

# **ABAP für Fortgeschrittene**

**Teil 3: Anpassen des SAP-Standards**

## Copyright

- Das vorliegende Skriptum baut zu großen Teilen auf den Publikationen zum TAW11- und TAW12-Kurs – das Copyright dieser Teile liegt bei der SAP AG.
- Die in diesem Kurs verwendeten Abbildungen wurden – falls nicht anders gekennzeichnet – in Anlehnung zum TAW11- und TAW12-Kurs erstellt. Das Copyright dieser Teile liegt bei der SAP AG.
- Für alle Screenshots im Skriptum, auch wenn diese nur verkürzt oder auszugsweise gezeigt werden, gilt der Hinweis: Copyright SAP AG
- Die Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die schriftliche Genehmigung von Prof. Dr. Heimo H. Adelsberger, Dipl.-Wirt.-Inf. Pouyan Khatami und Dipl.-Wirt.-Inf. Taymaz Khatami nicht gestattet..

## Inhaltsverzeichnis

COPYRIGHT .....	2
INHALTSVERZEICHNIS .....	3
ABBILDUNGSVERZEICHNIS .....	5
<b>6 ANPASSEN DES SAP-SYSTEMS.....</b>	<b>8</b>
6.1 ERWEITERUNGEN AN DICTIONARY-ELEMENTEN .....	14
6.1.1 Texterweiterungen.....	14
6.1.2 Erweiterungen an Strukturen und Tabellen.....	17
6.2 CUSTOMER-EXITS.....	20
6.2.1 Vorgehensweise für Erweiterungsprojekte .....	22
6.2.2 Programm-Exits .....	26
6.2.3 Praxis: Beispiel-Suche nach Customer-Exits .....	29
6.2.4 Aufbau von Exit-Funktionsgruppen.....	30
6.2.5 Menü-Exits .....	33
6.2.6 Dynpro-Exits .....	35
6.3 KLASSISCHE BUSINESS ADD INS.....	41
6.3.1 Programm-Exits mit BAdIs.....	43
6.3.2 Praxis: Implementierung eines BAdI .....	49
6.3.3 Menü-Exits mit BAdIs .....	52
6.4 MODIFIKATIONEN .....	54
6.4.1 Grundbegriffe .....	54
6.4.2 Der Modifikationsassistent.....	58
6.4.3 Der Modification Browser .....	61
6.4.4 Der Note Assistant .....	63
6.4.5 User-Exits.....	66
6.4.6 Modifikationen abgleichen.....	68
6.4.7 Fazit: Modifikationen.....	71
6.5 ERWEITERUNGEN DURCH DAS NEUE ERWEITERUNGSKONZEPT .....	72
6.5.1 Enhancement Points .....	72
6.5.2 Praxis: Beispiel zur Verwendung von Enhancement Points in Programmen .....	75
6.5.3 Praxis: Beispiel für implizite Erweiterungsmöglichkeiten bei Klassen .....	77
6.5.3.1 Hinzufügen eines Attributs .....	78
6.5.3.2 Definition eines post-exits .....	78
6.5.4 Explizite Enhancement Points und Enhancement Sections .....	80
6.5.5 Kernelbasierte BAdIs .....	81
6.5.6 Praxis: Beispiel zur Verwendung eines filterabhängigen, kernelbasierten BAdI	
85	
6.5.7 Das Switch-Framework.....	88
6.6 KONTROLLFRAGEN .....	91
6.7 ANTWORTEN.....	92
6.8 KAPITELABSCHLUSS .....	93



## **Abbildungsverzeichnis**

Abbildung 1: Anpassungsmöglichkeiten in einem SAP-System .....	8
Abbildung 2: Entscheidungsbaum zur Auswahl einer Anpassungsmöglichkeit .....	9
Abbildung 3: Änderungsebenen aus Sicht der ABAP Workbench .....	10
Abbildung 4: Funktionsweise von Programm-Erweiterungen .....	11
Abbildung 5: Menü-Exits.....	12
Abbildung 6: Dynpro-Exits.....	13
Abbildung 7: Modifikationen.....	14
Abbildung 8: Erweiterung der Dokumentation eines SAP-Datenelements.....	15
Abbildung 9: Ersetzen oder Ergänzen der Dokumentation? .....	15
Abbildung 10: Pflege von Feldbezeichnern im Dictionary: SAP-System-Screenshot.....	16
Abbildung 11: SAP-Feldbezeichner Überlagern.....	16
Abbildung 12: Überlagerte Feldbezeichner bei Releasewechsel .....	17
Abbildung 13: Tabellenerweiterungen: Append-Strukturen und Customizing Includes .....	18
Abbildung 14: Append-Struktur vs. Customizing Include .....	18
Abbildung 15: Customizing-Includes .....	19
Abbildung 16: Neue Felder und Append-Strukturen .....	20
Abbildung 17: SAP-Anwendungserweiterungen .....	21
Abbildung 18: Erweiterungen und Erweiterungsprojekte .....	21
Abbildung 19: Aufbau von Erweiterungen und Erweiterungsprojekten .....	22
Abbildung 20: Vorgehensweise des SAP-Anwendungsentwicklers .....	23
Abbildung 21: Vorgehen auf Kundenseite .....	23
Abbildung 22: Anlegen eines Erweiterungsprojekts: SAP-System-Screenshot.....	24
Abbildung 23: Suche nach Erweiterungen: SAP-System-Screenshot.....	25
Abbildung 24: Aktivierung / Deaktivierung eines Projekts: SAP-System-Screenshot .....	25
Abbildung 25: Programm-Exits .....	26
Abbildung 26: Architektur eines Programm-Exits.....	27
Abbildung 27: Aufruf des Exitfunktionsbausteins aus dem SAP-Programm.....	27
Abbildung 28: Definition eines Exit-Funktionsbausteins .....	28
Abbildung 29: Aufruf des Quelltextes per System-Status: SAP-System-Screenshot .....	28
Abbildung 30: Suchmaske für Erweiterungen: SAP-System-Screenshot .....	29
Abbildung 31: Ergebnis der Suche: SAP-System-Screenshot .....	29
Abbildung 32: Aufbau einer Exit-Funktionsgruppe.....	30
Abbildung 33: Globale Datendefinitionen in einer Exit-Funktionsgruppe .....	31
Abbildung 34: SAP-Objekte in einer Exit-Funktionsgruppe .....	32

