

Roland Schwaiger



PRESS

Inkl.
Downloads

Mit Syntax-
Highlighting!

Schrödinger programmiert ABAP

Das etwas andere Fachbuch

- Von den ersten Schritten bis zum kompletten Programm
- Topaktuell zu S/4HANA und Fiori:
das beste ABAP aller Zeiten

- Umwerfende Beispiele, fantastisch illustriert

DRITTE AUFLAGE



Rheinwerk
Computing

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

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,

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

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



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 Datenbanktabellen-definition für dich erstellt. Sind die Inhalte mandantenabhängig oder mandantenunabhängig?



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

To be dependent or not to be ...

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

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.



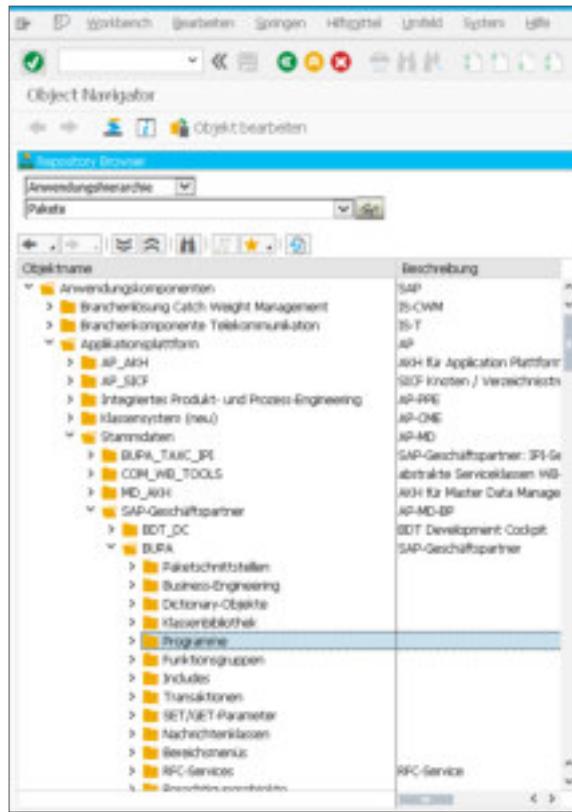
Wer Ordnung hält, hat kein Selbstvertrauen.

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

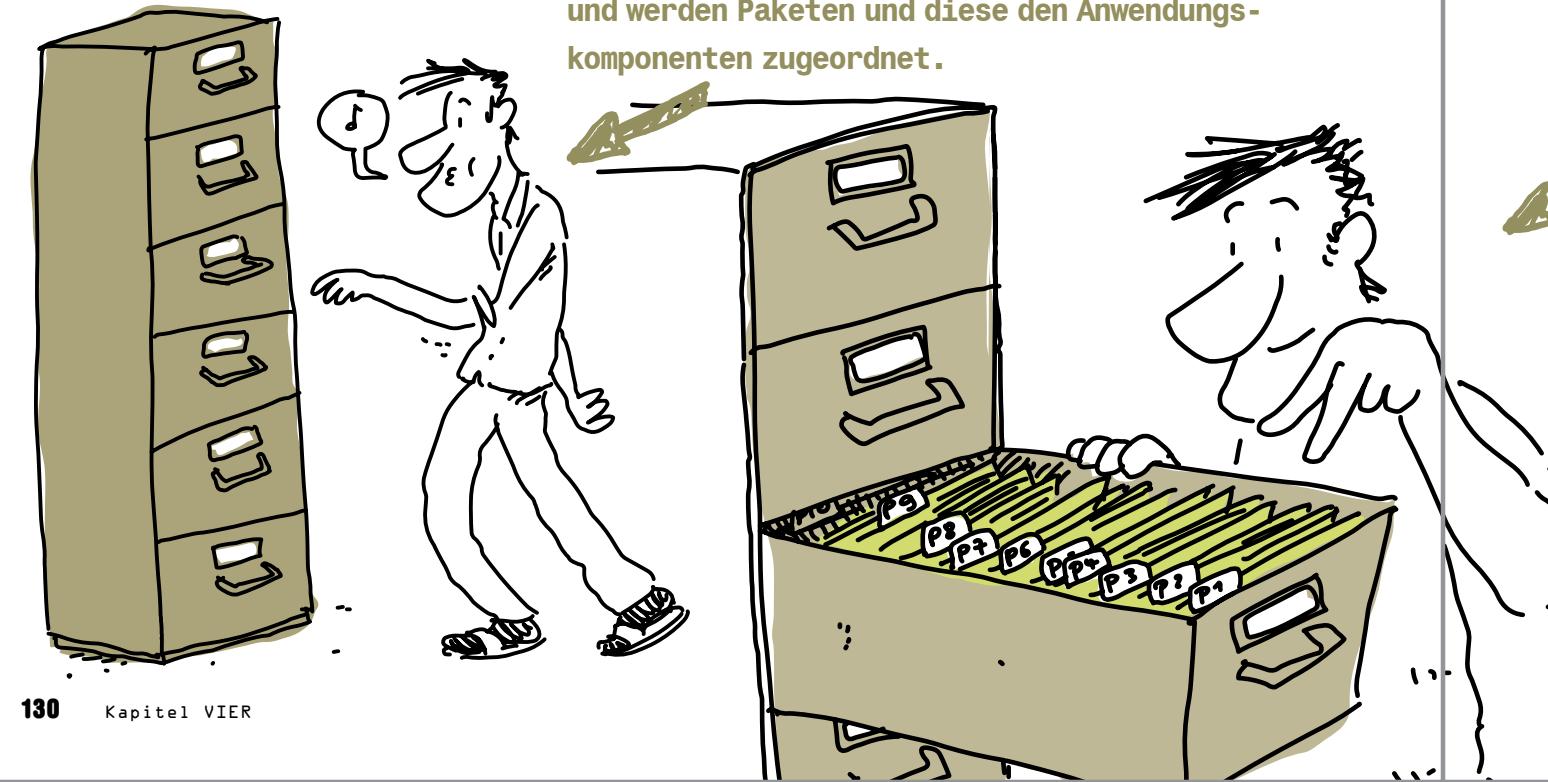
[Begriffsdefinition]

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 wiederwendbare 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 Geschäftsbereich (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 **Geschäftsbereiche**. 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.

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.



Die ANWENDUNGSKOMPONENTEN sind durch PAKETE weiter unterteilt.

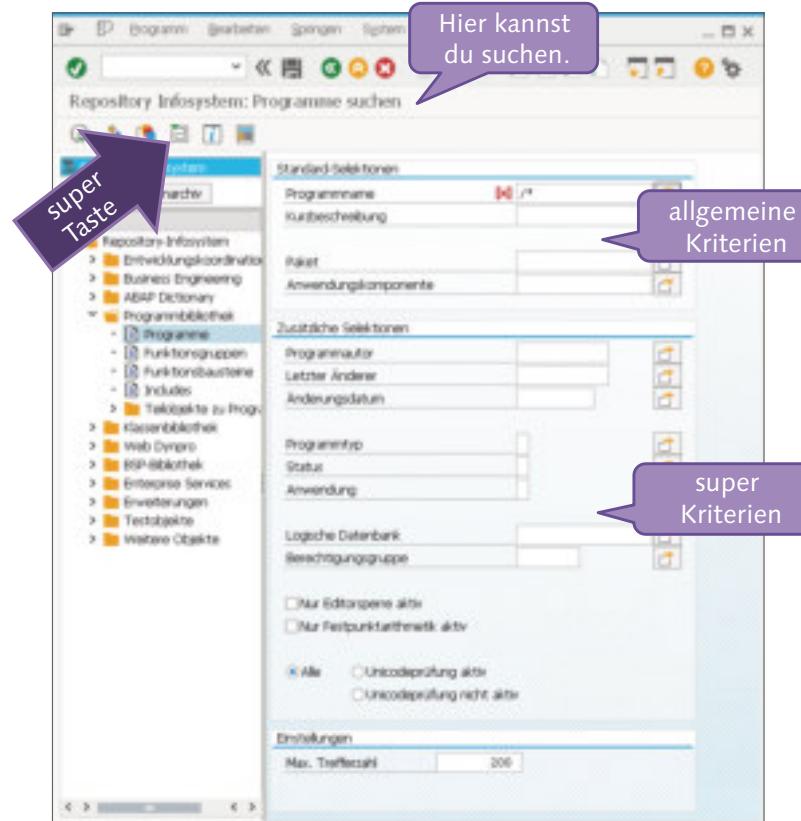
Eine Anwendungskomponente mit Paketen, die aber nicht von der Post gebracht werden.

Alles gut verpackt im Paket. Das Paket ist die letzte Kapselungsebene für Entwicklungsobjekte. Jedoch stehen dir drei Paket-Granularitäts-ebenen 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.



Wer sucht 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.



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)** ($\diamond + F9$) 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.

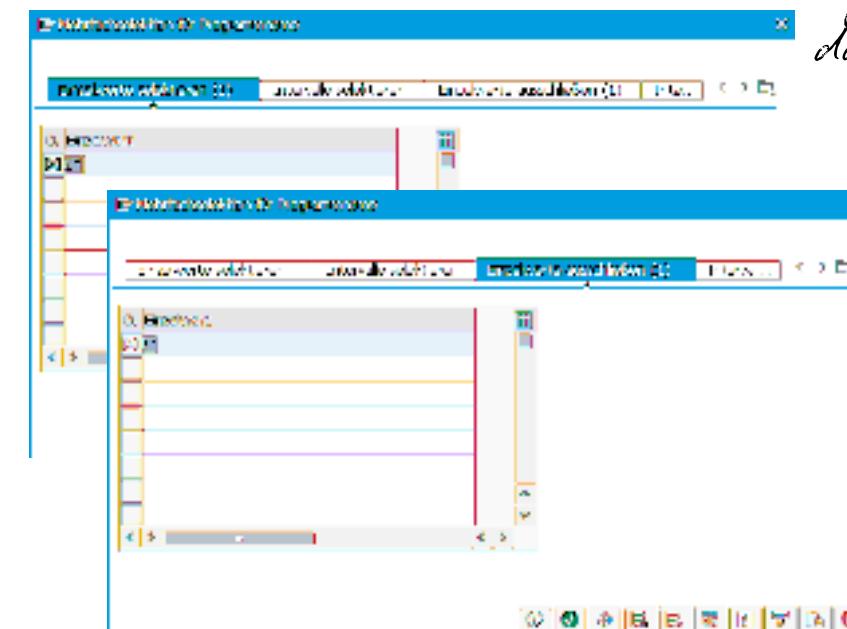
Ungewöhnlich ist, dass du **einschließende** (das soll im Ergebnis drinnen sein) und **ausschließende** (das soll sicher nicht dabei sein) **Bedingungen** pflegen kannst. Für meine Aufgabenstellung „Z“ dabei, „_“ nicht dabei“ würden wir also auf dem Karteireiter **Einzelwerte selektieren** in der ersten 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?

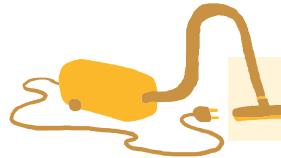
Und das ist gut so, denn das System hat die Werte in die Selektionskriterien übernommen, und mit dem vollen Kästchen wird dir angezeigt, dass da mehr als ein Wert hinterlegt ist.



[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.



