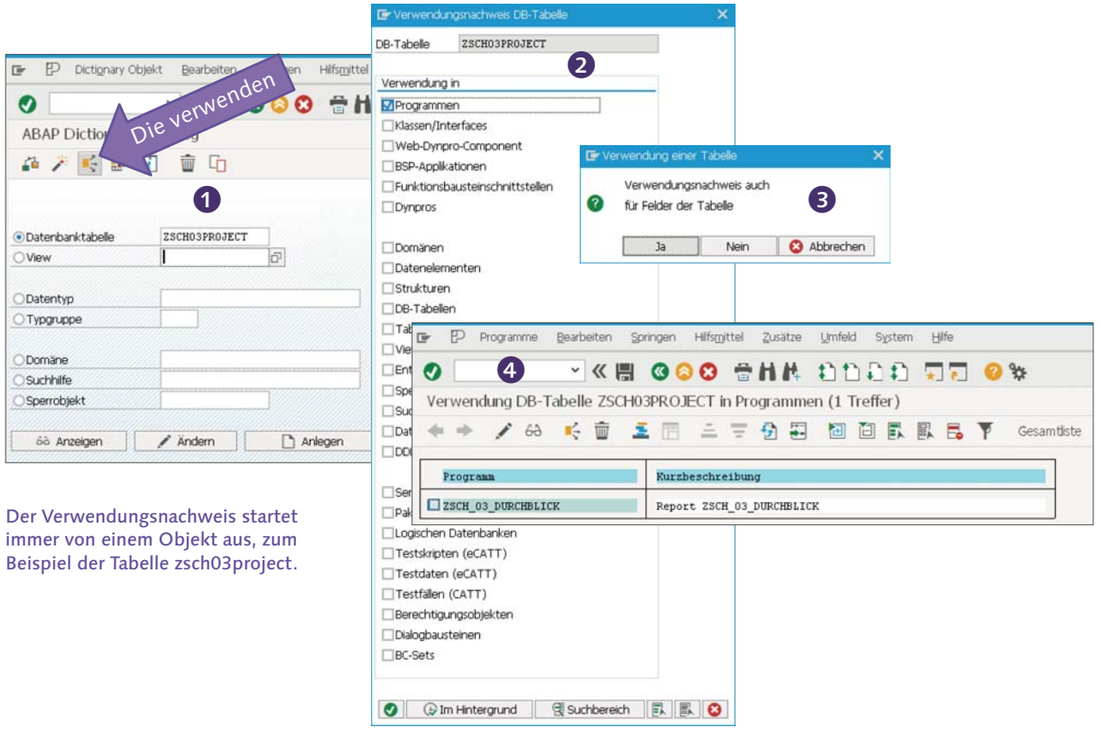
[Hintergrundinfo]
Vor SAP Web Application Server 6.40 hat es nur den **klassischen** (klassisch, hihihi, cooles Wort für ALT) **Debugger** gegeben, der im **selben externen Modus** abläuft wie das zu analysierende Programm. Nach 6.40 steht dir auch „DER Neue" Debugger zur Verfügung, der in einem **eigenen externen Modus** abläuft.

Das gewonnene Wissen wird dir für die nächsten Kapitel sicher hilfreich sein. Du wirst schon sehen.

[Belohnung]
Pause? Turnübung? Liegestütz, oder doch unter den Tisch reinhängen und hochziehen?

Verwendungsnachweis – Der Rückwärtssucher

Auf ein besonders praktisches Feature möchte ich dich noch zusätzlich hinweisen: den **Verwendungsnachweis** (Strg + ⇧ + F3), Kästchen mit drei Pfeilen. Bis jetzt habe ich dir mehrfach geschildert, dass du per Doppelklick auf ein bestimmtes Entwicklungsobjekt zu dessen Definition springen kannst. Das ist einfach und eindeutig.
Jetzt drehen wir mal den Spieß um und gehen von der Definition aus. **Praktisch rückwärts**. Wir könnten also fragen:
Wo wird dieses Entwicklungsobjekt überall verwendet? Welche anderen Entwicklungsobjekte verwenden dieses Entwicklungsobjekt? Den Suchbereich kannst du zusätzlich einschränken.
Ist das nicht toll? Super mächtiges, total oft verwendetes Feature!
VERWENDUNGSNACHWEIS!
Gleich mal praktisch. Wo wird die Tabelle `zsch03project` verwendet?



