ABAP in Eclipse: Abschied von Transaktion SE80

Grundrenovierung

Ralf Wenzel

ABAP-Entwickler, die sich in ihrer Welt gemütlich eingerichtet haben, müssen umlernen. Da SAPs proprietäre Entwicklungswerkzeuge in die Jahre gekommen sind, setzt das Softwarehaus mit den ABAP Development Tools nun auf Eclipse.

Für SAP-Entwickler brechen neue Zeiten an: Business Server Pages, Web Dynpro und PDF-Formulare haben noch nicht überall beim Kunden Einzug gehalten, da folgen schon HANA und SAPUI5 – Techniken, denen man mit ABAP-Mitteln nicht mehr beikommt. Dem heterogenen Programmieren mit Java, JavaScript, SQLScript, HTML5 und CSS3 dürfte die Zukunft gehören.

Sogar von ihrem primären Arbeitsmittel müssen sich die SAP-Entwickler sukzessive verabschieden. Einem kleinen gallischen Dorf gleich arbeiten sie nicht etwa mit Eclipse, sondern mit der proprietären Transaktion SE80, die der Walldorfer Konzern schon seit 2012 nicht mehr weiterentwickelt. Neuerungen gibt es nur bei den Eclipse-basierten Nachfolgern, namentlich den ABAP Development Tools für Eclipse (ADT) sowie dem HANA Studio für die In-Memory-Datenverarbeitung.

In diesem Prozess entledigt sich das Softwarehaus einiger klassischer Techniken, indem es keinerlei Kapazitäten mehr in die prozedurale und die Dialogprogrammierung mit Dynpros investiert. SAP richtet sich deutlich in Richtung SAPUI5 (OpenUI5) aus. Und für das Programmieren in diesem Umfeld sind die Eclipse-IDEs gut geeignet (siehe Kasten „SAP zum Ausprobieren“).

Frühere Stärken der SAP-Entwicklungsumgebung (wie der kompletten SAPGUI) haben an Bedeutung verloren, denn sie sollten vor allem die Schwächen früherer IT-Landschaften kompensieren. Die Ära leistungsarmer Clients an kleinen Monitoren mit niedriger Auflösung ist jedoch längst vorbei, Netzwerke haben inzwischen enorme Bandbreiten, selbst über Kontinente hinweg. Stattdessen kommen nun die konzeptionellen Nachteile zum Tragen: Der Artikel „High Definition SAP“ von Enno Wulff zeigt deutlich, dass man auf großen, zeitgemäßen Displays mit SAP kaum arbeiten kann; zu kleine Icons in unpassender Auflösung, viel verschwendeter Platz, die Liste der Mängel ist lang (siehe „Alle Links“ am Ende des Artikels).

Antiquierter Werkzeugkasten

Die Werkzeuge innerhalb der SE80 sind ebenfalls veraltet, etwa der umständliche Refactoring-Assistent. Und das endlose Herumgeklicke in der Formularansicht zwischen Methodendefinition, Parametern und Ausnahmen, um in den Editor zu gelangen, in dem man die Implementierung schreibt, entspricht ebenfalls nicht modernen Standards. Zahlreiche Entwickler arbeiten schon jetzt am liebsten in der sogenannten Quelltextansicht, die nichts anderes ist als ein Editor, in dem sie einfach alles hintereinanderschreiben. Zudem sind die Anforderungen gestiegen: Es gibt Add-on-Entwickler, die gern eigene Frameworks und Entwicklungsumgebungen einsetzen