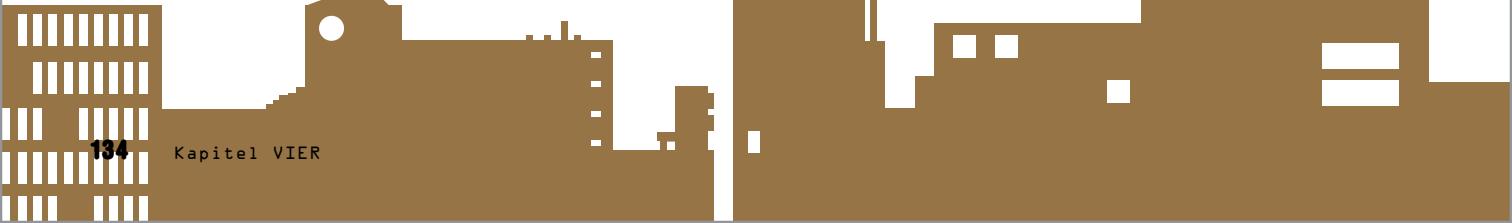
[Schwierige Aufgabe]

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 ($\Delta + F9$) gedrückt (das war noch fieser) und dann im Feld Transaktionscode *ABAP* eingegeben und im Block Zusätzliche Selektionen Klassifikation Professional User Transaction ausgewählt. Pfffff. Da kommen tatsächlich einige Ergebnisse. Spannend ist die Transaktion ABAPDOC1, 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

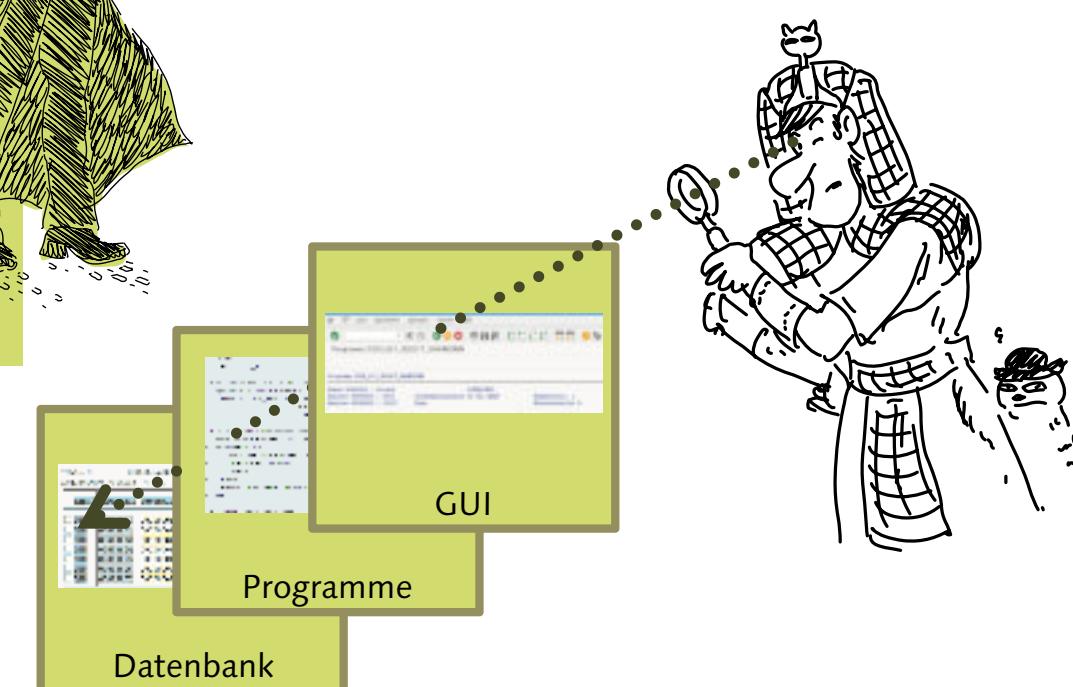
Jetzt komme ich noch zu der Frage: Wo läuft was ab?

Was ist der Unterschied zwischen dir vor dem SAP GUI und einem Anwender vor dem SAP GUI?



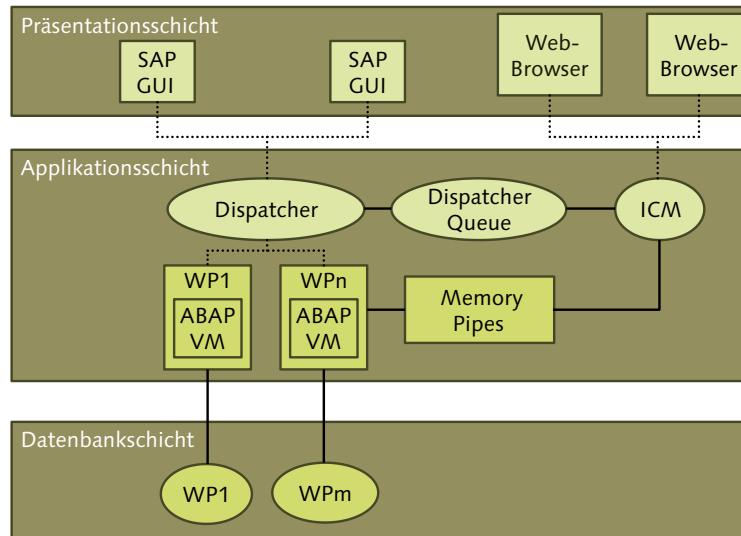
Ich sehe cooler ans!

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 schichten,
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.

Mhm, sehr
interessantes GUI.



```

* 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**.



```

* 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.

* Value display and update
gs_project->fill_gr_container(gr_container).
gr_container->set_gr_picture(gr_picture).
gr_picture->set_gr_picture(gs_project->get_gs_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.

```

DATA: gs_project TYPE CC_RS_POINTER.
SELECT SINGLE * FROM zsch03project INTO gs_project
  WHERE projekt = pa_proj.

```

Kommunikation mit der Datenbank.

Wohin mit dem Anwender – Speicherorganisation extern und intern

*Alles wieder okay.
Der Zuckerschrank
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.

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.

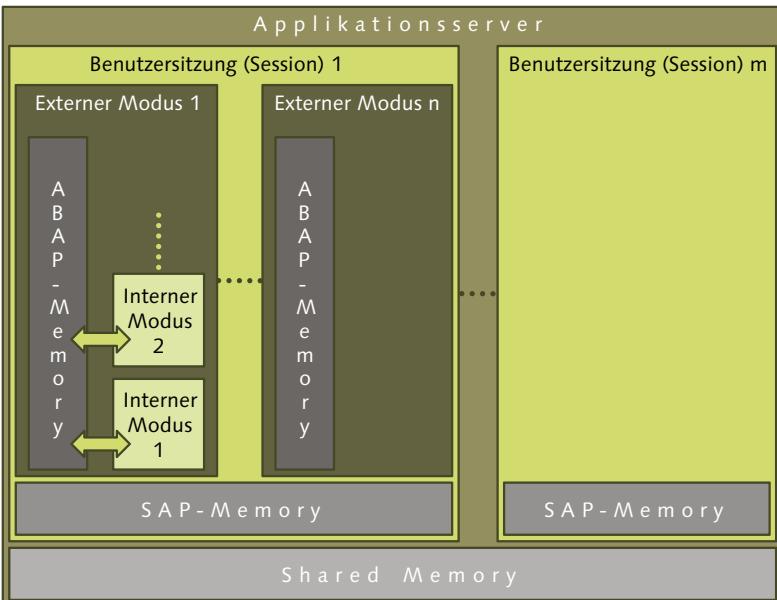
Falls Daten von der Datenbank benötigt werden, wird auf die Datenbank zugegriffen. Jedem **Arbeitsprozess** ist eine **Datenbankverbindung** fest zugeordnet. Eine feste Zuordnung von Anwendern zu Arbeitsprozessen gibt es nicht. Das hilft bei der Skalierung der Benutzeranfragen!

Eine längere Benutzersitzung kann der Reihe nach unterschiedliche Workprozesse nutzen.

Diese Info wird noch bei der Besprechung der SAP-Transaktionen wichtig.

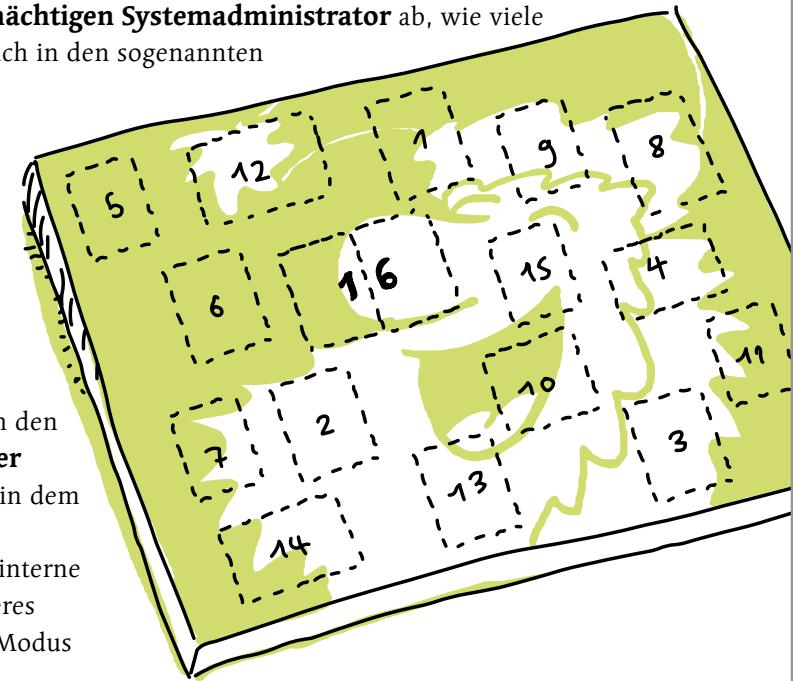
[Belohnung]

Damit du noch von Schichten träumst, könntest du dir eine ordentliche Portion Tiramisu gönnen, mit einem Espresso. Wohl bekomm's!



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 au ruft, 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.

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>**

Werkzeugkiste – Entwicklungswerzeuge

Du hast es gut!

Ich? Warum?

Das will ich dir sagen: Es reicht, dass du dir eine Transaktion merbst: 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 Universal-schlüssel zur allumfassenden Entwicklung.

Endlich selbstständig!

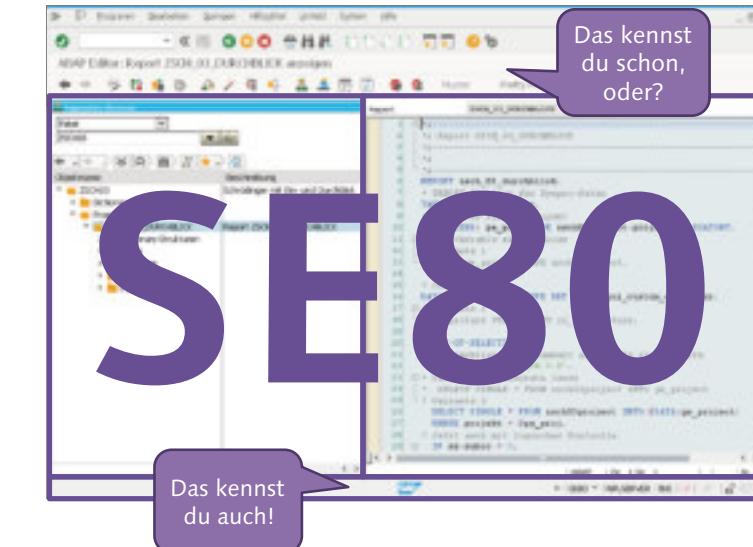


[Einfache Aufgabe]

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

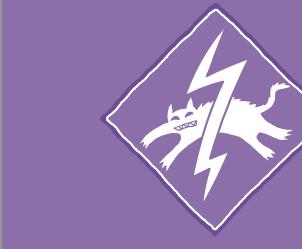


Object Navigator – Der Regisseur



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.

SE80, das Füllhorn der Entwicklung.



[Achtung]

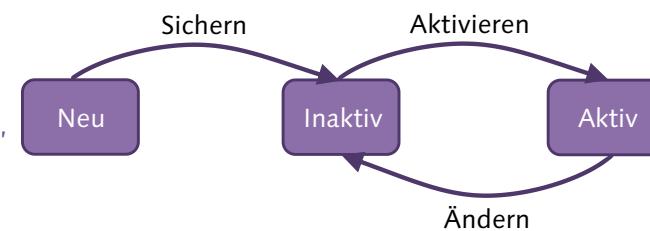
Es wird Zeit, über **Zustände** zu sprechen. Ein Entwicklungsobjekt im SAP-System hat prinzipiell drei Zustände: **neu**, **in-aktiv** und **aktiv**. Das siehst du immer im Werkzeug, rechts neben dem technischen Namen des Objektes. **Neu** ist das Objekt, wenn es angelegt, aber noch nicht gespeichert 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.