Der Verwendungsnachweis startet immer von einem Objekt aus, zum Beispiel der Tabelle zsch03project.

1. Rufe die SE11 auf, und gib den Namen der Tabelle **`zsch03project`** ein.
2. Mit der Drucktaste **Verwendungsnachweis** (Strg + ⇧ + F3) startest du die Suche. Das System hätte ganz gerne von dir ein wenig mehr Infos, wo es suchen soll. Zum Beispiel nur in den Programmen. Mit **Ausführen** (Enter) kannst du die Suche starten.
3. Spezialfall Tabelle: Das neugierige System möchte auch noch wissen, ob neben der Tabelle selbst auch die Verwendung der Felder gesucht werden soll. Probiere jetzt mal **Ja**, und dann später die **Nein**-Variante.
4. Tataaaa, da ist das Ergebnis. In genau einem Programm wird die Tabelle **`zsch03project`** verwendet.

[Einfache Aufgabe]
Versuche die **Nein**-Variante, also nur für die Tabelle selbst, aber nicht für die Felder.
Ist die Treffermenge unterschiedlich?

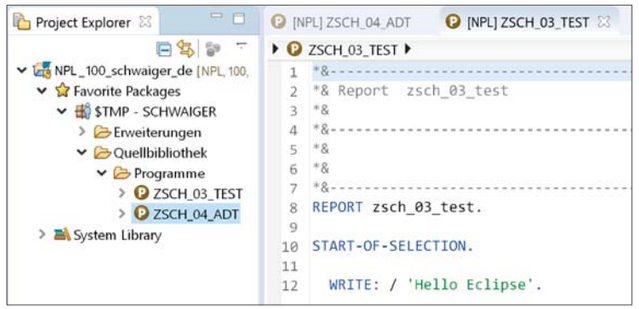
Ende der Vorstellungsrunde. Die anderen üblichen Verdächtigen wirst du in späteren Kapiteln noch kennenlernen: den **Class Builder** für die Objektorientierung, den **Function Builder** für Funktionsbausteine, das **ABAP Dictionary** für globale Typen, den **Screen-Painter** für Fensterprogrammierung.

Dein Job ist nun, die Augen zu schließen, nochmals die Bilder unserer Ausstellung vor deinem inneren Auge von links nach rechts oder rechts nach links vorbeigleiten zu lassen und dann ein Stück meines Apfelkuchens zu genießen.

Apfel, gedeckt! Ich sehe, du hattest schon.

Die ADT können das auch

Das Jüngste im Bunde der ABAP-Entwicklungswerkzeuge hat es faustdick hinter den Ohren. Schauen wir mal, was unser kleiner Prinz so zu bieten hat. Bist du in Eclipse immer noch am System angemeldet? Sonst blättere kurz zurück zu Kapitel 3.
Du kannst für die folgenden Schritte irgendein Programm im Project Explorer nehmen. Auf geht's im Sauseschritt durch die **Funktionen der ADT**. Da gibt's eine Menge Analogien zur SE80. Du kannst ein Programm anlegen, ändern, speichern, aktivieren und debuggen.