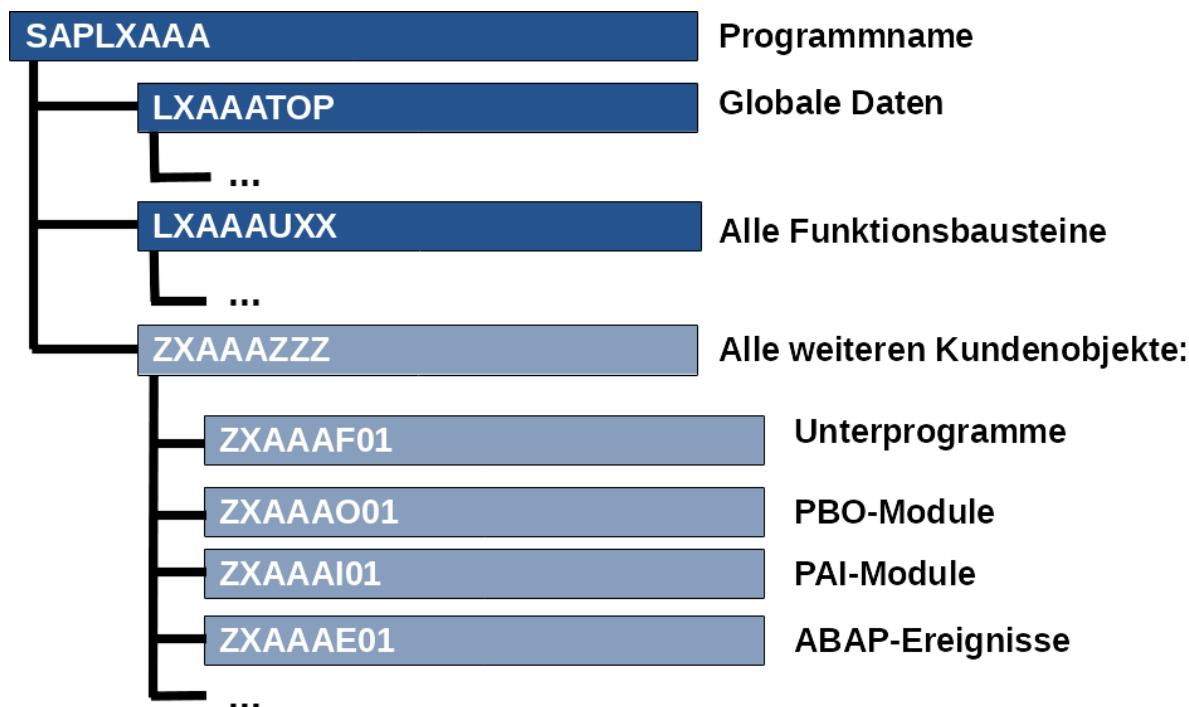


Abbildung 35: Kundeneigene Objekte in einer Exit-Funktionsgruppe .....	32
Abbildung 36: Beispiel für die Einbindung von Dynpros durch Programm-Exits.....	33
Abbildung 37: Menü-Exits.....	33
Abbildung 38: Menü mit reservierter Kundenfunktion .....	34
Abbildung 39: Menü-Exit, dass ein Programm-Exit verwendet .....	34
Abbildung 40: Festlegen des Menütexts für ein Menü-Exit.....	35
Abbildung 41: Dynpro-Exits .....	35
Abbildung 42: Subscreen Exits .....	36
Abbildung 43: Syntax zum Aufruf eines (normalen) Subscreens .....	36
Abbildung 44: Beispiel zur Einbindung per Dynpro-Exit.....	37
Abbildung 45: Aufruf der PBO- / PAI-Module .....	38
Abbildung 46: Datentransport zum Subscreen-Dynpro .....	39
Abbildung 47: Datentransport vom Subscreen-Dynpro .....	40
Abbildung 48: Name und Bearbeitung eines Dynpro-Exits .....	41
Abbildung 49: Veränderte Software-Auslieferungswege.....	41
Abbildung 50: Architektur klassischer BAdIs im Kontext der Auslieferungskette .....	42
Abbildung 51: Aufruf eines BAdIs auf Seiten des SAP-Programms .....	44
Abbildung 52: Komponenten von BAdIs.....	45
Abbildung 53: Ausführung der Default-Implementierung .....	46
Abbildung 54: Erweiterbare Filtertypen.....	47
Abbildung 55: Anlegen von BAdIs: SAP-System-Screenshot.....	48
Abbildung 56: Implementierung von BAdI-Methoden .....	48
Abbildung 57: Aktivieren / Deaktivieren von BAdI-Implementierungen: SAP-System-Screenshot.....	49
Abbildung 58: Öffnen des Programms: SAP-System-Screenshot .....	50
Abbildung 59: Menü-Exit mit BAdI .....	52
Abbildung 60: Für Kundenfunktion reservierter Funktionscode im Menü .....	52
Abbildung 61: Mehrfachverwendung? .....	53
Abbildung 62: Filterabhängiger BAdI-Aufruf .....	53
Abbildung 63: Original vs. Kopie .....	55
Abbildung 64: Notwendigkeit des Modifikationsabgleichs .....	56
Abbildung 65: Abfrage des Modifikationsschlüssels .....	57
Abbildung 66: Versionsverwaltung beim modifizierten von SAP-Objekten .....	58
Abbildung 67: Vorteile des Modifikationsassistenten .....	58