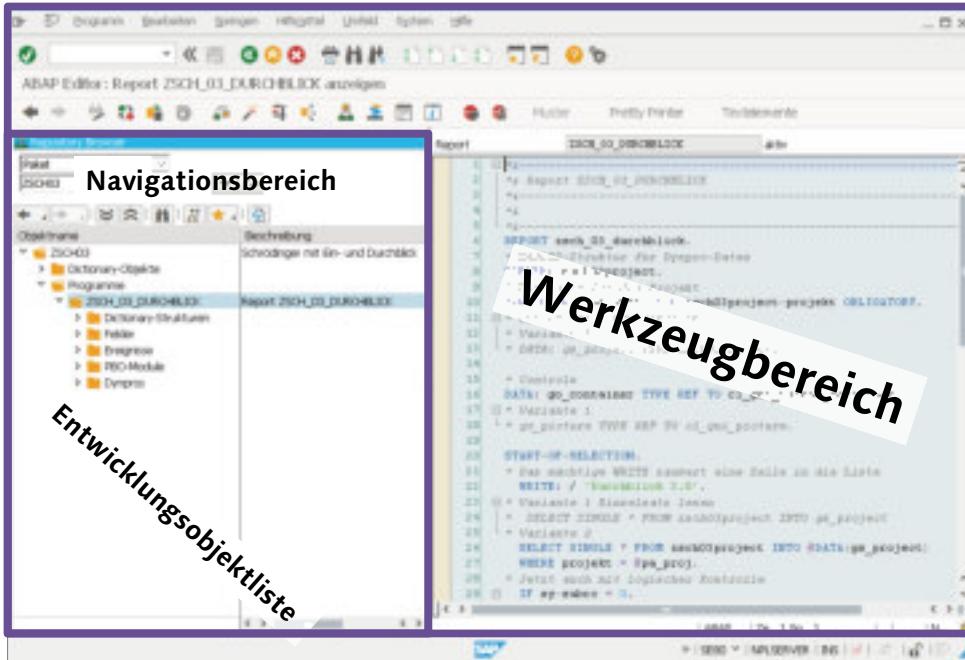


Der SE80-Wunderwuzzi.

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.

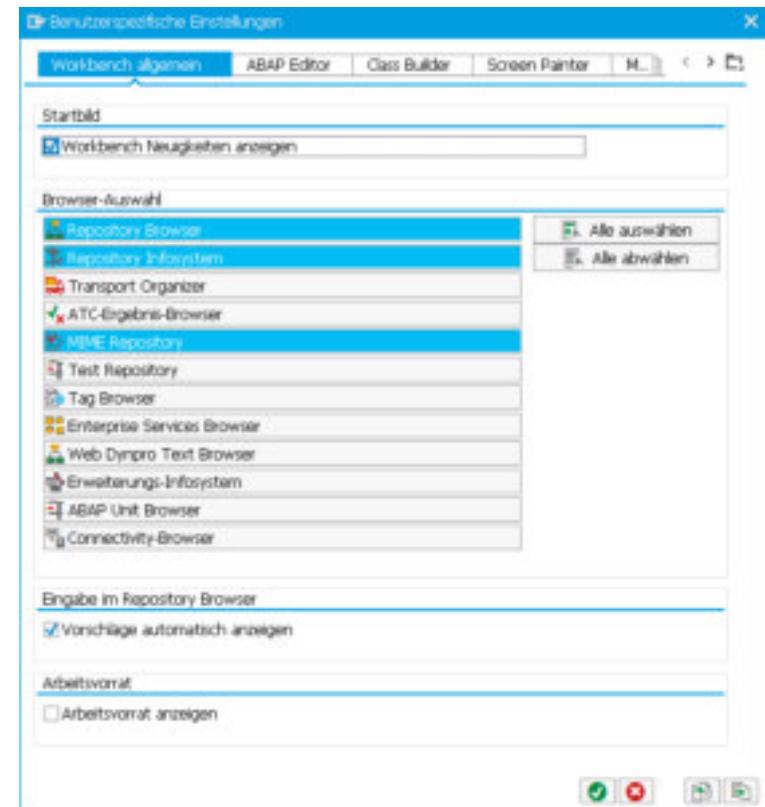
Auf dem Karteireiter **Workbench allgemein** kannst du deine gewünschten Browser durch Setzen der Checkbox für die Anzeige auswählen. Da ist ja der **Repository Browser**!

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



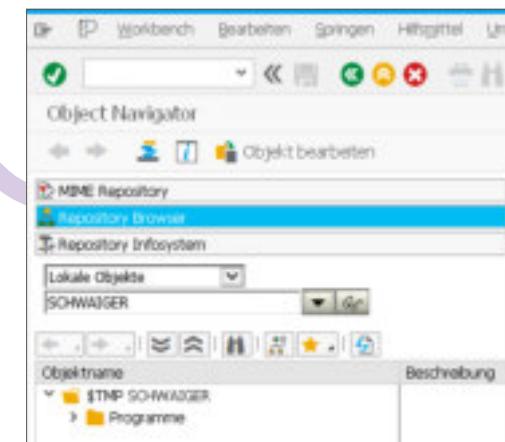
[Einfache Aufgabe]

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 Benutzernamen 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.

Kategorie

Entwicklungsobjekt

Teilobjekte

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
Andere Objekt (↑ + FS).

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)



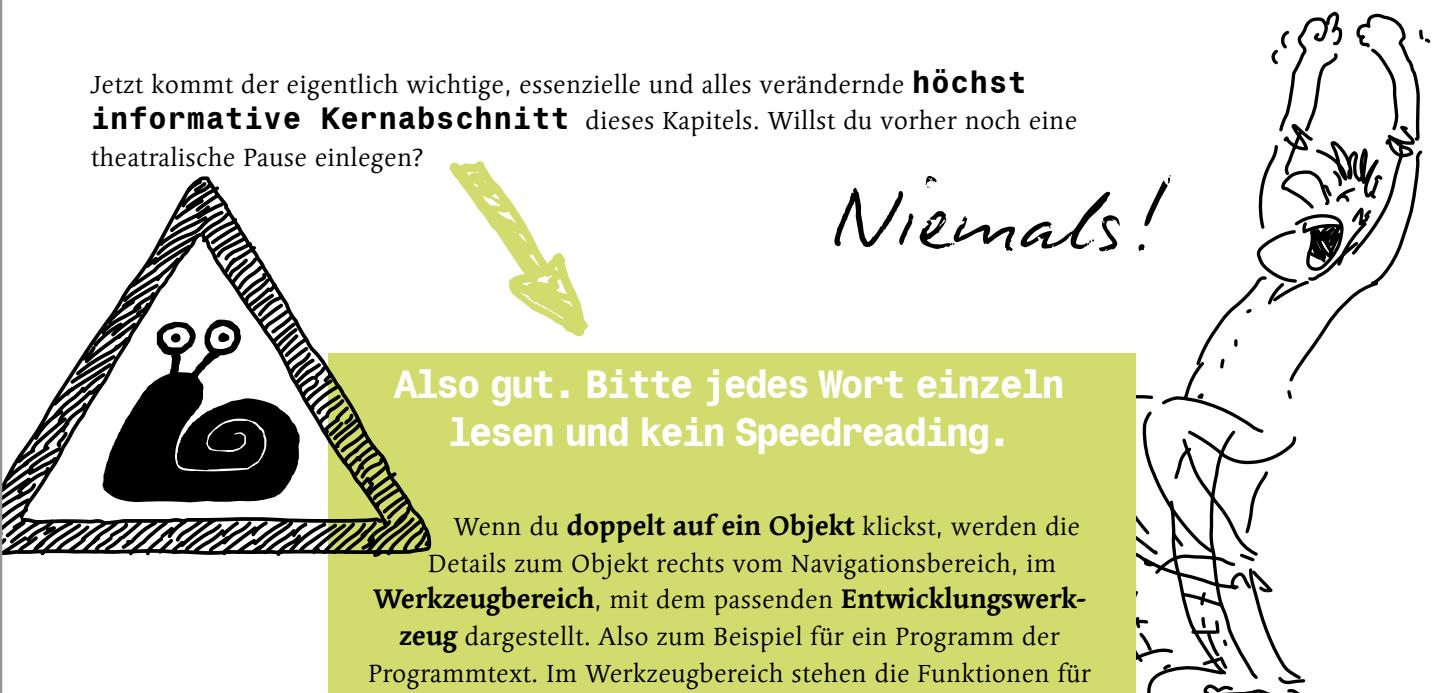
[Einfache Aufgabe]

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 was, humm?
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?



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 **Entwicklungswerzeug** 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 **Vorwärtssnavigation** 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?

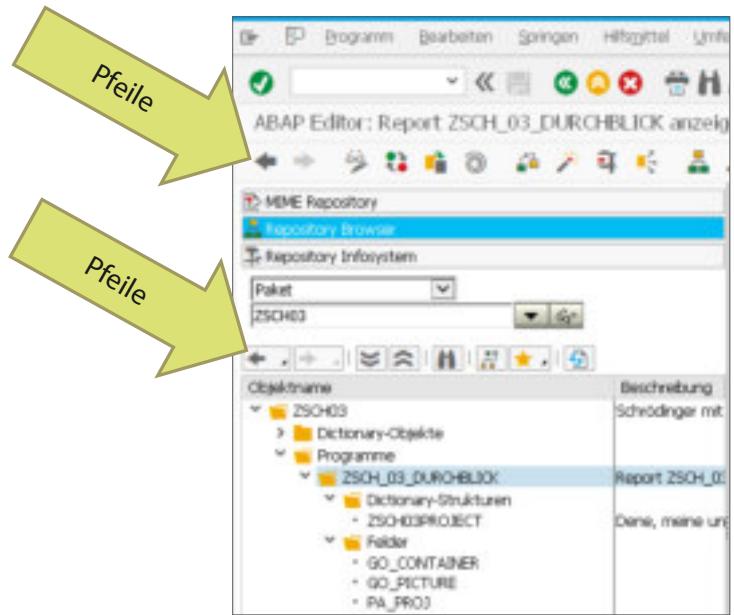
Ach, DDIC hab ich ja noch gar nicht erklärt. Das Data Dictionary – dafür steht die Abkürzung – ist eine alternative Bezeichnung fürs ABAP Dictionary. DDIC war die bevorzugte Bezeichnung, bevor das Java Dictionary ins Spiel kam.



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!



[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 bearbeitest 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).



Anzeigen ↔ Ändern

Aktivieren

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.

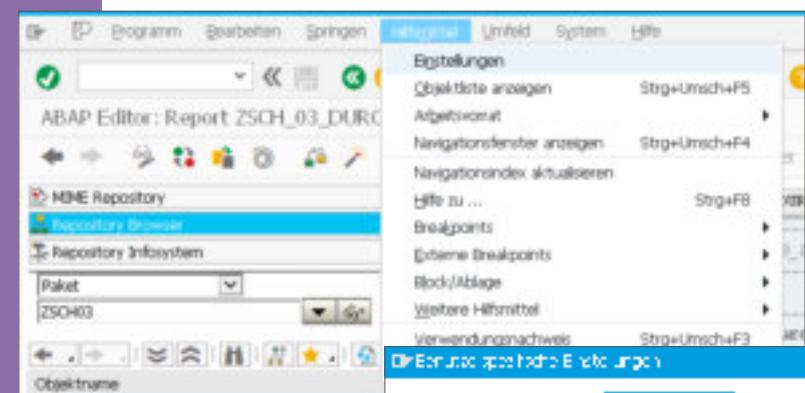
Und zum Schluss noch ein Anwendungsszenario: Die **Anlage eines neuen Objektes** erfolgt entweder über die **Vorwärtsnavigation** aus dem Navigationsbereich heraus (Namen eingeben und mit der **Enter**-Taste bestätigen, den Rest erledigt das System) oder über die Drucktaste **Anderes Objekt** (♂ + F5)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



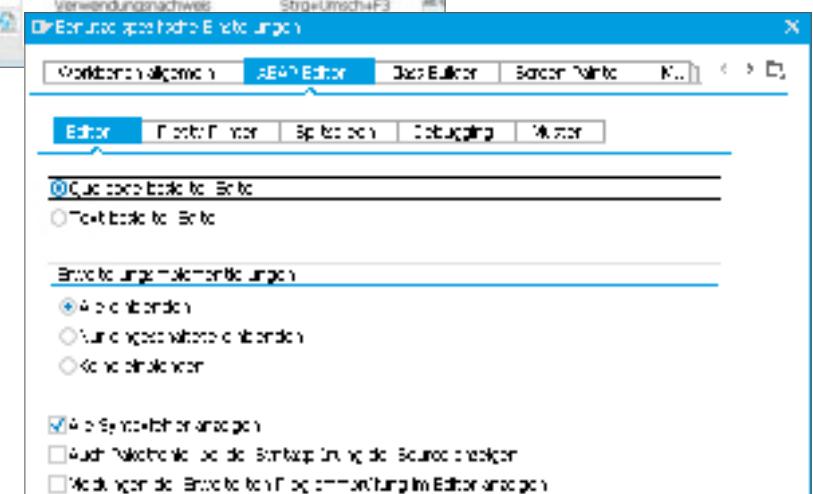
ABAP Editor – Die Schreibmaschine

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.