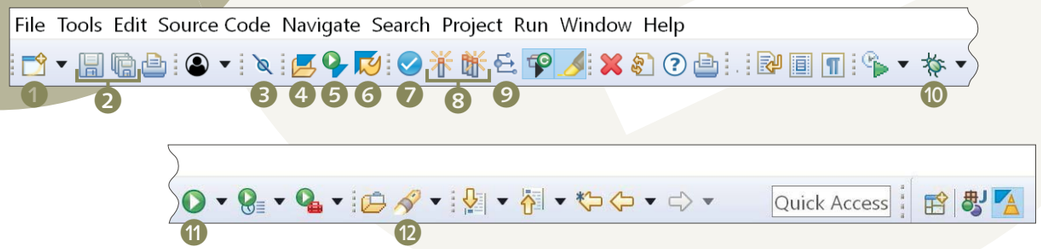
Mit einem Doppelklick auf ein Entwicklungsobjekt, zum Beispiel das Programm **ZSCH_03_TEST**, kannst du den Werkzeugbereich rechts öffnen. Ich habe auch gleich noch das zweite Programm **ZSCH_04_ADT** angelegt – ich zeige dir gleich, wie das geht – damit **ZSCH_03_TEST** nicht so alleine ist.

Dann kann ich also mehrere Werkzeuge gleichzeitig offen haben? Sehr praktisch!

Super praktisch! Links in der Objektliste ist das Programm **ZSCH_04_ADT** markiert. Im Werkzeugbereich sind zwei Registerkarten geöffnet, aber nur die für das Programm **ZSCH_03_TEST** ist gerade aktiv. Direkt unter dem Reiter für das Programm wird eine **Breadcrumb**-Navigation eingeblendet. Damit kannst du zu den Teilobjekten des Entwicklungsobjekts navigieren.

[Zettel]
Wenn du doppelt auf einen Reiter klickst, wird der Werkzeugbereich maximiert. Wenn du noch mal klickst, schrumpft er wieder auf Normalgröße.

[Hintergrundinfo]
Noch kurz zu den Symbolen auf dem Tab:
- ☛ Wenn ein Sternchen vor dem Titel angezeigt wird, ist das Programm nicht aktiviert.
- ☛ Wenn ein Schloss mit Raute vor dem Titel angezeigt wird, ist das Programm gesperrt und nicht aktiviert.

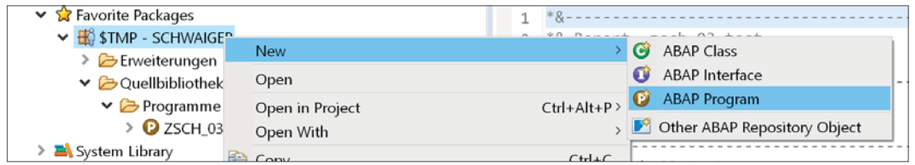


Festhalten, jetzt kommen gaaanz viele Funktionen. Die findest du alle in der Toolbar. Ich erkläre dir die wichtigsten.