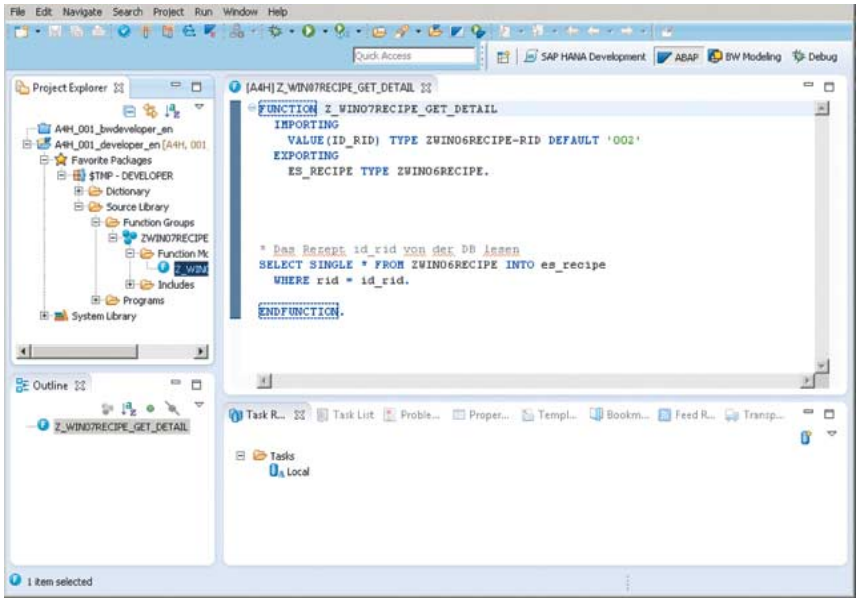
iX-TRACT

- SAPs Entwicklungstool, die Transaktion SE80, ist veraltet und wird seit zwei Jahren nicht mehr weitergepflegt.
- SAPUI5 und andere moderne Techniken zwingen die SAP-Programmierer zum Umlernen: Das Softwarehaus baut mit den ABAP Development Tools auf Eclipse.
- Mit den Eclipse-Werkzeugen können Entwickler moderne SAP-Anwendungen erstellen. Da sich einstmals abgeschottete ERP-Landschaften immer mehr öffnen, ist der Schritt zu der IDE überfällig.

möchten – und das ermöglichen nun die ABAP Development Tools (siehe Kasten „Eclipse in a Nutshell“).

Nach dem Einrichten der ADT und dem Aufruf der ABAP-Perspektive legt man zunächst ein neues Projekt an (Dateimenü -> *New ABAP Project*). Der Project Explorer View ähnelt der Baumdarstellung der SE80 auf Paketebene und ist daher kaum erklärungsbedürftig. Hinter dem Icon *Open ABAP Development Object* verbirgt sich eine mächtige Funktion, die das System nach eingegebenen Mustern durchsucht. Der Entwickler kann mehrere Objekte anklicken und gleichzeitig in verschiedenen Editor-Tabreitern öffnen (Abbildung 1). Der Outline View zeigt die Hierarchie innerhalb eines Objekts (auch direkt im Code via *Ctrl+O*) in einem Fenster an, genannt *Quick Outline*. Hat der Entwickler die Funktion *Link with Editor* gewählt (was empfehlenswert ist), wird im Hierarchiebaum mitnavigiert, wenn der Benutzer im Editor das Entwicklungsobjekt wechselt oder in ihm scrollt.

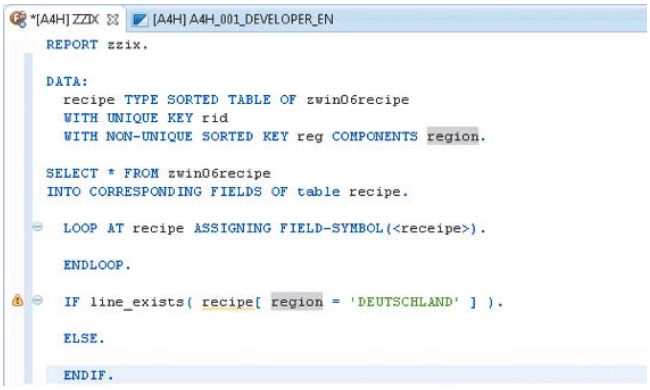
Da er es jetzt mit einer reinen Client-Anwendung zu tun hat, kann er mehrere Sitzungen auf mehreren SAP-Systemen in einem Programm öffnen und alle im selben Programm bearbeiten. Bisher war es notwendig, in jedem System die SE80 aufzurufen. Sollte die Verbindung zum System verloren gehen, fliegt einem die Session nicht (wie bei der SAPGUI) um die Ohren, sondern Eclipse versucht im Hintergrund, den Kontakt wiederherzustellen. Es ist sogar möglich, dabei weiterzuarbeiten.