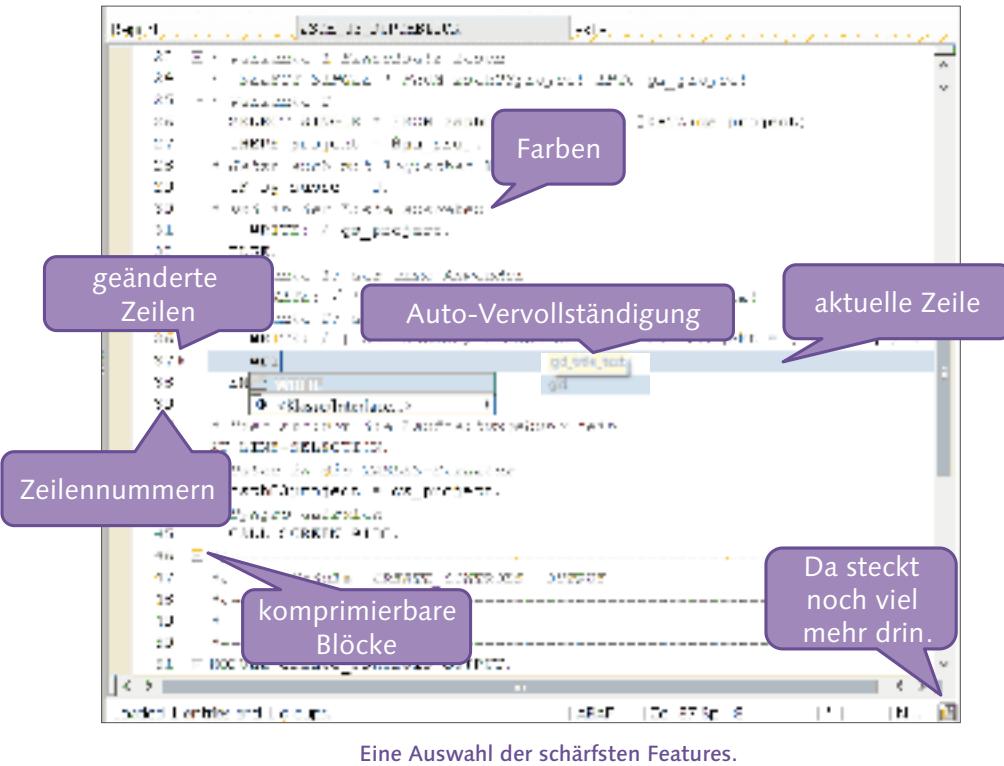
Der Editor kommt je nach SAP-Release in unterschiedlichen/mehreren Versionen daher. Du kannst ganz einfach feststellen, woran du bist:
Hilfsmittel • Einstellungen • ABAP Editor.



Welchen Editor hätten Sie denn gerne?

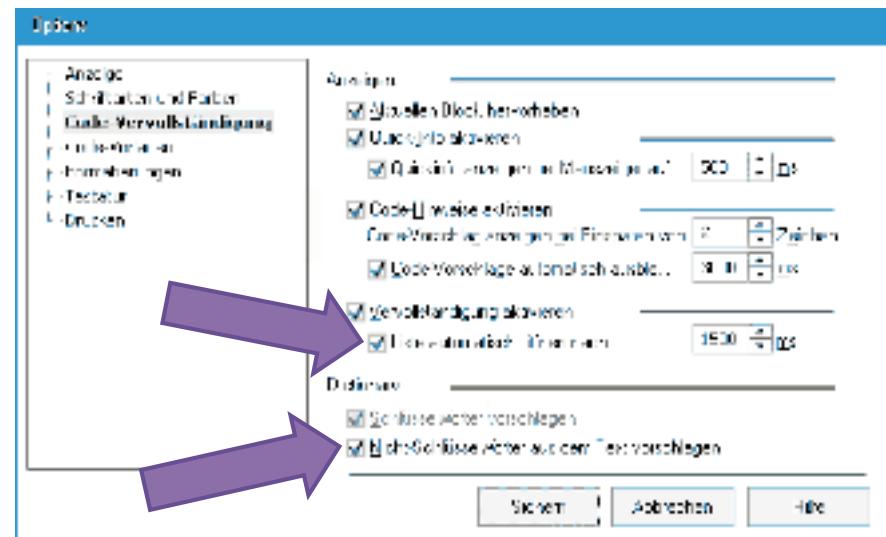


Natürlich verwenden wir den **Quellcode-basierten Editor**, der kann naturgemäß am meisten, übrigens ab SAP NetWeaver 7.0. Außerdem ist er schön bunt, hebt zum Beispiel Schlüsselwörter farbig hervor, kontrolliert automatisch die Syntax usw.



Eine Auswahl der schärfsten Features.

Du kannst Farben definieren, Schriftarten einstellen, Blöcke komprimieren/expandieren, die Zeilenummern 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**.

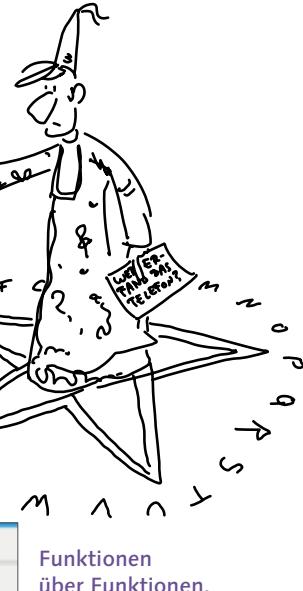
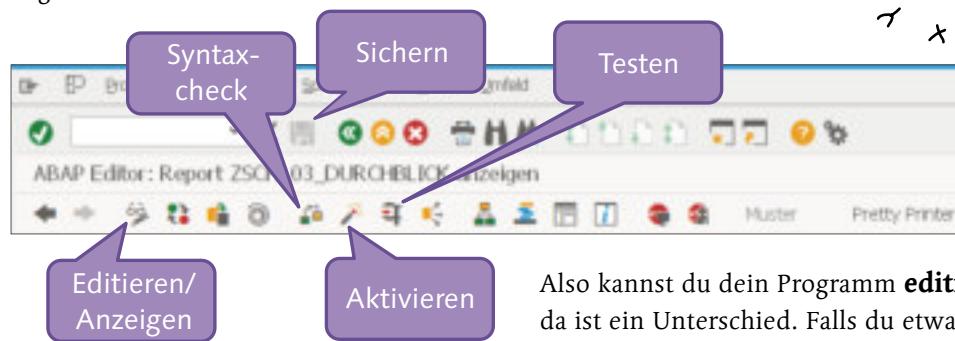


Vervollständige dich.

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** (Entwandler) 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]

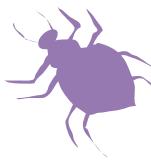
Der Start mit /h ist besonders dann geeignet, wenn du ab einem gewissen Zeitpunkt während der Programm-ausführung in das Debugging wechseln möchtest.

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.



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.

REPORT zschr_03_inzweistundenistes.

* Party on

START-OF-SELECTION. *1

* Jetzt

WRITE: | Jetzt ist es: { sy-zeit TIME = USER } | . *2

* In zwei Stunden ist es ...

* Zeitarithmetik

DATA(gd_inzweistunden) = CONV t(sy-zeit + 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**.

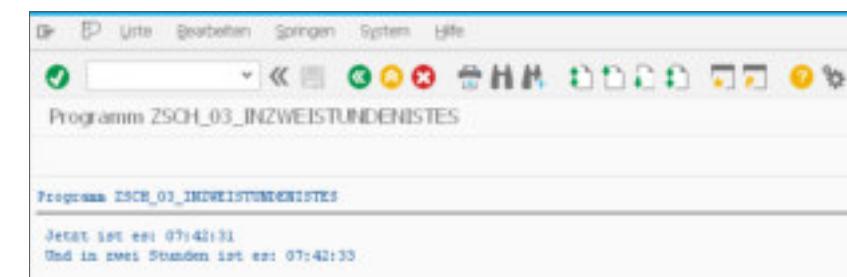
*5 <Füge bitte hier die Erläuterung ein>

*2 Mit **WRITE** geben wir eine Zeile aus, und zwar einen Text und den Inhalt des Feldes **sy-zeit**, 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.

*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-zeit** ist ein wichtiges Systemfeld, das dir die Zeit am Applicationsserver liefert. Es hat das Format Stunden:Minuten:Sekunden, jeweils zweistellig.

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?

```

REPORT zsch_03_inzweistundenstest.
* party on
* START-OF-SELECTION.
* Jetzt
WRITE: | Jetzt ist es: | sy-zeit TIME = USER ||.
* In zwei Stunden ist es ...
* Zeitzauber
DATA(gd_inzweistunden) = CONV t( sy-zeit + 2 ||).
* und in zwei Stunden
WRITE: | Und in zwei Stunden ist es: | gd_inzweistunden TIME = USER ||.

```

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 anzeigen: Hier ist mal ein Zwischenhalt.

Jetzt wird es ernst. Teste das Programm!

```

REPORT zsch_03_inzweistundenstest.
* party on
* START-OF-SELECTION.
* Jetzt
WRITE: | Jetzt ist es: | sy-zeit TIME = USER ||.
* In zwei Stunden ist es ...
* Zeitzauber
DATA(gd_inzweistunden) = CONV t( sy-zeit + 2 ||).
* und in zwei Stunden
WRITE: | Und in zwei Stunden ist es: | gd_inzweistunden TIME = USER ||.

```

Einzelnschritt hoppeln

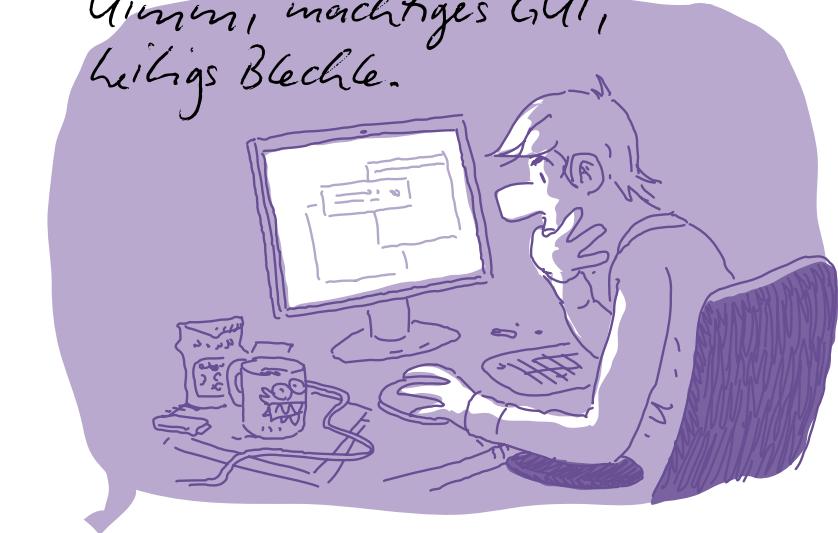
Hier steht gerade die Abarbeitung.

Hier kannst du dir den Inhalt einer Variablen anzeigen lassen.

Das ist dein Breakpoint.

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

Pow, was ist denn das?
Uuuuu, mächtiges GUI,
heilig's Bleche.