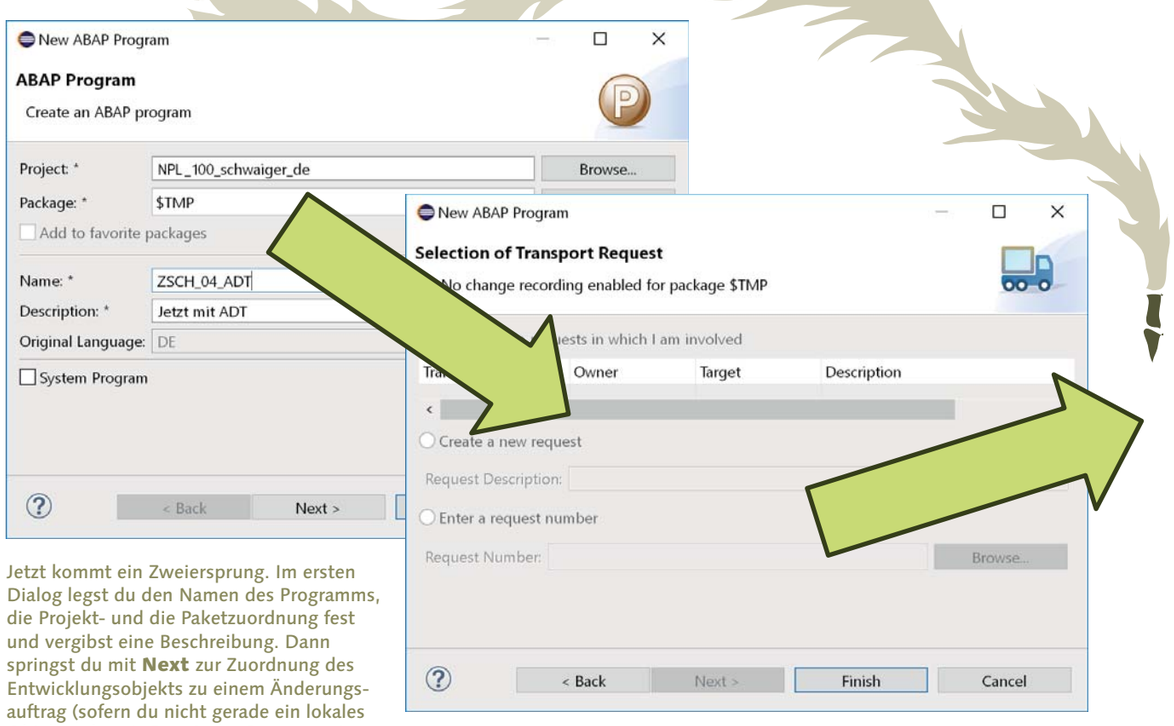
1. **Neuanlage** eines Projekts, eines ABAP-Programms, einer ABAP-Klasse oder was immer du willst
2. **Sichern** des Entwicklungsobjekts der aktiven Registerkarte bzw. Sichern aller Entwicklungsobjekte, die noch nicht gesichert wurden (nur zur Sicherheit …)
3. Diese Funktion ist im Zusammenhang mit dem Debugging wichtig – falls du im Debugger nicht immer bei den Breakpoints stehen bleiben möchtest, sondern das Programm einfach mal ohne Unterbrechung laufen lassen möchtest. Ein temporärer **Breakpoint-Eliminator**!
4. **Öffnen** eines ABAP-Entwicklungsobjekts. Nach den Objekten kannst du zum Beispiel über ihren Namen oder auch nur Teilen des Namens suchen. Die ADT füllen deine Gedächtnislücken.
5. **Ausführen** des Entwicklungsobjekts im SAP GUI, das in Eclipse gerade dargestellt wird
6. Öffnen des **SAP GUI**
7. Das ist dein **Syntaxchecker**.
8. **Aktivieren** des aktuellen Objekts oder aller Objekte
9. der (g/b)eliebte **Verwendungsnachweis**, um alle Stellen zu finden, an denen das aktuelle Objekt angesprochen wird
10. der **Debugger** zum Aufstöbern von Fehlern während der Ausführung, die der Syntaxchecker vorher nicht finden konnte
11. **Ausführen** des Programms
12. **Suchen** nach Entwicklungsobjekten

Mit dem meisten kann ich was anfangen, ist ja wie in der SE80.

Super, dann spielen wir jetzt mal das Anlegen eines Programms durch. Als kleine Aufgabe kannst du das Programm **ZSCH_04_ADT** anlegen. Starte am besten mit dem Kontextmenü eines Pakets, zum Beispiel dem lokalen Paket **$STMP**.