Die ABAP-Perspektive: Links oben sieht man den Project Explorer View, links unten der Outline View, rechts oben die Editoren in Tabreitern (Abb. 1).

ABAP 7.40 mit vielen Neuerungen

New -> ABAP Program erstellt ein neues Programm, der Einfachheit halber als lokales Objekt, dann ist kein Transportauftrag notwendig (Abbildung 2). Die Syntax dürfte dem einen oder anderen fremd vorkommen – es handelt sich um ABAP 7.40. Diese Version bringt die Programmiersprache einen großen Schritt nach vorn: Deklarationen lassen sich inline im Code anbringen (statt über explizite deklarative Kommandos wie *DATA*), es sind kompakte Ausdrücke für Operationen auf internen Tabellen erlaubt und vieles mehr. Horst Keller erklärt ABAP 7.40 in seinem SCN-Blog ausführlich (siehe „Alle Links“).



So sieht ein neu erstelltes Programm in ABAP-7.40-Code aus (Abb. 2).

Für Klassen gibt es den Class Wizard. Nach *New -> ABAP Class* wählt der Entwickler den Transportauftrag (sofern gewünscht), und das Codegerüst für eine Klasse wird im Quelltext eingefügt. Entwicklungsobjekte löscht er über den Project Explorer, ganz wie von Desktop-Anwendungen gewohnt (also per Kontextmenü oder *Entf*-Taste). Die Funktionen sind dieselben wie in der Transaktion SE80.

Wenn der Entwickler zum Beispiel einen *Pretty-Printer*-Lauf (für markierte Codeteile per *Ctrl+Shift+F1* oder für das ganze Entwicklungsobjekt per *Shift+F1*) startet, passiert Folgendes: Eclipse schickt die Rohtextdatei an das Backend, das den Pretty Printer ausführt und den aufbereiteten Code sogleich an die IDE zurücksendet, die ihn gegen den alten austauscht. Diese Vorgehensweise stellt sicher, dass die SAP-Entwickler das Rad nicht neu erfinden müssen.

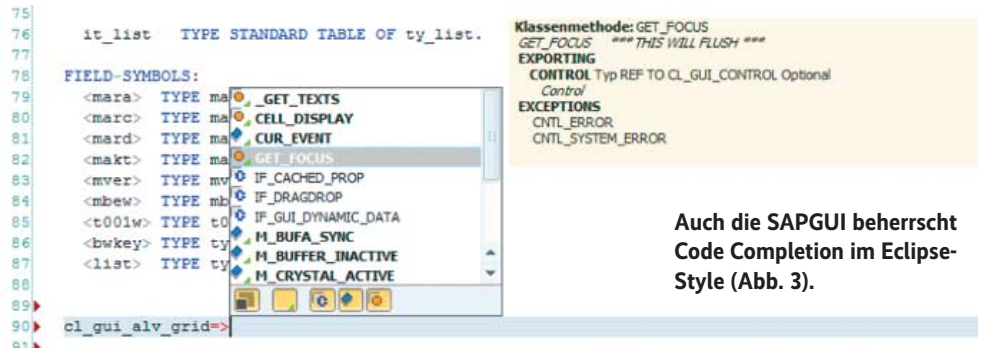
Bei dieser Art der Implementierung fand SAP eine Reihe von Fehlern, die zuvor nicht auffielen, weil die SE80 schon lange niemand mehr durchgetestet hatte. Da man die entsprechenden Funktionen aber für Eclipse braucht, hat SAP sie im Backend korrigiert. Das Aktivieren (also das Erzeugen des Laufzeitobjektes) erfolgt ohnehin auf dem Applikationsserver.