Mit dem **gelben Pfeil** in der braunen Spalte wird dir angezeigt, wo die Abarbeitung des Programms gerade steht. Wenn du die Taste **Einzelnschritt** (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-zeit**.

```

REPORT zsch_03_inzweistundenstest.
* party on
* START-OF-SELECTION.
* Jetzt
WRITE: | Jetzt ist es: | sy-zeit TIME = USER ||.
* In zwei Stunden ist es ...
* Zeitzauber
DATA(gd_inzweistunden) = CONV t( sy-zeit + 2 ||).
* und in zwei Stunden
WRITE: | Und in zwei Stunden ist es: | gd_inzweistunden TIME = USER ||.

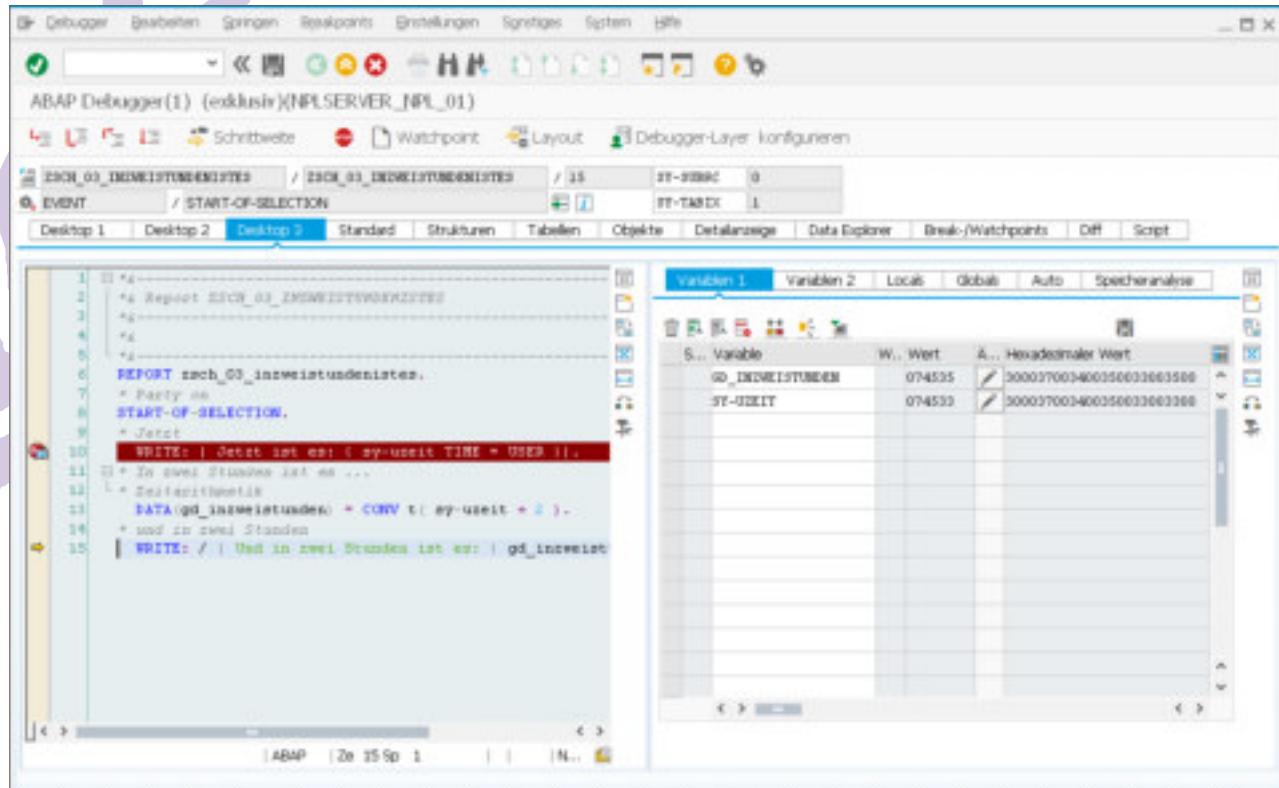
```

Variablen 1	Variablen 2	Locals	Global	Auto	Spezialeinst.
S... Variable	W... Wert	A... Hexadezimaler Wert			
sy-zeit	000000	360030003000300030003000			

Variablen 1	Variablen 2	Locals	Global	Auto	Spezialeinst.
S... Variable	W... Wert	A... Hexadezimaler Wert			
gd_inzweistunden	000000	300030003000300030003000			
sy-zeit	074533	300037003400350033003300			

Mit einem Doppelklick auf eine Variable kannst du dir den Inhalt anzeigen. Schau mal auf der rechten Seite. Da steht für die Variable **sy-zeit** 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 Programmablaufnung ansehen. Ich bin gleich zweimal weitgesprungen. Dadurch ist die hochwissenschaftliche Berechnung ausgeführt worden.



The screenshot shows the SAP ABAP Debugger interface. The code editor displays a program named ZSCH_03_ZINNWEISTUNDENLISTE. The variable table shows two variables: GD_INNWEISTUNDEN with a value of 074535 and ST-UZEIT with a value of 074533. The status bar at the bottom indicates ABAP 12 15 Sp 1.

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]

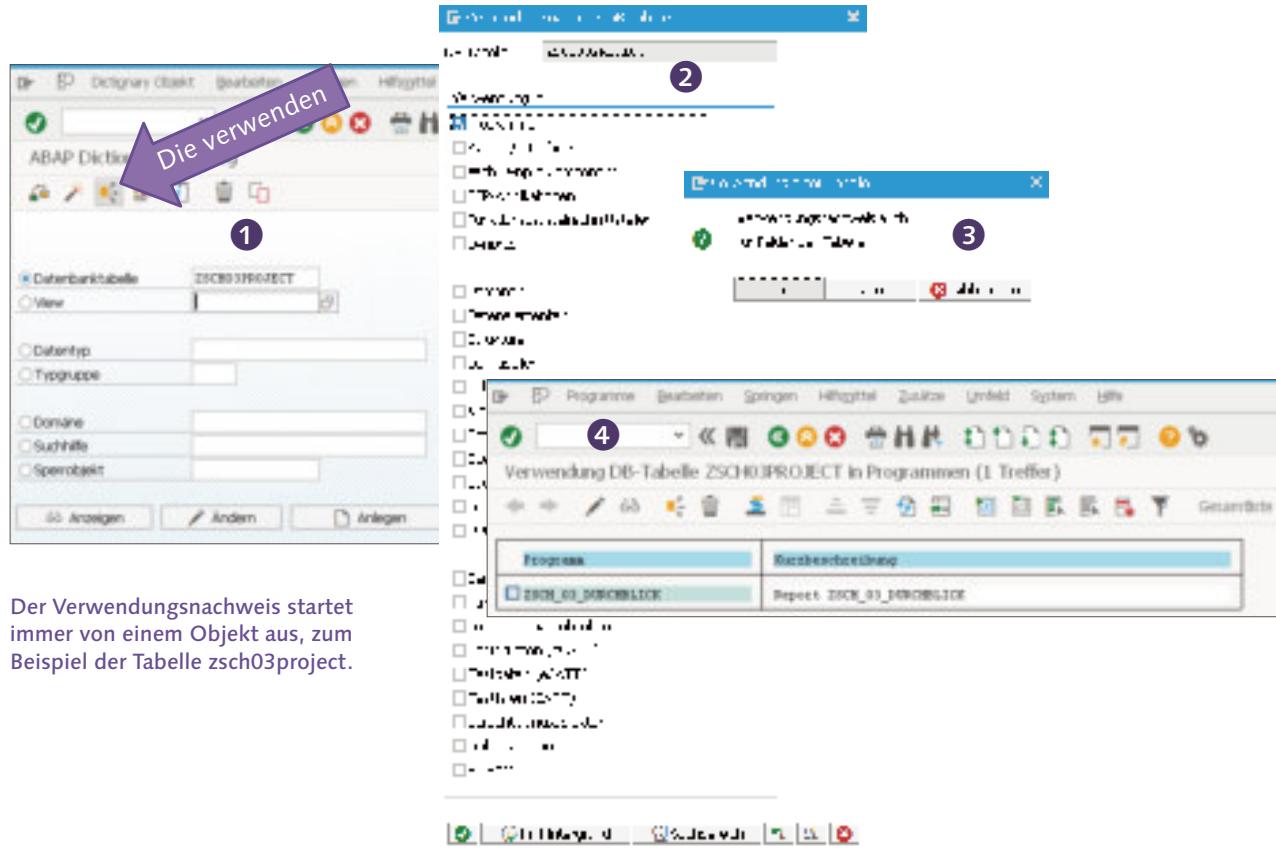
Pause? Turnübung? Liegestütz, oder doch unter den Tisch reihängen und hochziehen?

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 **zscho3project** verwendet?



- 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?

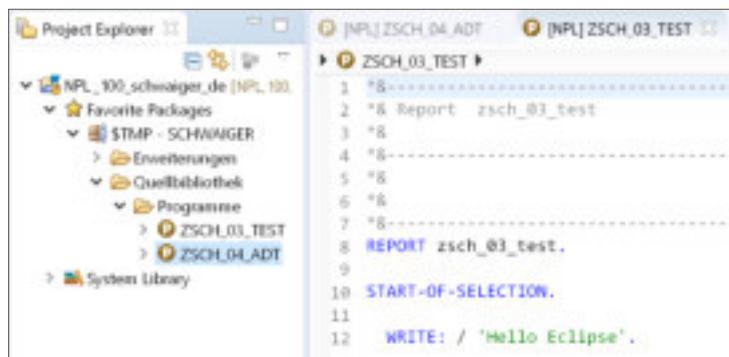
Ende der Vorstellungsrunde. Die anderen üblichen Verdächtigen wirst du in späteren Kapiteln noch kennenlernen: den **Class Builder** für die Objektorientierung, den **Function Builder** für Funktionsbausteine, das **ABAP Dictionary** für globale Typen, den **Screen-Painter** für Fensterprogrammierung.

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.



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.



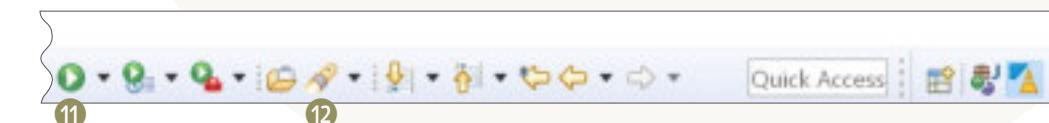
Dann kann ich also mehrere Werkzeuge gleichzeitig offen haben? Sehr praktisch!

Super praktisch! Links in der Objektliste ist das Programm **ZSCH_04_ADAPTER** 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 Teillokten 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.

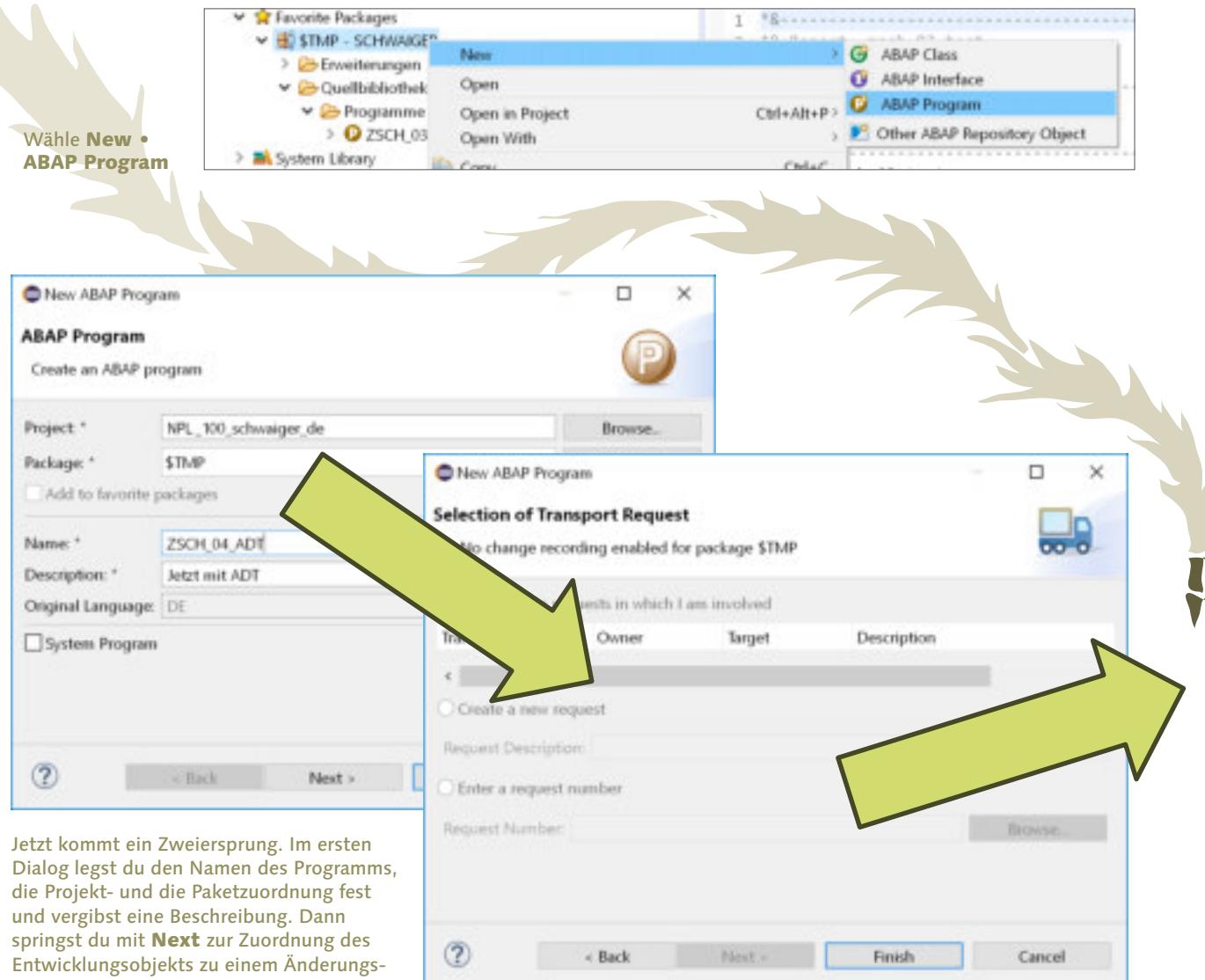


Festhalten, jetzt kommen gaaaanz viele Funktionen. Die findest du alle in der Toolbar. Ich erkläre dir die wichtigsten.