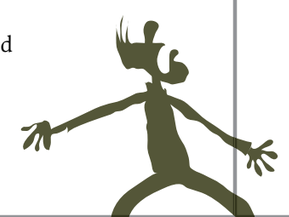


Wähle **New • ABAP Program**



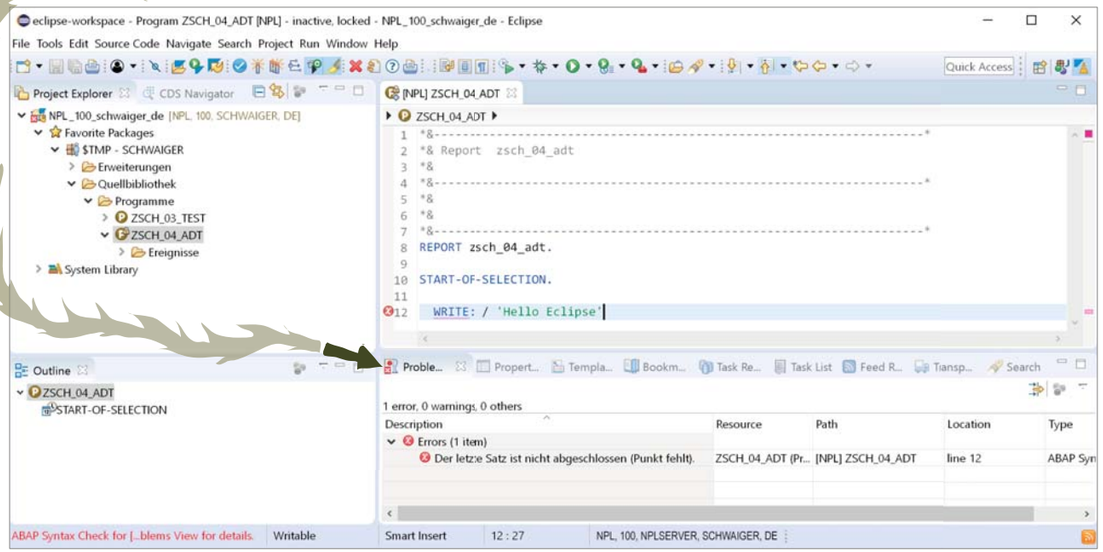
Jetzt kommt ein Zweiersprung. Im ersten Dialog legst du den Namen des Programms, die Projekt- und die Paketzuordnung fest und vergibst eine Beschreibung. Dann springst du mit **Next** zur Zuordnung des Entwicklungsobjekts zu einem Änderungsauftrag (sofern du nicht gerade ein lokales Objekt anlegst). Da wir das Programm im lokalen Paket $TMP angelegt haben, wird uns kein Änderungsauftrag angeboten.

Jetzt nur noch mit **Finish** abschließen, und schon kannst du dir die Finger wund programmieren. Der Editor ist zuvorkommenderweise bereits für dich geöffnet.