Wenn es um den Gebrauch des Editors geht, dürfte die Leserschaft in zwei Fraktionen zerfallen: Eine verwendet den neuen Frontend-Editor und reizt seine Optionen aus, die andere fügt die Vorlagen noch mit dem *Muster*-Button ein und kennt die umfangreichen Code-Completion-Variationen des Editors kaum. Dieser Artikel beschreibt daher die Optionen, obwohl sie

SAP zum Ausprobieren

Da nicht jeder ein SAP-System im Keller stehen hat, gibt es seit einigen Jahren das früher als „miniSAP“ und heute als „NetWeaver Trial“ bekannte Basissystem. Damit soll der Entwickler ABAP erlernen, Modulfunktionen fehlen jedoch. Der Benutzer kann also keine Bestellungen buchen oder Materialstammdaten im System finden. Weitere Informationen über die Trial-Version finden sich im SAP Community Network (SCN) (siehe „Alle Links“).

Da SAP noch eine Weile braucht, alle Eclipse-Erweiterungen zu implementieren, wird weiterhin mit der Vorjahresversion der IDE namens Kepler gearbeitet. SAP schreibt Dokumentationen nicht mehr in deutsch und übersetzt sie, sondern verfasst sie in Englisch und übersetzt sie in absehbarer Zeit nicht.



Auch die SAPGUI beherrscht Code Completion im Eclipse-Style (Abb. 3).

```
cl_gui_alv_grid=>get_focus(
*  IMPORTING
*    control           = control    " Control
*  EXCEPTIONS
*    cntl_error        = 1
*    cntl_system_error = 2
*    others            = 3
).
IF sy-subrc <> 0.
* MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
*            WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
ENDIF.
```

Eingefügte Codevorlage mit Schnittstelle; analog zur deutlich umständlicheren Erzeugung mit der Funktion „Muster einfügen“ in der SAPGUI (Abb. 4)

nicht neu sind. Sie lösen jedoch SAPGUI-Funktionen ab, mit denen viele noch arbeiten, insbesondere bei Mustern und Code Completion. Das bedeutet, dass sich die hier angeführten Beschreibungen sowohl im SAP-Editor als auch in Eclipse abrufen lassen. Zugegebenermaßen hat SAP die Funktion zum Aktivieren und Einstellen der Code Completion und der Coding-Vorlagen gut versteckt. In der Hilfe ist sie jedoch dokumentiert (siehe „Alle Links“).

Muster bei Eclipse ausgemustert

Die Unkenntnis erstreckt sich vor allem auf die Eclipse-Umsetzung der Muster des Editors. Statt der klassischen Musterfunktion via Popup kann der Editor längst ohne Klick-Orgie Namen von Klassen, Objekten und sogar von lokalen Variablen vervollständigen. Entwickler, die das bisher nicht nutzen, müssen umlernen, denn der Button *Muster* existiert nicht in Eclipse.

Statt auf diesen Button zu klicken, beginnt man, den Befehl, den Namen eines Funktionsbausteins, einer Klasse oder einer Variablen zu tippen, und erhält via *Ctrl+Space* eine Auswahl passender Befehlsgerüste, Klassennamen, Variablen et cetera. Sind sie eindeutig, braucht man die Tastenkombination nicht, und der automatisch angezeigte Tooltip ist schwarz. Das Betätigen der Tabulatortaste vervollständigt dann den Ausdruck (Abbildung 3).

Hat sich der Entwickler die Alternativen anzeigen lassen, wie bei Klassen notwendig, die alle mit dem gleichen Präfix anfangen, kann er die jeweilige Dokumentation sehen, wenn er den Mauszeiger auf einen der Vorschläge führt. Bei einer zu langen Liste aktiviert der letzte Punkt die Suchfunktion, in der sich exzessiv mit Wildcards arbeiten lässt. Ein Mausklick (oder Return) ergänzt den Namen der gewählten Funktion, Methode oder Klasse (Abbildung 4). Das Drücken der *Shift*-Taste dabei fügt die gesamte Signatur ein. Das funktionierte schon in der SAPGUI.