- 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**.



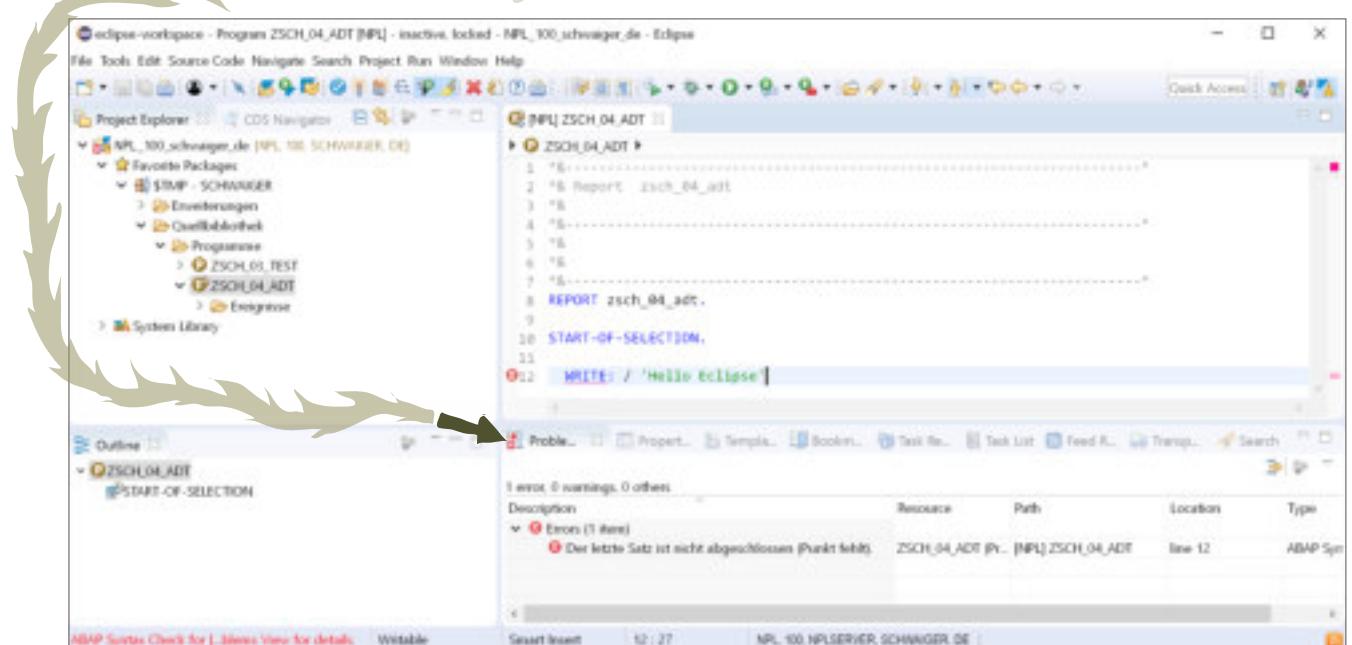
Jetzt kommt ein Zweiersprung. Im ersten Dialog legst du den Namen des Programms, die Projekt- und die Paketzuordnung fest und vergibst eine Beschreibung. Dann springst du mit **Next >** zur Zuordnung des Entwicklungsobjekts zu einem Änderungsauftrag (sofern du nicht gerade ein lokales Objekt anlegst). 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:

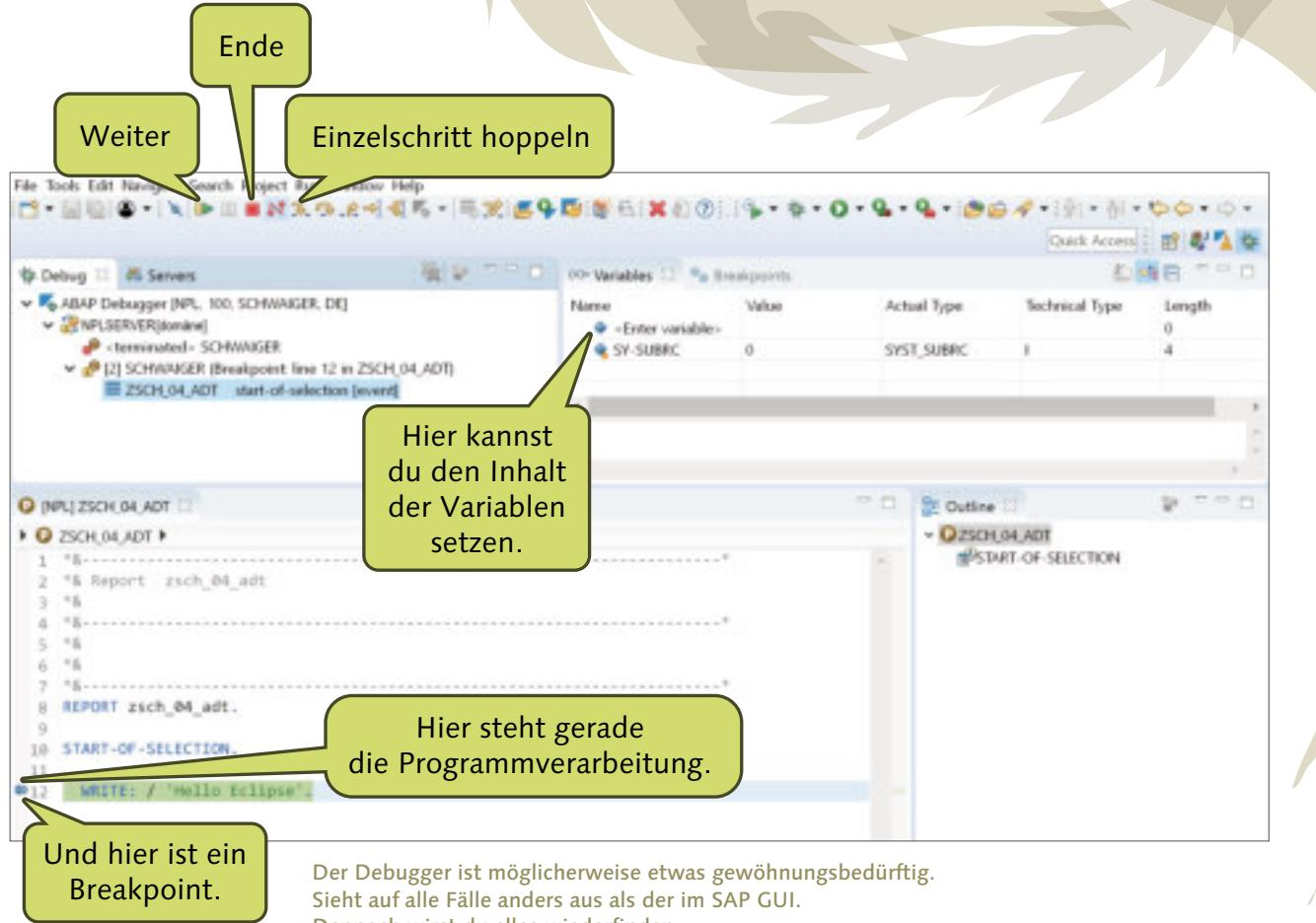
- ☞ Wenn du eine **Auto-Vervollständigung** wünschst, drücke Strg + die Leertaste. Prinz Editor liefert dir stets passende Vorschläge.
- ☞ Mit ⌘ + F1 kannst du deinen Code **pretty printen**, also formatieren.
- ☞ Über die bereits besprochenen Symbole in der Toolbar kannst du dein Programm **prüfen, 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!

INHALTSVERZEICHNIS

Vorwort	24
---------------	----

Kapitel 1: Eine Nudelmaschine für zu Hause

Die SAP, das SAP-System und ABAP

Seite 27

Den Schrödinger hat es mal wieder erwischt: Ein österreichischer Nudelhersteller hat seine alte Firma, Spaghetti Infinito, übernommen. Dabei war das doch bis jetzt so gemütlich in der Spaghetti-Informatik. Man konnte programmieren, wie und womit man wollte – Hauptsache, es lief. Wenn er zukünftig keinen Spaghetticode mehr programmieren darf, kann er sich gleich etwas Neues suchen: Zum Beispiel bei der ABAP-Fabrik, von der er schon so viel gehört hat. Klingt lustig, ABAP. Und wenn die auch Code produzieren ..., vielleicht kann er dann ja doch wieder ein bisschen Spaghetticode programmieren! An seinem ersten Tag lernt er den Schwaiger Roland kennen, seinen ABAP-Ausbilder. Der seine Vorliebe für Spaghetticode sofort entdeckt und sich zunutze macht. Aber anders, als Schrödinger denkt.

Servus Schrödinger!!!	28	Kompilieren/Interpretieren	41
Schrödingers SAP-System	31	ABAP allgemein	42
Die SAP – Das Unternehmen	33	ABAP-Syntax	43
SAP-System	34	Beispiel ABAP	45
Die Erfolgsgeschichte: Vom ABAP zum ABAP	37	Schrödingers ABAP	46

Kapitel 2: Rein in die Küche – raus aus der Küche

Starten mit dem SAP-System

Seite 47

Ein spannender Tag für den Schrödinger: Das erste Mal! Der Schwaiger Roland meint, das tut gar nicht so weh, wie alle sagen, der hat leicht reden. Und fängt gleich mit der Anmeldung am SAP-System an. Äh, wo eigentlich? Ach ja, SAP Logon. Geschafft, und jetzt das Anmeldebild. Wie? Ach ja, SAP GUI. Aber was soll das mit diesen Mandanten? Wir sind doch keine Anwälte, wir sind Entwickler!

Anmelden zum Tauchkurs: Eintauchen in die SAP-Welt	48	Transaktion starten	53
Alles ist Transaktion – Transaktionen als Shortcuts	50	Willkommen Schrödinger! – SAP-Willkommenstext	54
Melde dich an! – SAP-Anmeldung	51	Die Auslage des SAP-Systems – SAP GUI-Aufbau	57

Alles auf einen Blick – SAP Easy Access	59	Hüben und drüben, nichts als Üben – Rolands Modus Operandi – Zusätzliche Fenster öffnen	61	Folterkammer oder Übung macht den Meister	64
Das ist doch kein Thema	63	Hoppi-Galoppi ... SAP Fiori	66		

Kapitel 3: Ciao a tutti! (Hallo Welt!)

Der 20-Minuten-Einstieg in die ABAP-Programmierung

Seite 69

Endlich, das erste ABAP-Programm! Schrödinger war sich bislang gar nicht sicher, ob der Schwaiger Roland überhaupt programmieren kann. Und dann legt er auf einmal richtig los: Pakete anlegen, Datenbankzugriffe, Ausnahmebehandlung, Schlüsselwortdokumentation. Dieses ABAP fängt an, Spaß zu machen, weil es so viel zu entdecken gibt.