Abbildung 68: Funktionsweise des Modifikationsassistenten.....	59
Abbildung 69: Modifikationen entlang der Software-Auslieferungskette .....	59
Abbildung 70: Änderungen mit dem Modifikationsassistenten .....	60
Abbildung 71: Modifikationsübersicht .....	61
Abbildung 72: Einstiegsbild des Modification Browsers: SAP-System-Screenshot.....	62
Abbildung 73: Ergebnisanzeige des Modification Browsers: SAP-System-Screenshot .....	62
Abbildung 74: Übersicht über den Note Assistant .....	64
Abbildung 75: Neue Hinweise im Note Assistant: SAP-System-Screenshot.....	64
Abbildung 76: Aufbau eines Modulpools mit User-Exits .....	67
Abbildung 77: Technische Realisierung von User-Exits .....	67
Abbildung 78: Schnittmenge, für die ein Abgleich erforderlich ist .....	68
Abbildung 79: Abgleich und Transport .....	69
Abbildung 80: Einstiegsbild des Modifikationsabgleichs (SPAU): SAP-System-Screenshot .....	70
Abbildung 81: Liste abzugleichender Objekte: SAP-System-Screenshot .....	70
Abbildung 82: Implizite vs. explizite Enhancement Points .....	73
Abbildung 83: Anzeige eines impliziten Enhancement Points: SAP-System-Screenshot .....	74
Abbildung 84: Schaltfläche zur Erweiterungsimplementierung: SAP-System-Screenshot .....	75
Abbildung 85: Öffnen des Programms: SAP-System-Screenshot .....	75
Abbildung 86: Ausgabe des Programms ohne Erweiterung: SAP-System-Screenshot.....	76
Abbildung 87: Erweiterungsimplementierung anlegen: SAP-System-Screenshot.....	76
Abbildung 88: Enhancement im Code: SAP-System-Screenshot .....	77
Abbildung 89: Ausgabe des Erweiterten Programms: SAP-System-Screenshot .....	77
Abbildung 90: Öffnen der Klasse: SAP-System-Screenshot .....	77
Abbildung 91: Leere Post-Methode: SAP-System-Screenshot .....	79
Abbildung 92: Explizite Enhancement Points und Enhancement Sections.....	80
Abbildung 93: Enhancement Spots .....	81
Abbildung 94: Architektur klassischer BAdIs .....	82
Abbildung 95: Aufruf klassischer BAdIs .....	83
Abbildung 96: Architektur neuer, kernelbasierter BAdIs .....	84
Abbildung 97: Aufruf neuer (kernelbasierter) BAdIs .....	85
Abbildung 98: Öffnen des Programms: SAP-System-Screenshot .....	86
Abbildung 99: Ausgabe des Programms: SAP-System-Screenshot .....	86
Abbildung 100: Navigation zur implementierenden Klasse: SAP-System-Screenshot .....	87
Abbildung 101: Einstellen der Filterung: SAP-System-Screenshot.....	88
Abbildung 102: Ausgabe bei aktiver Implementierung: SAP-System-Screenshot .....	88
Abbildung 103: Elemente des Switch Frameworks .....	89
Abbildung 104: Business Function Sets und Business Functions (Transaktion SFW5): SAP-System-Screenshot .....	89

## 6 Anpassen des SAP-Systems

Das ERP-System der SAP bietet eine große Fülle von Funktionalitäten, die den Einsatz in verschiedensten Unternehmen unterschiedlicher Branchen und Größen erlauben. Um das System an die Bedürfnisse des jeweiligen Unternehmens anzupassen, gibt es eine ganze Palette von Möglichkeiten.

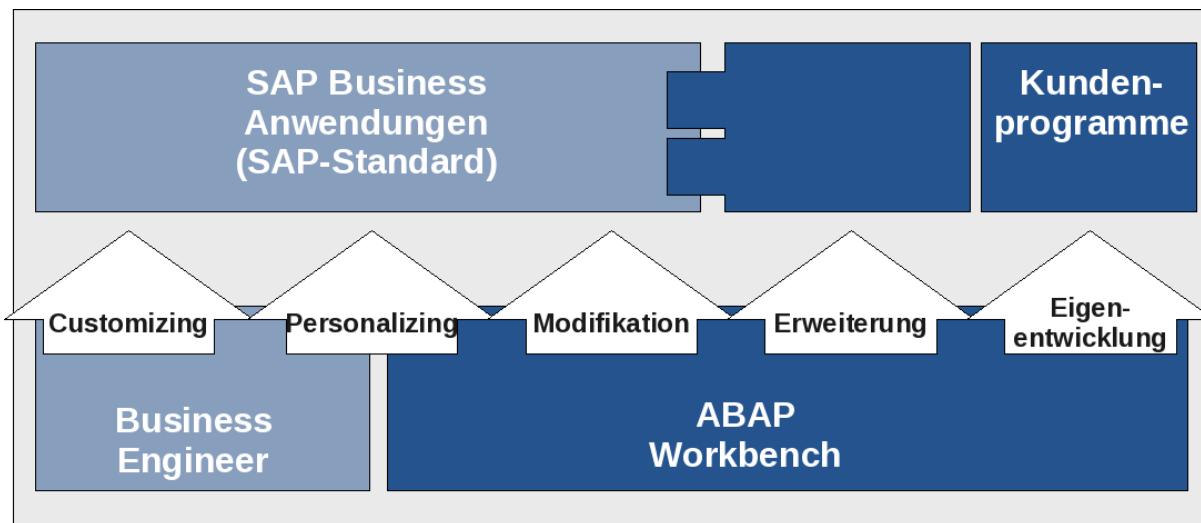


Abbildung 1: Anpassungsmöglichkeiten in einem SAP-System

Die obige Abbildung skizziert die Anpassungsmöglichkeiten, die Ihnen das SAP-System bietet. Im Einzelnen sind dies:

- **Customizing:** Von SAP vorgedachte Änderungen, bei denen Geschäftsprozesse und Funktionen im SAP-System anhand eines Einführungsleitfadens an die Bedürfnisse angepasst werden.
- **Personalizing:** Anpassung von Anzeigeeigenschaften von Feldern an die Bedürfnisse des Benutzers, etwa durch das vorbelegen von Feldern mit Vorgabewerten oder das ausblenden nicht benötigter Felder; sowie die Anpassung der Menüführung.
- **Modifikation:** Änderungen an Repository-Objekten der SAP auf einem System des Kunden. Bei erscheinen einer neuen Version des Objekts seitens der SAP muss die Kundenversion abgeglichen werden. Seit Release 4.5A steht hierfür der Modifikationsassistent zur Verfügung, bis Release 4.0B musste der Abgleich hingegen manuell mit Upgrade-Utilities durchgeführt werden.
- **Erweiterung:** Ergänzung von Repository-Objekten der SAP durch Einbindung von Repository-Objekten des Kunden.
- **Eigenentwicklung:** Kundenspezifische Repository-Objekte, die in einem von SAP-Objekten abgetrennten Namensraum liegen.

Modifikationen, Eigenentwicklungen und Erweiterungen fallen in den Aufgabenbereich der ABAP Workbench, während Customizing und weite Teile des Personalizing in den Aufgabenbereich des **Business Engineers** fallen.

Der Business Engineer fasst die Einführungswerzeuge der SAP zusammen. Dies sind insbesondere Referenzmodelle (Prozessmodell, Datenmodell, Organisationsmodell) und der Einführungsleitfaden (Liste der Customizingaktivitäten). Die Möglichkeiten, außerhalb der

ABAP Workbench Anpassungen vorzunehmen sind nicht zu unterschätzen. Durch Personalizing können etwa Menüs und Anzeigeeigenschaften beeinflusst, Felder, Spalten von Table Controls oder ganze Dynpros aus Transaktionen ausgeblendet werden, so dass Geschäftsvorfälle im SAP ERP-System schneller und einfacher durchgeführt werden können und den Bedürfnissen einzelner Benutzer bei der Arbeit mit dem System Rechnung getragen werden kann.

Zur Auswahl, welche Möglichkeit zur Anpassung für eine gegebene Problemstellung verwendet werden sollte, kann der folgende Entscheidungsbaum verwendet werden:

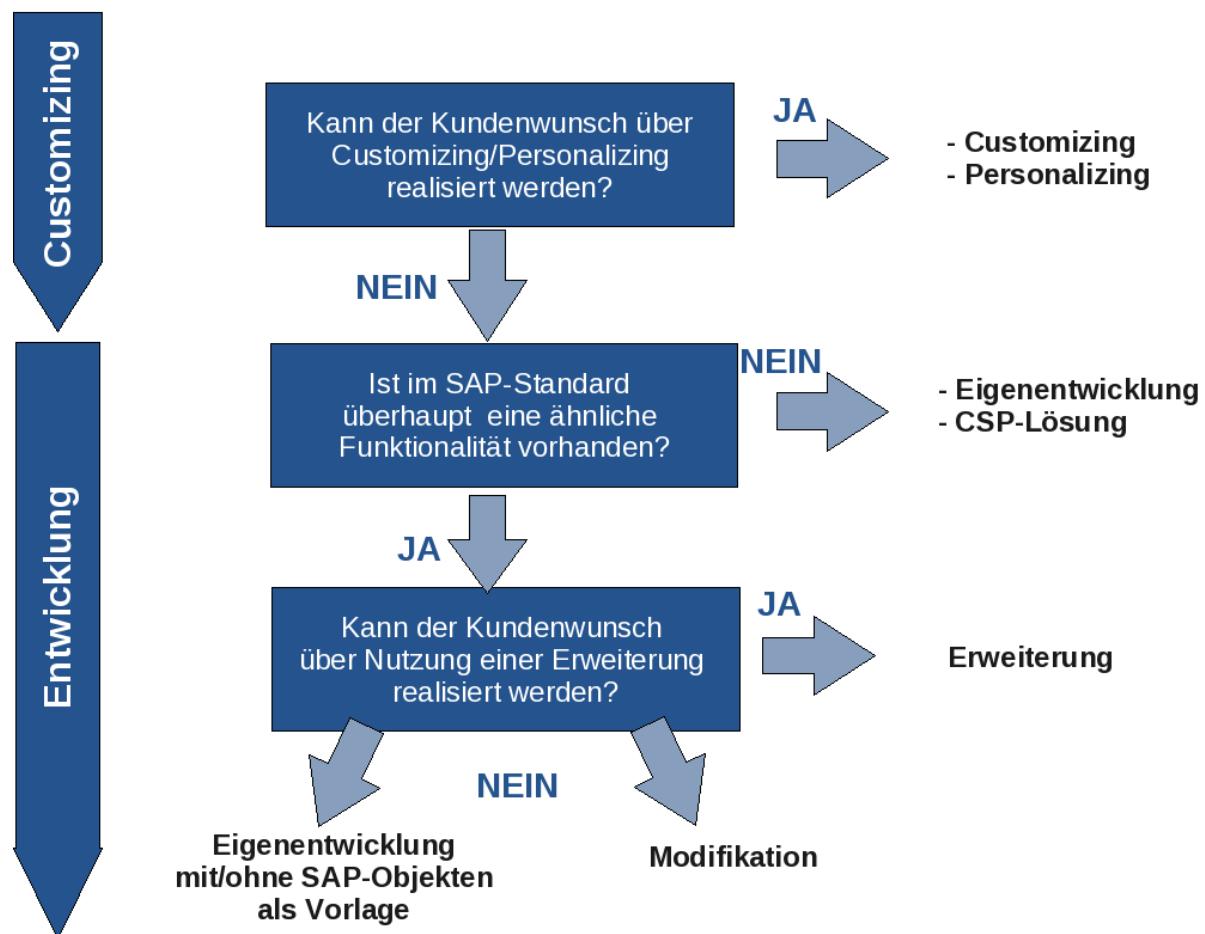


Abbildung 2: Entscheidungsbaum zur Auswahl einer Anpassungsmöglichkeit

Zunächst ist zu prüfen, ob durch Customizing oder Personalizing eine Anpassung möglich ist, die der Anforderung des Kunden genügt. Ist dies nicht der Fall, muss geprüft werden, ob eine der benötigten Funktionalität ähnliche Funktionalität im SAP-System zur Verfügung steht, die als Basis der Entwicklung verwendet werden kann. Ist eine solche Funktionalität nicht vorhanden, muss entweder ein Entwicklungsprojekt initiiert werden, um die Funktionalität selbst unabhängig von der SAP-Funktionalität zu entwickeln, oder auf eine CSP-Lösung zurückgegriffen werden. CSP steht für Complementary Software Product, es handelt sich dabei um von SAP zertifizierte Lösungen von Partnern.

Ist hingegen eine ähnliche Funktionalität gegeben, muss geprüft werden, ob die SAP-Softwarekomponente eine Erweiterungsmöglichkeit bietet, mit der die Funktionalität in die SAP-Softwarekomponente integriert werden kann. Hier gibt es eine Reihe von Techniken,

und nicht überall wird jede dieser Techniken angeboten oder ist sinnvoll nutzbar. Im Einzelnen werden Ihnen diese Techniken in diesem Kapitel des Kurses vorgestellt. Gibt es keine geeignete Technik, bleiben im Wesentlichen zwei Möglichkeiten: Entweder, es wird eine Eigenentwicklung gestartet, bei der evtl. die betroffenen SAP-Objekte als Vorlage benutzt werden können, oder es werden direkt SAP-Objekte durch eine Modifikation bearbeitet.

Für eine Modifikation wird ein spezieller Modifikationsschlüssel von SAP benötigt, da Entwicklungsarbeiten im SAP-Namensraum erforderlich werden. Modifikationen sind zwar sehr mächtig, haben aber den Nachteil, dass Komplikationen entstehen können, wenn SAP das System aktualisiert. Dabei müssen die geänderte Version von SAP und die modifizierte Version des Kunden zusammengeführt („abgeglichen“) werden. Aufgrund der hiermit verbundenen Problematik (insbesondere kann es zu hohem, manuellem Aufwand kommen) sollte eine Modifikation nur die letzte Wahl sein, wenn keine der anderen Möglichkeiten anwendbar ist bzw. sinnvoll erscheint.