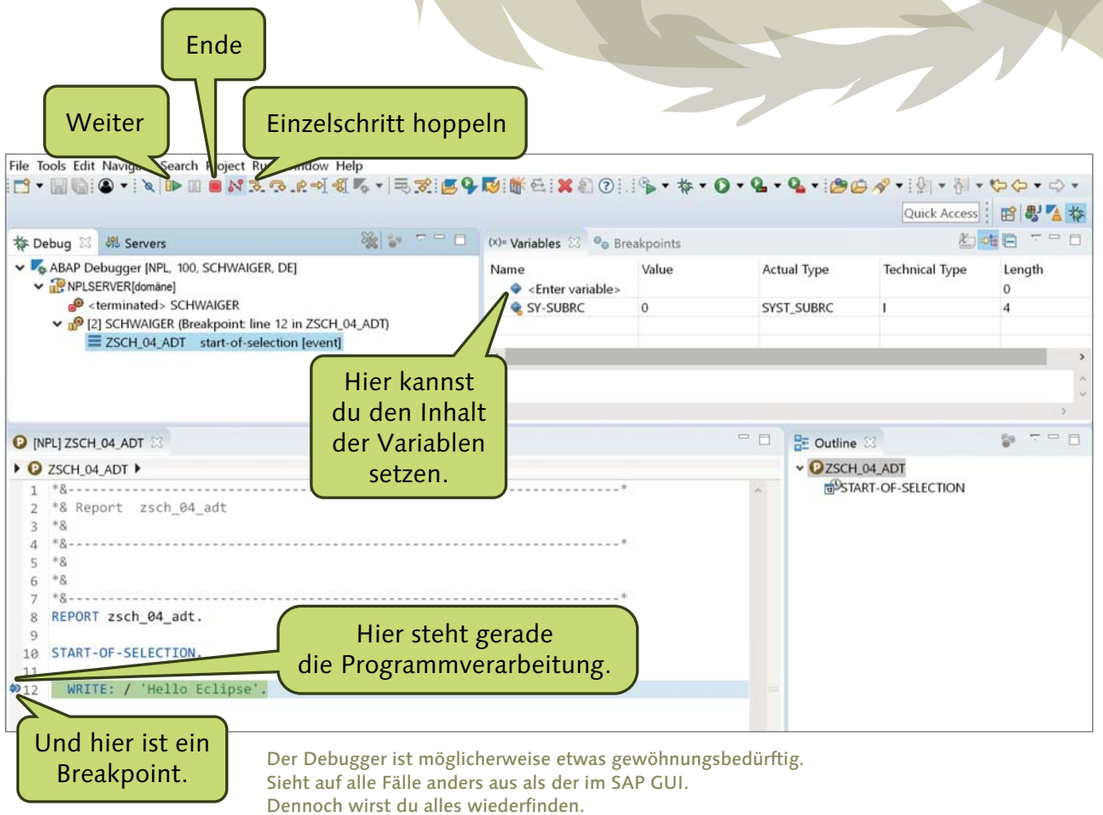
Es folgen einige sachdienliche Hinweise zur Bearbeitung der ABAP-Programme im Eclipse-Editor:

- Wenn du eine **Auto-Vervollständigung** wünschst, drücke Strg + die Leertaste. Prinz Editor liefert dir stets passende Vorschläge.
- Mit ⇧ + F1 kannst du deinen Code **pretty printen**, also formatieren.
- Über die bereits besprochenen Symbole in der Toolbar kannst du dein Programm **prüfen, testen und aktivieren**.
- Ein rotes X vor einer ABAP-Zeile bedeutet nichts Gutes, sondern ist der Indikator für einen Fehler. Hoppla, Fehler sollte man aus didaktischen Gründen nicht sagen. Darum nennt sie Eclipse auch **Probleme** und zeigt diese unter dem Editor-View an.



Da gibt es einen Fe … äh … ein klitzekleines Problemchen.
Eigentlich kaum der Rede Wert.
Solltest du dir trotzdem ansehen.

Alles, was der Syntaxchecker nicht als Problem erkannt hat, kann möglicherweise der **Debugger** entlarven. Um einen Breakpoint zu setzen, klickst du einfach auf den Balken vor der Zeile, in der du halten möchtest, und dann führst du das Programm aus. Bei deinem ersten Debugging-Versuch wird ein Pop-up dich fragen, ob es die Debugging-Perspektive öffnen soll. **Yes!** Das ist voll in deinem Sinne!



Der Debugger ist möglicherweise etwas gewöhnungsbedürftig.
Sieht auf alle Fälle anders aus als der im SAP GUI.
Dennoch wirst du alles wiederfinden.

Nach dem Aufruf steht der Debugger bereits an der Stelle, die du mit dem Breakpoint markiert hast. Rechts oben kannst du die Werte der Variablen verändern. Falls du dich für eine bestimmte Variable interessierst, tippe einfach ihren Namen im Feld **<Enter variable>** ein. Den neuen Wert schreibst du direkt in die Spalte **Value**. Im Debugger hoppelst du entweder in Einzelschritten weiter oder mit der **Weiter**-Taste gleich bis zum nächsten Haltepunkt.

Danke, Roland, aber ich mag jetzt nicht mehr. Meine Programme sind fabelhaft fehlerfrei. Die Essiggurken habe ich auch schon alle aufgegessen. Keine mehr im Glas!