Einsteigen und anschnallen!	70	Das ABAP-Einmaleins – Grundlegende Syntax	96
Object Navigator – Die integrierte Entwicklungsumgebung	72	Der Durchblicker – Programm entwickeln	98
Entwicklung organisieren – Systemlandschaft, Änderungsauftrag	73	Her mit den Daten – DB-Zugriff	100
Ihr neuer Auftrag lautet: Auftragsanlage	75	Jetzt lese ich	101
Organisationskapsel Paket	77	Ohne Struktur läuft nichts – Grundlegende Programmstruktur	102
Vom Entwickler zum Pakettier – Paket anlegen	79	Alles muss es auch nicht sein – Einfaches Selektionsbild	104
Einen Schritt zurück und bald zwei nach vorne	82	Layouthen und Daten transportieren –	
Die Infos zum Aufbau einer Datenbanktabelle	83	Ein einfaches Dynpro	108
Das zweischichtige Domänenkonzept	84	Dekorieren der Auslage – Dynpro-Layout	111
Technisches Fundament eines Typs – Domäne anlegen	86	Ablauflogik ohne Ablaufdatum – Ablauflogik programmieren	117
Lege die Bedeutung an – Datenelement anlegen	89	Ein Shortcut für den User – Transaktionscode anlegen/ausführen	121
Kühlschrankdesign – DB-Tabelle anlegen	91	Aufbruch in eine neue Welt:	
Daten verarbeiten – Programm anlegen	95	ABAP Development Tools	122

Kapitel 4: AAA – Alles außer ABAP

ABAP-Infrastruktur und -Werkzeuge

Seite 125

Der Schrödinger hat es schon gemerkt: Zum ABAPen reicht es nicht aus, sich die Sprache ABAP selbst reinzu ziehen. 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?	126	Object Navigator – Der Regisseur	141
Die Ablage der Entwicklungsobjekte –		Repository Browser ausnutzen	143
Repository	127	Repository Browser browsen	144
Wer suchet der findet – Suchen mit dem		Synchron oder aus dem Tritt –	
Repository Infosystem	132	Objekt-Browser-Synchronisation	147
Suchen und Finden – Repository Infosystem	134	ABAP Editor – Die Schreibmaschine	149
Geschichtet, aber nicht gefaltet –		Debugger – Der Kammerjäger	152
ABAP-Infrastruktur	135	Debugger entschleunigt	153
Wohin mit dem Anwender –		Verwendungsnachweis –	
Speicherorganisation extern und intern	138	Der Rückwärtssucher	157
Werkzeugkiste – Entwicklungswerzeuge	140	Die ADT können das auch	160

Kapitel 5: Schräge Typen* *also jetzt nicht der Schwaiger Roland

Datentypen und -objekte

Seite 165

Definieren und deklarieren: das tägliche Geschäft des ABAP-Programmierers. Das ist ja wie mit den unterschiedlichen Nudeltypen. Die eine Sorte ist lang und dünn, andere sehen aus wie Röhren oder wie Schmetterlinge. Jedoch, und das weiß der Schrödinger natürlich, wird für ein bestimmtes Gericht oder eine besondere Pastasoße ein bestimmter Nudeltyp benötigt. Und so ist es bei der Programmierung eben auch: Für bestimmte Aufgaben werden bestimmte Typen benötigt.

Vorspann	166	Dann elementar global – Globale	
Von 0 auf 100 in wenigen Seiten – Technik der		elementare Typen	180
Datenspeicherung im Hauptspeicher	166	Elementar global con domäne – Domäne	183
Variables Standgas – Datenobjekte	168	Strukturell lokal – Lokale strukturierte Typen	186
Variablen sind zum Angreifen	169	Struktur global – Globale strukturierte Typen	188
Variablen Deklaration inline	170	Tabellarisch – typisch/intern –	
Ein typischer Blickwinkel – Datentypen	171	Lokaler Tabellentyp	192
Anders betrachtet – Datentypen	173	Her mit der internen Tabelle	193
Die Vermessung	174	Wie viel Spalten hätten Sie denn gerne –	
Ansprache – Datenobjekte verwenden	175	Zeilentyp	194
Dynamik pur oder Statik nur? – Statisch und		Normal, sortiert oder doch gehasht –	
dynamisch typisierte Variablen	176	Tabellenart	195
Mein lokaler Typ	177	Open the door please – Schlüssel	197
Zuerst mal elementar lokal – Lokale		Heute mal kopflos – Kopfzeile	198
elementare Typen	178	Tabellen für alle – Globaler Tabellentyp	199

Operation: Table – Tabellenoperationen	201	Ändern und Löschen zum Abrunden –	
Bibliothek einräumen	203	Noch mehr Tabellenoperationen	206
Einzelne geht es auch – Einzelverarbeitung von		Eine kleine Gutenachtgeschichte (über CDS)	207
Tabelleneinträgen	205		

Kapitel 6: Si parla ABAP? Echtes Küchenlatein

ABAP-Syntax I

Seite 213

Jetzt geht's ans Eingemachte, Schrödinger, oder sollten wir besser sagen ans Eingeckochte? Hier lernst du mal das Basisvokabular, so wie damals, als nur Spaghetti gemacht wurden und nicht auch dieses andere Zeugs.

Von Kontrollen und Schleifen	214	Notandor – Logische Ausdrücke	228
Zuweisungen, Operationen, Funktionen	214	Verzweige, falls ... – IF ... THEN ... ELSE	230
Bitte nicht abweisen – Zuweisungen	217	Cassis – CASE ... WHEN ... ENDCASE	232
Was du darfst und was du sollst –		Schleifen	233
Konvertierungsregeln	219	Bedingungslose Schleifen – DO ... ENDDO	233
Ketten ohne Perlen – Zeichenketten-Funktionen	221	Krasses Projekt für Hardcore-ABAPer	234
Von Segmenten und Offsetten –		Bedingte Schleife – Fußgesteuert	236
Zeichenketten-Operationen	225	Bedingte Schleife – Kopfgesteuert	237
Ohne Rechnung, ohne mich – Arithmetische		Jetzt aber mal systematisch – Systemfelder	238
Operationen	227	Großbaustelle Rezeptprogramm	240

Kapitel 7: Kleine Häppchen sind leichter wiederverdaut („Schluss mit Spaghetti!“)

ABAP-Syntax II

Seite 243

Schwaiger wirkt wild und entschlossen. Er hat endgültig genug vom Spaghetti-Coding, das Schrödinger fabriziert. „Heute mache ich aus dir einen Wiederverwender“, hallt die letzte Schwaiger-Drohung durch die Hallen, und Schrödinger versucht, sich mit Wehmut von seiner letzten Spaghetti zu trennen ...

Motivation durch Demotivieren	244	Ereignisse: Rekapitulation mit Selbstreflexion	247
Den richtigen Zeitpunkt finden –		Zur richtigen Zeit an irgendeinem Fleck?	249
Ereignisblöcke	245	Meine schönsten Ereignisse –	
Ereignisreiche Programme –		Rahmenprogramm	250
Ereignisschlüsselwörter	246	Dynpros mit Modul – PBO, PAI, HOV	252

Module, ganz kurz – Dynpro-Programmierung	253	Sichtbar oder nicht – Lokale Überdeckung	262
Aber nun mit Schnittstelle – Unterprogramme	254	Aufruf bitte – Unterprogramm-Aufruf	263
Definiere zuerst! Unterprogramm-Definition	255	Totalumbau	264
Übergeben und Übernehmen – Schnittstellenparameter für das Unterprogramm	256	Globale Wiederverwendung – Funktionsbausteine	268
Lesen von DB mit Kapsel – DB-Zugriff in Unterprogramm	258	Funktionsgruppe	270
Ein Typ muss es sein – Schnittstelle typisieren	259	Leg dir eine Funktionsgruppe an	273
Bitte mit Typ – Typisieren	260	Es ist so weit, global bereit – Funktionsbaustein	274
Warum in die Ferne schweifen? – Lokale Variablen	261	Her mit dem Rezept	276
		Haaalloooooo – Funktionsbaustein-Aufruf	279
		Dein Finale	281

Kapitel 8: Schrödinger zeigt Klasse

ABAP Objects

Seite 283

Schrödinger ist heute nicht gut aufgelegt, weil der Schwaiger Roland mit der Objektorientierung daherkommt. Es funktioniert doch alles bestens mit dem „prozeduralen ABAP“ (so nennt das der Schwaiger Roland). Was hat der bloß? Na okay, ein paar Mal haben sich schon Fehler eingeschlichen, die schwer zu finden waren. Mit der Objektorientierung soll ja alles besser werden mit der Ordnung. Und wo mehr Ordnung ist, ist alles leichter zu finden – sogar die Fehler. Und außerdem kann man mit ABAP Objects anscheinend tolle GUIs realisieren. Nach dem ersten Schockerlebnis ist Schrödinger also gleich überzeugt.

Motivation zur OO	284	Mit Methode – ran an den Code	316
Begriffe aus der Realität	285	Die Schnittstelle der Methode	320
Holistische Sichtweise	287	Ganz spezielle Methoden	323
Modellierung	288	Methoden mit Fülle	325
Erst denken, dann lenken: Einfache UML als Analysewerkzeug zur Visualisierung von Klassen/Objekten	292	Selbstreferenz	328
Klasse Kaffee(-vollautomat) auf dem Papier	294	Statisches Attribut	329
Ohne meine Kollaborateure bin ich nichts	296	Methodenauftrag	330
Klasse oder doch Objekt	298	Laufzeit sequenziert	332
Klasse Lokal	300	Methoden für den Wasserbehälter	334
Dein kleines Objektistan	303	Von lokal nach global	336
Kaffee für zwei	306	Going global – SE24	337
Datenablage – Attribute	308	Attribute und Methoden	338
Wasserstand und Pause	310	Kaffeebohnen im Behälter	342
Klasse Attribute, oder was?	311	Senden und Empfangen	346
Privat verkalkt/Private Attribute	314	Sender mit Sendungsbedürfnis	347
Ab jetzt mit Methode – Schrödinger frischt auf	315	Nächster Zubereitungsschritt: der Empfänger	348
		Objektorientiertes Verkuppeln	349
		Kaffee s'il vous plaît	351

Kapitel 9: Erben oder nicht, das ist hier die Frage: Pasta di Erbe

Vererbung

Seite 353

Hoffentlich ist der Schrödinger heute in der passenden Stimmung. Eine Vertiefung der Objektorientierung, puh, da wird er schwitzen. „Was, da geht noch mehr? Jetzt hab ich mich gerade erst vom ersten OO-Schock erholt!“ Schrödinger wird wohl erkennen, dass Spaghetti, Ravioli, Tortiglioni etc. alles Nudeln sind, oder?

Motivation	354	Redefinieren – Polymorphie von Methoden	371
Vererben und Erben von Klassen	357	Redefinieren als globale Herausforderung	374
Vererbung lokal	360	Statische und dynamische Typen von Klassen	
Meister der Pyramiden	362	mit ein wenig Casting	380
Globale Vererbung: Ein praktisches Beispiel im SAP-Standard	364	Festigung der Einsichten	382
Was wird wie und wo vererbt	367	Down-Cast	385
		Abstrakte Klassen	388

Kapitel 10: Keine Details bitte

Der Blick durch Interfaces

Seite 393

„Man kann auch noch einen Schritt weiter gehen und die Implementierung von Methoden von deren Schnittstellendefinition trennen“, verkündet Roland und legt eine Kommunikationspause ein. Schrödinger steht da, der Mund ist offen, und er ist sich noch nicht der Tragweite der Aussage bewusst.

Schizophrenie?	394	Von Suchmaschinen und Tilden	403
Lokal zum Einsteigen	397	Hast du gerufen?	404
Schnittig und definiert	398	Die vielen Gestalten der Methoden	406
Volles Risiko – alles oder nichts	399	Globalisierung mit positiven Auswirkungen	409
Klasse Zuordnung	400	Global klasse Implementierung	411
Klasse Definition	401	Singleton-Party	413
Klasse Implementierung	402		

Kapitel 11: Das Kapitel für Feiglinge

Ausnahmen

Seite 417

„Fehlerbehandlung ist für Feiglinge! Das ist ja so, als ob ich beim Nudelkochen die Nudeln koste, ob sie al dente sind oder nicht!“ Schrödinger hält nichts von Ausnahmebehandlungen. Bis zum ersten Zwischenfall: als seine Freundin sich über klebrige und zerkochte Nudeln beschwerte.

Ausnahmsweise behandelt	418	Ausnahmen melden (mit Nummern)	430
Eine kleine Geschichte der Ausnahmen	420	... und behandeln (nach Nummern)	431
Noch mal langsam: Mit SY-SUBRC.		Ausnahmslos Objekte	433
Ausnahmewert wird gesetzt	424	Definieren geht vor	435
... und von dir behandelt	425	Ausnahmen melden (mit Ausnahmeklassen)	437
It's RAISING man, hallelujah.	426	... und Ausnahmen behandeln, aber richtig!	
Ausnahmen melden	426	Mit Ausnahmeklassen.	440
... und behandeln	427	Das ausnahmengespickte Projekt	441
The Catcher in the System	430		

Kapitel 12: Spaghetti-Programmierung

Selektionsbilder, Listen und Nachrichten (GUI I)

Seite 447

Es ist so einfach, eine SAP-Oberfläche zu programmieren: Ein Selektionsbild erstellen, eine Liste ausgeben, vielleicht noch ein paar Nachrichten an den Anwender verschicken – und fertig ist das berühmt-berüchtigte SAP GUI! Das klingt nach Spaghetti-Code und ist damit ganz nach dem Geschmack von Schrödinger. Zum Einstieg in die GUI-Programmierung ist es der beste Weg, einfach mal ein paar Elemente auf dem Bildschirm auszugeben. Für das richtige Ambiente sollen dabei Klettererdbeeren und Vogerlsalat sorgen ..., sagt der Schwager Roland.