Code-Templates für Befehle lassen sich ändern oder neu definieren, indem die entsprechende Sektion in den Editoreinstellungen (*Window -> Preferences*) gewählt wird. Dort kann man beispielsweise angeben, was die IDE bei einer CASE-Anweisung einfügen soll und wo der Cursor danach stehen zu bleiben hat. Hier finden sich auch eine Reihe definierter Variablen (Abbildung 5).

Weitere bekannte Operationen arbeiten wie gewohnt, Kommentare fügt man per *Crtl+>* ein und nimmt sie per *Ctrl+<* wieder zurück. Methoden, Klassen und andere Sektionen lassen sich ebenfalls über die Plus-/Minus-Buttons aus- und einklappen (Abbildung 6). Die Darstellung des so ausgeblendeten Codes über ein Popup ist ebenfalls aus der SE80 bekannt. Selbst den Verwendungsnachweis gibt es (*where-used list*), nur hat sich der Button verändert. Die Baumdarstellung des Suchergebnisses erscheint deutlich performanter als die bisheri-

Eclipse in a Nutshell

Im Rahmen dieses Artikels kam ein SAP Release 7.30 zum Einsatz, auf dem ein ABAP Release 7.40 läuft. Eclipse kennt verschiedene Perspektiven, darunter eine für ABAP. Wer an einem HANA-System arbeitet, kann direkt in Eclipse JavaScript-, Java- oder SQL-Code entwickeln, für die ebenfalls eigene Perspektiven existieren. Innerhalb der Perspektive ordnet der Entwickler die einzelnen Views an, zum Beispiel den Project Explorer View, den Outline View oder den ABAP Unit Test Runner View.

Linux- und Macintosh-Anwender dürfen sich glücklich schätzen, mit den ABAP Development Tools (ADT) erhalten sie eine native Möglichkeit, in SAP zu entwickeln. Denn die SAPGUI läuft bekanntlich nur unter Windows, und SAPGUI für Java war nie wirklich zu gebrauchen, unter anderem, weil sie nicht alle Controls unterstützt.

Im Prinzip – und das ist ein wesentlicher Vorteil – sieht ein Eclipse-Fenster immer gleich aus. Shortcuts und Menüs präsentieren sich (sogar plattformübergreifend) ebenfalls im einheitlichen Look. SAP hat nicht versucht, die gewohnten Shortcuts in Eclipse umzusetzen. SAPler müssen daher umlernen, denn es handelt sich bei den ADT nicht um Eclipse für SAP, sondern um eine SAP-Erweiterung für Eclipse.

Beispielsweise führt ein Doppelklick auf einen Methodennamen nicht zu einer Methode, sondern er markiert ein Wort, so wie überall außerhalb der SAP-Welt üblich. Navigieren kann man mit *F3* (der in SAP üblichen Taste zum Verlassen des Programms) oder per Klick mit gedrückter *Ctrl*-Taste. Macht der Entwickler das bei einer Klasse, kann er zwischen Definitions- und Implementierungsteil wählen. *F1* zeigt jedoch immer noch die Hilfe an. Die wichtigste Tastenkombination ist *Ctrl+Shift+L*. Sie blendet eine Liste aller verfügbaren Tastenkombinationen ein. Die sind aus zwei Gründen unentbehrlich: Erstens gibt es einige von der SAPGUI bekannte Buttons in Eclipse nicht, und zweitens helfen Tastaturkommandos bei der Arbeit. Im Vergleich zur SAPGUI gibt es unter *Window -> Preferences* deutlich mehr Optionen für persönliche Einstellungen.

ge ABAP-Liste. Die meisten Funktionen sind zudem über Kontextmenüs erreichbar.