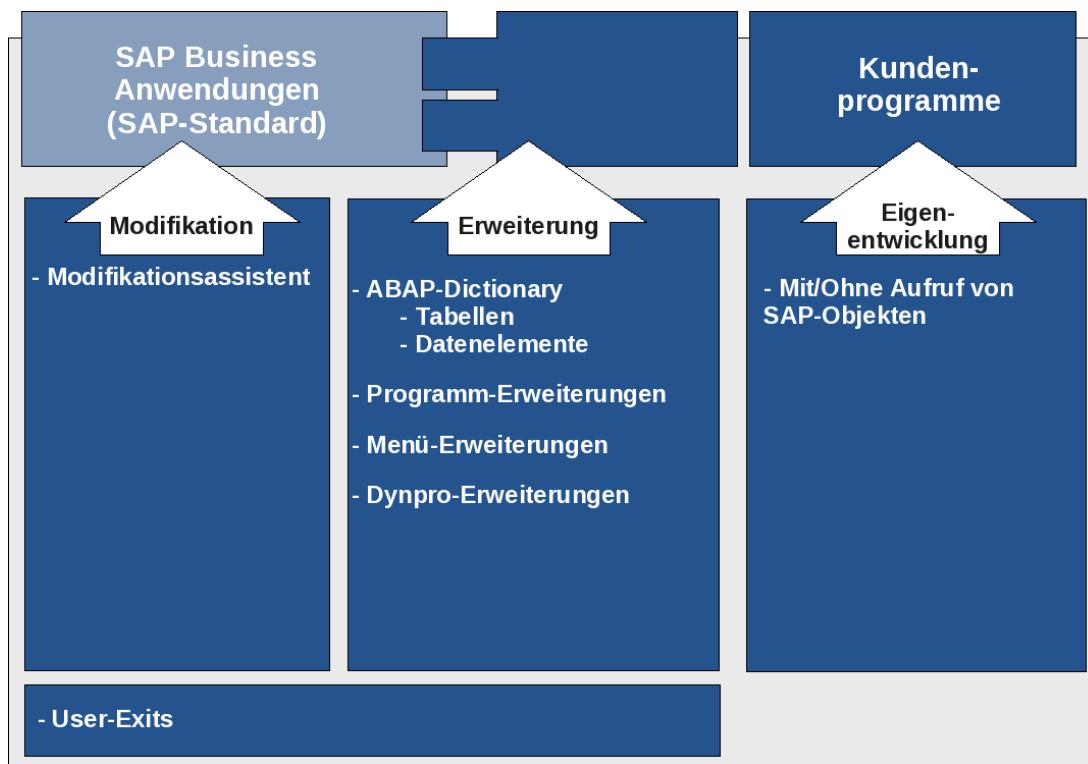


Abbildung 3: Änderungsebenen aus Sicht der ABAP Workbench

Bei Modifikationen sind User-Exits und „harte“ Änderungen zu unterscheiden. Letztere sind Anpassungen an beliebiger Stelle innerhalb der SAP-Objekte. User-Exits sind hingegen eine geführte Form der Modifikation: In Objekten des SAP-Namensraums befinden sich Unterprogramme, die explizit für Kundenentwicklungen vorgesehen sind. Aufgrund der oben angedeuteten Problematik bei Updates wird dazu geraten, Änderungen möglichst „modifikationsfrei“, also ohne die Verwendung von Modifikationen durchzuführen. User-Exits stellen hier einen Mittelweg dar, da sie zwar nicht modifikationsfrei sind, aber die Problematik des Abgleichs umgehen.

Bei Eigenentwicklungen ist es möglich, SAP-Repository-Objekte aufzurufen. Um so einen Fall handelt es sich etwa, wenn ein selbstentwickeltes Programm einen Funktionsbaustein der SAP aufruft. Bei Erweiterungen ist das Konzept hingegen umgekehrt: Hier sind in SAP-Repository-Objekten Aufrufe von Programmen aus dem Kundennamensraum vorgesehen. Der Kunde kann diese Programme in seinem Namensraum entwickeln und so die Funktionalität des SAP-Repository-Objekts beeinflussen, ohne dieses selbst zu bearbeiten.

Durch Erweiterungen kann nicht nur die Programmlogik angepasst werden. Auch Anpassungen an SAP-Dynpros, in die eigene Subscreen-Dynpros eingebunden werden, Anpassungen an Feldern (Feld-Exits), Menüs oder Dictionary-Elementen (Tabellenerweiterungen) sind möglich. Der Entwickler auf Kundenseite ist je nach verwendeter Erweiterungstechnik entweder darauf angewiesen, dass der SAP-Entwickler entsprechende Erweiterungsmöglichkeiten vorgedacht hat, oder kann auch ohne Vorbereitung eine Anpassung vornehmen. Es ist daher für jede Anforderung nach Anpassung zu prüfen, ob und wie diese umgesetzt werden kann.

Bei Erweiterungen von Tabellen gibt es etwa zwei Möglichkeiten: Append-Strukturen und Customizing Includes (CI-Includes). Erstere müssen nicht vom SAP-Entwickler vorgedacht werden, sondern stehen automatisch für jede Tabelle zur Verfügung, wohingegen Customizing Includes explizit vom SAP-Entwickler angelegt werden müssen. Der Vorteil der Customizing Includes liegt in ihrer Mehrfachverwendbarkeit: Während Append-Strukturen zu genau einer Tabelle gehören, kann eine Struktur durch Customizing Includes zur Erweiterung mehrerer Tabellen verwendet werden.

Für die Erweiterung der Programmlogik gibt es eine ganze Reihe von Techniken, die im Laufe der Zeit entstanden sind bzw. weiterentwickelt wurde. Die Grundidee ist bei diesen Techniken, aus dem SAP-Programm heraus ein Objekt (nicht im Sinne der Objektorientierung) des Kundennamensraums aufzurufen. Dieses kann dann vom Kunden bearbeitet werden.