Vom kleinen Window-Farmer bis zum Groß-		Mehr, Mehrere, Mehreres (ohne Rauschen)	470
GUI-Besitzer: Jeder will ein Selektionsbild!	448	Zur Anwendung gebracht ... fast	474
Selektionsbild für Beschränkte mit Liste	450	Graue Theorie: Eingabehilfe, Prüftabelle	476
PARAMETERS: Das kleinere der		Entfesse den Künstler in dir:	
Selektionsbild-Ungeheuer	452	Screen-Gestaltung	478
Typ und Vorschlag	455	Endlich Texte!	482
Hey, Checker!	457	Wie im Hamsterrad	484
Rund wie ein Radiobutton	459	Ereignisreich geht's weiter	490
„Du darfst“ war gestern, heute ist „Du musst“	461	Tagesschau ... also eine Nachrichtensendung	493
Abflug in den Keller	462	L – Li – Lis – List – Liste	497
Aber satt war er noch immer nicht:		Keine Beichte notwendig:	
SELECT-OPTIONS	465	Interaktion mit einer Liste	504

Kapitel 13: Lasagne aufs Auge

Dynpro-Programmierung (GUI II)

Seite 507

„Wow, ein dynamisches Programm!“ Das gefällt Schrödinger sofort, dieses Dynpro. Dynamik ist einfach klasse, außer natürlich, es geht um Sport. Mit der Ablauflogik, Elementen zur Gestaltung der Darstellung und Eigenschaften, die man selbst programmieren kann: Super! Das ist schon richtig modular. (Ja, so langsam gewöhnt sich Schrödinger an die Feinheiten im ABAP-Vokabular.) Und es sind auch schon Schichten erkennbar, die Bausteine zur Dynpro-Programmierung sehen fast schon aus wie eine geschichtete MVC-Architektur. Und Schrödinger liebt Schichten, vor allem in Form von Lasagne!

Dynamisch programmieren	508	Die Gestaltung eines eigenen Menüs	531
Wer schreit hier so? Rahmenprogramme,		Kannst du mit dem Druck umgehen?	534
die Dynpros rufen!	512	FFFFF	537
Dreieinigkeit	515	Weck den Künstler in dir	539
Alles ist im Fluss und manchmal geht es		Dein Dynpro	547
im Kreis	517	Sammle Elemente	549
Wohin soll ich mich wenden?	521	Wohin mit den Daten?	551
Wo Module bestimmen	522	Ablauflogik Reloaded	553
Mit welchem Titel darf ich dich ansprechen?	525	Dynpro rufen mal anders	561
Über der Fläche steht die Oberfläche	528		

Kapitel 14: Ravioli

Web-Dynpro-ABAP-Programmierung (GUI III)

Seite 565

Heute wartet ein besonderer Leckerbissen auf Schrödinger: Web Dynpro ABAP. Webanwendungen erstellen ohne HTML-Kenntnisse. Das klingt gut, findet Schrödinger. Ist aber für den ABAP-Programmierer eher ein i-Tüpfelchen. Kann man gut online lesen. Also: Schau auf der Bonus-Seite unter <http://www.sap-press.de/4741>

**Materialien zum
Herunterladen:**

<http://www.sap-press.de/4741>

Kapitel 15: Wie Phönix aus der Asche

SAPUI5 oder wie baue ich eine moderne Webanwendung (aus Sicht von SAP ;-))

Seite 567

Schrödinger hat was Neues entdeckt: Webanwendungen mit HTML5, JavaScript, JQuery und CSS3. Wow, und das alles fürs SAP-System! Na, das kann ja lustig werden.

SAP Ju-Ai-Feif	568	Destinationen muss man haben	583
Der kleinstmögliche SAPUI5-Werkzeugkoffer	571	Was ist Ihre Destination, Herr Schrödinger?	584
Hello OData-World!	575	Eine Architektur für SAPUI5-Anwendungen	586
Bei Aufruf Service (und dessen Analyse)	578	Wir generieren uns eine Anwendung	587
Jetzt aber: Daten anzeigen lassen	581		

Kapitel 16: Raus aus meiner Küche!

Berechtigungen

Seite 593

„Hör mal, Schwaiger Roland, zu viele Köche verderben doch bekanntlich die Nudelsoße. Gibt es Möglichkeiten in ABAP, dass nicht jeder alles darf? Also dass ich bestimmten Anwendern manche Aktionen verbieten oder erlauben darf? Wenigstens einschränken?“ Der Schwaiger Roland lächelt nur weise.

Berechtigungsroundumschlag – Überblick			
Berechtigungen	594	Der Berechtigungs-Selbstchecker – S_TCODE prüfen	602
Am Anfang steht das Objekt mit Klasse – Berechtigungsobjekt	597	Experimente mit AUTHORITY-CHECK	604
Die Details zur Berechtigung	600	Warum mag mich keiner – SU53	605
		Finger weg von meinem View!	607



Kapitel 17: Vorratskammer einrichten mit ziemlich viel Schnickschnack

DB-Tabellen erstellen

Seite 611

Was geschieht, wenn Anwender alle Daten im GUI eingegeben haben? Wohin gehen die Daten dann? Die müssen doch gespeichert werden? Denn die Anwender werden wahrscheinlich nicht immer wieder die gleichen Daten eingeben wollen – außer wenn sie an einem schwachen Kurzzeitgedächtnis leiden. Das würde sich Schrödinger öfter wünschen, das mit dem Kurzzeitgedächtnis, denn dann könnten sich die Anwender nicht an seine Programmierfehler erinnern ..., aber der Schwaiger Roland erklärt ihm dann doch lieber, wie er eine SAP-Datenbank-tabelle erstellt.

Freiland-Datenhaltung – Daten persistieren	612	Ich will auch anders suchen – Sekundärindex	638
Warum einfach, wenn es mit Schnittstelle geht –		Ändern oder nicht, was geht –	
Die Datenbankschnittstelle	614	Erweiterungskategorie	641
Transparente Tabelle en Detail	617	Definieren und Realisieren –	
Spalten und der Rest – Tabellenfelder	620	Datenbankobjekt	643
Ohne Technik keine Tabelle – Technische Einstellungen	625	Mein erster Eintrag – Datenbanktabellen- Einträge erzeugen	644
Mehr als eine Tabelle	630	Artenvielfalt im Dictionary –	
Welcher Wert ist möglich – Fremdschlüssel	633	Weitere Tabellenarten	646

Kapitel 18: Vorratskammer plündern

DB-Tabellen auslesen

Seite 647

Daten rein, Daten raus, und das möglichst einfach: So wünscht sich das der Schrödinger. Da kann ihm der Schwaiger Roland helfen. Und alleine die Begriffe „Open SQL“ und „ANSI SQL“ klingen wie Musik in seinen Ohren. Und er kann dann Daten aus einer oder sogar mehreren Datenbanktabellen lesen. Manchmal wünscht sich Schrödinger, er könnte den Schwaiger Roland in der Vorratskammer einsperren. Nur über Nacht.

Erster Takt – SQL	648	Open SQL grundiert – Basisbefehlssatz	
Zweiter Takt – SQL	648	aus Open SQL	651
Eingelagert und geplündert – Datenpufferung	649	Verklausuliert – SELECT-Klausel	653
Datenmanipulator nativ – DML mit Native SQL	650	Wohin damit – INTO target	656
Einfach definieren – Datendefinition mit dem ABAP Dictionary	650	Woher – FROM source	657
Alles recht und schön – Berechtigungen und Konsistenzprüfungen	651	Aber bitte nicht alles – WHERE-Klausel	659
		Da geht noch mehr – Weitere Klauseln	660
		Und jetzt alles kombiniert – SELECT-Beispiel	661

Mehr als eine Tabelle ist auch okay – Views	666	Mit Views programmieren	682
Schritt für Schritt zum View mit relationalen Operatoren	670	Weil du so brav warst – Programmieren mit Views	684
Da will ich auch nicht alles – Selektionsbedingungen	679	Die CDS-Beziehungskiste: Von Joins und Assoziationen	686
Zusammenfassung in Bildern – View-Definition	680	Mit CDS-Views programmieren	688

Kapitel 19: Vorratskammer in Schuss halten

DB-Daten pflegen

Seite 691

Daten pflegen – den Ausdruck mag Schrödinger. Und denkt an die Kräutersammlung auf seinem Fensterbrett. Da müssen manchmal auch neue Kräuter eingefügt werden, mal muss er das eine durch das andere austauschen. Und für das Löschen hat er auch schon eine leckere Soßenidee. Datenpflegeservice Schrödinger!

Lesen ist Silber, Schreiben ist Gold –		Und noch eins und noch eins und ... – INSERT	706
DELETE, UPDATE, MODIFY und INSERT	692	Massendaten einfügen	708
Sichere Daten – Transaktionskonzept	692	Ändern muss man auch ab und zu – UPDATE	711
Von einem Zustand zum nächsten –		Massendaten	712
Datenbank-LUW	694	Weg damit – DELETE	714
Bist du konsistent?	697	Kombianweisung – MODIFY	717
Aktionen bündeln – SAP-LUW	698	Halt, Sperre! – Sperrkonzept	719
Und jetzt auch noch transaktional –		Die fünf Gebote der performanten DB-Programmierung	720
SAP-Transaktion	699		
Datenmanipulator-Entwicklungsrahmen – Das Programm für die Open-SQL-Anweisungen	701		

Kapitel 20: Mit Schirm, Charme – und vielleicht noch einem Melonensorbet

Daten in Dateien, Datenablage ohne DB

Seite 721

„Warst du schon einmal bei einem richtig guten Italiener?“ Schrödinger ahnt nichts Gutes. „Also so einem, wo du bereits am Eingang dem Kellner deine Garderobe abgeben kannst.“ – „Ja, das Prinzip der Garderobe ist mir bekannt, wieso?“ – „Na ja, wäre es nicht klasse, wenn die Anwender in ABAP auch Dateien einlesen oder herunterladen könnten – also an der Garderobe abgeben könnten?“ Au Backe, das ist ja mal eine miese Metapher. Aber es ist vielleicht trotzdem wichtig zu wissen, wie man mit Daten umgeht, wenn man kein Datenbanksystem hat.

Daten ohne Datenbank	722	Download now!	732
GUI-Loads – Upload und Download	722	Einen hamma noch – Upload	735
Frontend Services	724	Daten auf dem Applikationsserver	737
Pfad ermitteln	727	OPEN House – OPEN DATASET	738
Download now, zumindest vorbereitet –		DatenTRANSFER – Daten schreiben	739
Download vorbereiten	729	READ DATASET – Daten lesen	741
Auch nett für Datenwiederverwendung –			
Clipboard	730		

Kapitel 21: Täglich wechselnde Speisekarten

Dynamische Programmierung

Seite 743

Morgens frische Lebensmittel auf dem Markt einkaufen, mittags daraus eine Speisekarte zaubern. Jeden Tag neu. Schrödinger bekommt Hunger, aber darum kann es dem Schwaiger Roland jetzt ja nicht gehen. „Was genau ...“ – „Wenn du erst während der Laufzeit eines Programms die Informationen erhältst, die für die Ausführung des Programms nötig sind, zum Beispiel den Namen einer Datenbanktabelle, dann ...“ – „Okay, okay, ein neues Konzept. Sag mir, wie es heißt, aber dann geht es in die Kantine.“

Dynamische Programmierung hat nichts mit Beweglichkeit zu tun	744	Dynamisches Feld	753
Feldsymbole	744	Dynamischer Typ	753
Datenrefs	746	Dynamische Komponente	754
RTTS = RTTI + RTTC	748	Dynamische Bedingung	754
Überschrift finden	749	Dynamisches Unterprogramm	756
Dynamische Tokens	751	Programmerzeugung im Hauptspeicher	756
		Programmerzeugung im Repository	757

Bildnachweis

Index