Hilfreich: Templates und Wizards

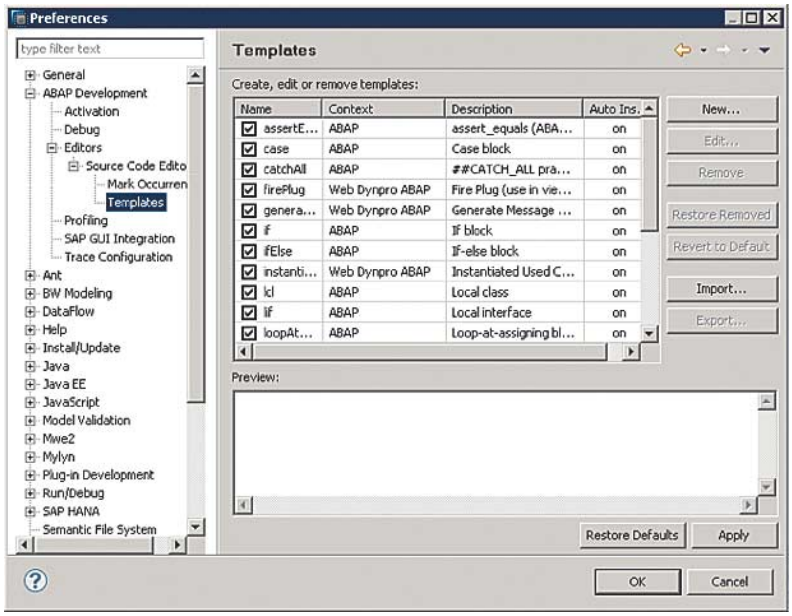
Einige lokale Typendefinitionen, Testklassen und Ähnliches, das bisher im Menü versteckt war, findet sich nun direkt unterhalb des Editorfensters. Bei Testklassen, in denen das Codegerüst den größten Teil einnimmt, helfen wieder die Code-Templates und Wizards (per *Crtl+Space* aufzurufen) (Abbildung 7). Das bedeutet das Ende der althergebrachten Funktion *Suchen+Ersetzen* beim Umbenennen von Variablen – denn über den Wizard ändert man die entsprechenden Stellen konsistent und gleichzeitig. An dieser Stelle zeigt sich, dass SAP deutlich in die objektorientierte Denkweise wechselt – denn für FORM-Routinen funktioniert das nicht, für Methodennamen aber sehr wohl.

Interessant ist die Art und Weise, wie sich Dokumentation und Schnittstellen im Code deklarieren lassen. Hier kommt ABAP Doc zum Einsatz, ein weiterer Fortschritt neben ABAP 7.40. Gemeint ist damit eine besondere Schreibweise von Kommentaren, die so genauso formatiert im Popup angezeigt werden wie die im System hinterlegte Dokumentation für globale Entwicklungsobjekte, für die es ja separate Dokumentationsmittel in SAPscript gibt, beispielsweise die Klassen- oder Methodendokumentationen zu den globalen Klassen (Abbildung 8). Weitere Informationen zu ABAP Doc liegen im SCN (siehe „Alle Links“),

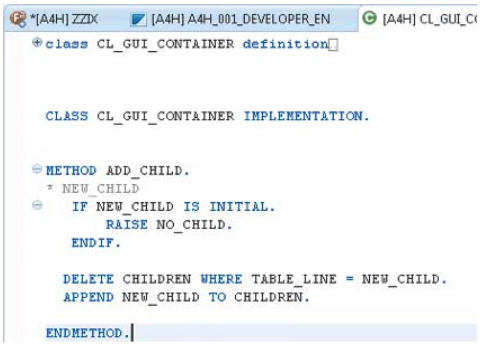
Weitsichtige Programmierer schreiben schon jetzt Kommentare in ABAP Doc und leisten damit gute Vorarbeit, denn irgendwann muss ihr Unternehmen auf Eclipse wechseln, selbst wenn die Bereitschaft dazu derzeit noch nicht mal im Ansatz erkennbar sein sollte. Dann verfügt der Code bereits über die Informationen, die man sich in den ABAP Development Tools via *F2* ansehen kann.