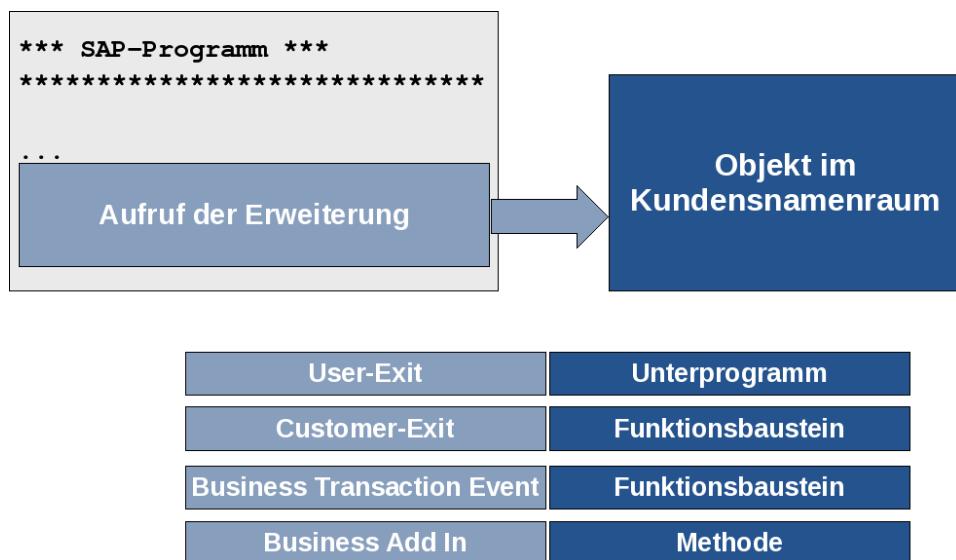
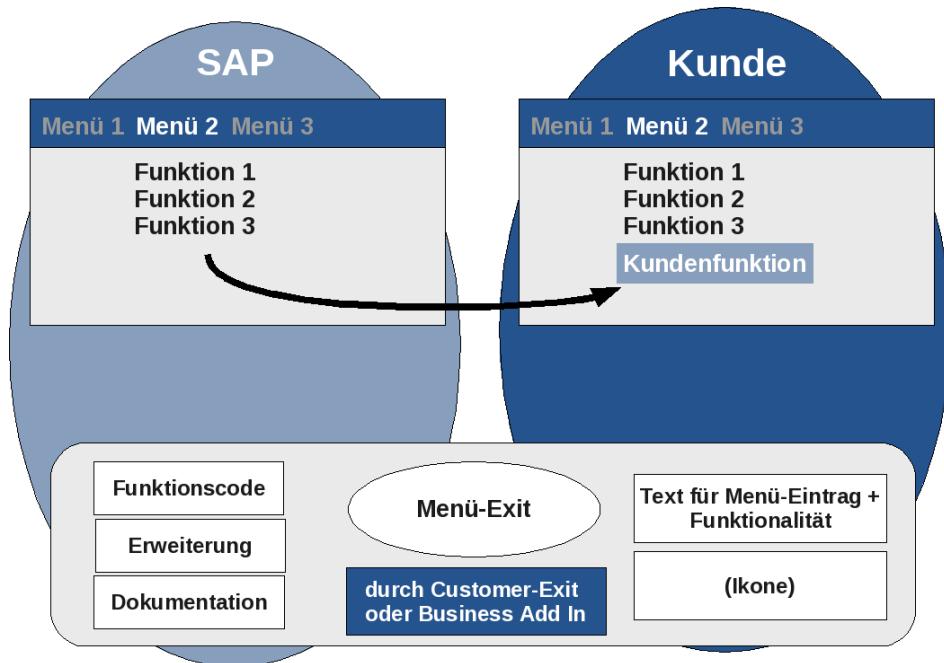


Abbildung 4: Funktionsweise von Programm-Erweiterungen

Je nach verwendeter Technik handelt es sich bei dem gerufenen Objekt des Kundennamensraums um ein Unterprogramm, einen Funktionsbaustein oder eine Methode. Die User-Exits nehmen eine Sonderrolle ein, da technisch gesehen nicht im Kundennamensraum entwickelt wird. Dies wird im Unterkapitel zu Modifikationen noch genauer erläutert. Business Transaction Events sind eine für das FI-Modul von SAP entwickelte Erweiterungstechnik, bei der ein Funktionsbaustein im Kundennamensraum dynamisch aufgerufen wird. Es sind keine Dynpro- oder Menüberweiterungen mit BTEs möglich und sie verfügen über keine übergeordnete Verwaltungsebene. Sie bieten allerdings eine Filterabhängigkeit sowie Mehrfachverwendbarkeit. Diese Technik wird in diesem Kurs nicht näher behandelt.

In einem späteren Unterkapitel lernen Sie zusätzlich zu diesen klassischen Techniken zur Programmerweiterung das neue Erweiterungskonzept der SAP kennen. Dieses erlaubt mit impliziten Enhancement Points auch Erweiterungen, die nicht von SAP vorgedacht wurden.



Zur Erweiterung von Menüs reserviert der SAP-Entwickler zusätzliche Funktionscodes im Menü. Der Kunde kann diesen dann einen Text und eine Funktionalität (und je nach Vorgabe durch SAP auch eine Ikone) zuweisen. Zur Realisierung von Menü-Exits kann sowohl die Customer-Exit- als auch die Business Add In-Technik verwendet werden.