Startet der Benutzer ein an die SAPGUI gebundenes Programm, öffnet sich das Benutzer-Interface inline in einem Eclipse-Fenster. Dies passiert ebenfalls bei (noch) nicht unterstützten Entwicklungsobjekten. Beispiel: Die Tabellenpflege des Data Dictionary ist in Eclipse noch nicht eingeführt, daher öffnet sich die Transaktion SE11 in Eclipse, wenn man auf ein solches Objekt klickt. *Run As -> Unit Test* startet die ABAP Unit Tests (Testklassen), die SAP erfreulicherweise schon komplett umgesetzt hat.

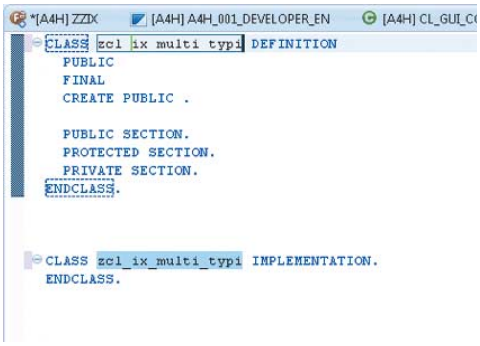
An einigen Punkten sieht man, dass die Eclipse-Erweiterungen noch nicht vollständig sind, zum Beispiel bei der Einbindung des Data Dictionary. Für diese Arbeitsmittel (und für die, die dauerhaft nicht unterstützt werden, wie der Screen Painter) öffnet sich die altbekannte SAPGUI inline in Eclipse. Darunter leidet die Produktivität nicht, denn das, was funktioniert, läuft rund. Mit den Eclipse-Tools kann der Entwickler SAP-Applikationen endlich zeitgemäß erstellen. Und das ist in der zunehmend heterogen SAP-Landschaft dringend geboten. (jd)



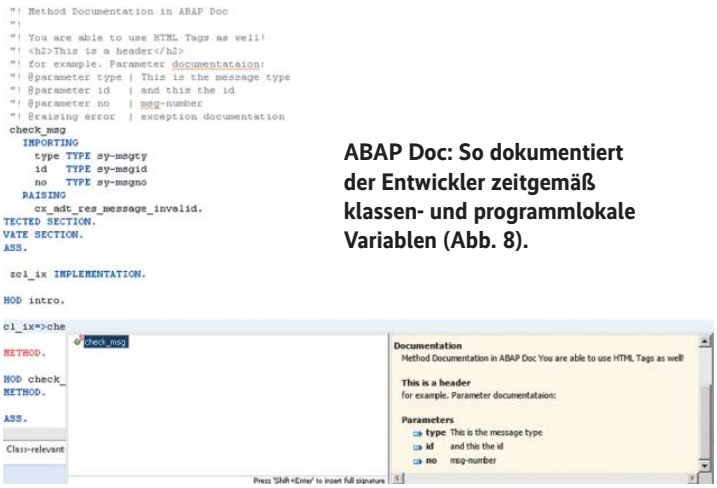
Code-Templates in Eclipse pflegen: Viele Entwickler wissen nicht, dass das auch in der SAPGUI möglich ist (Abb. 5).



Codeblöcke, etwa Klassendefinitionen, lassen sich für den besseren Überblick zusammenklappen (Abb. 6).



Modernes Refactoring: Ändern eines Variablen- oder Klassennamens ändert ihn simultan an allen Verwendungsstellen (Abb. 7).



ABAP Doc: So dokumentiert der Entwickler zeitgemäß klassen- und programmlokale Variablen (Abb. 8).

Ralf Wenzel

betreibt die Unternehmensberatung Heuristika in Hamburg.

Alle Links: www.ix.de/ix1501118