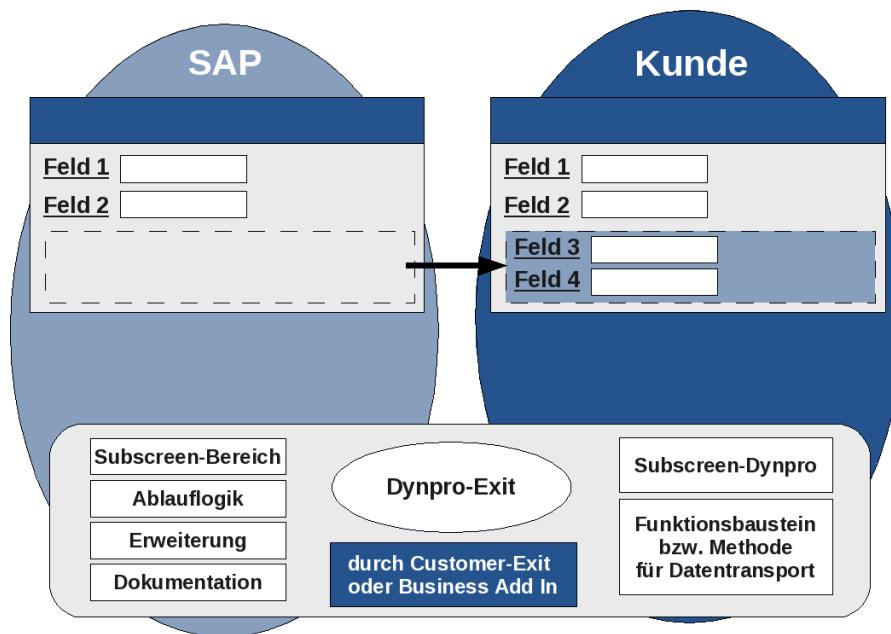


Abbildung 6: Dynpro-Exits

Bei Dynpro-Exits kommt ebenfalls die Customer-Exit-Technik zum Einsatz, alternativ kann seit Release 6.20 auch auf Business Add Ins zurückgegriffen werden. Bei Dynpro-Exits werden Subscreen-Dynpros des Kunden in ein Dynpro von SAP eingebunden. Dafür sieht der SAP-Entwickler entsprechende Subscreen-Bereiche sowie die Aufrufe der zugehörigen PBO- bzw. PAI-Module vor. Die besondere Herausforderung besteht darin, den Datentransport zu organisieren, da die Daten des SAP-Programms nicht automatisch im Subscreen-Dynpro zur Verfügung stehen. Einen entsprechenden Rahmen für den Transport stellt der SAP-Entwickler bereit. Wie der Transport dann genau funktioniert, wird im entsprechenden Teil dieses Kapitels erläutert.

Kommt für die benötigten Anpassungen keine der angebotenen Erweiterungstechniken in Frage, bleibt die Möglichkeit der Modifikation. Hierbei werden Objekte des SAP-Namensraums vom Kunden direkt bearbeitet.

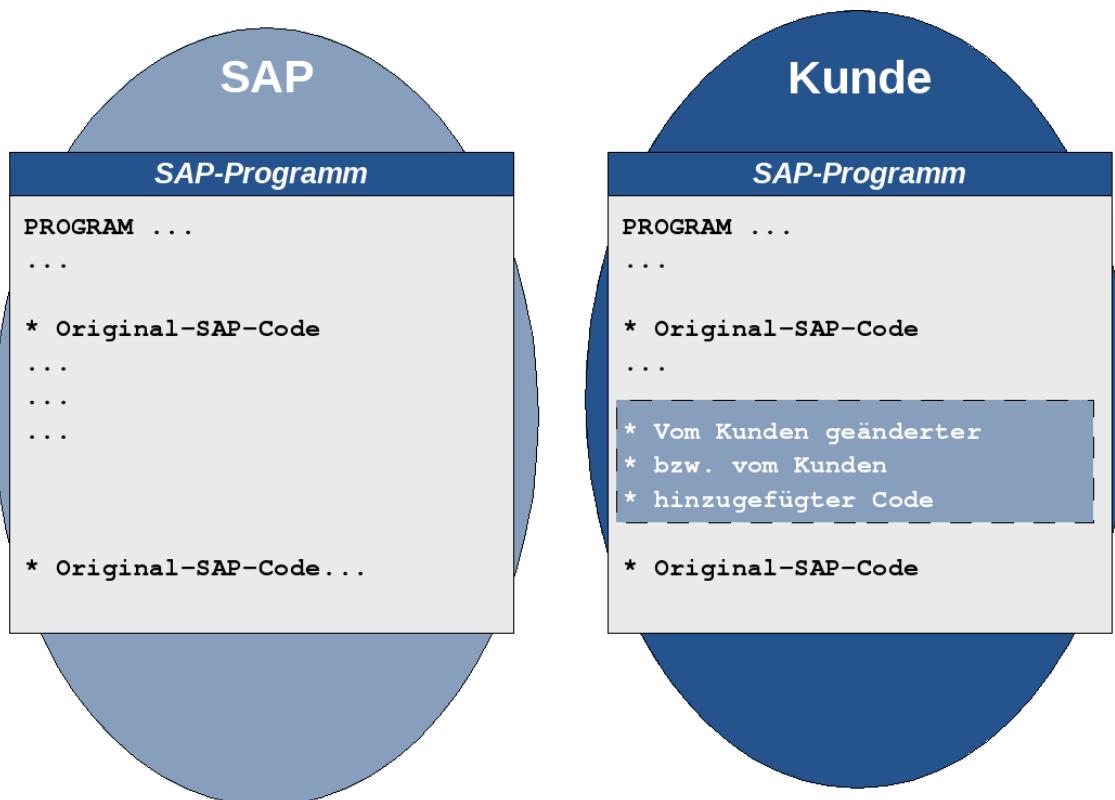


Abbildung 7: Modifikationen

Bei Auslieferung einer neuen Version des Programms durch SAP ist jedoch ein Abgleich der Änderungen erforderlich. Im Zuge der Einführung des Modifikationsassistenten wurde hier für Erleichterung gesorgt: Die feinste Granularität zur Aufzeichnung von Änderungen ist seit dessen Einführung (Release 4.5A) nicht mehr das jeweilige Repository-Objekt (z. B. ein Include-Programm), sondern einzelne Bestandteile des Programms. So kann festgestellt werden, ob die Änderungen am SAP-Programm die Bereiche, in denen der Kunde Modifikationen vorgenommen hat, überhaupt betreffen. Im Unterkapitel zu Modifikationen wird dies näher erläutert.

## 6.1 Erweiterungen an Dictionary-Elementen

Es gibt zwei Möglichkeiten, Dictionary-Elemente zu erweitern. Zum einen können die Texte, die als Feldbezeichner oder zur Dokumentation von Datenelementen verwendet werden, durch kundeneigene Texte verändert werden, zum anderen können Tabellen um kundeneigene Felder erweitert werden.

### 6.1.1 Texterweiterungen

Die kundeneigenen Texte zur Erweiterung der Dokumentation von SAP-Datenelementen und die Überlagerung von SAP-Feldbezeichnern werden unter dem Begriff **Texterweiterungen** zusammengefasst.

Die Erweiterung der Dokumentation wirkt sich auf die F1-Hilfe aus. Hier kann entweder ein vollständig eigener Text hinterlegt werden, oder der SAP-Text um eigene Angaben ergänzt werden.

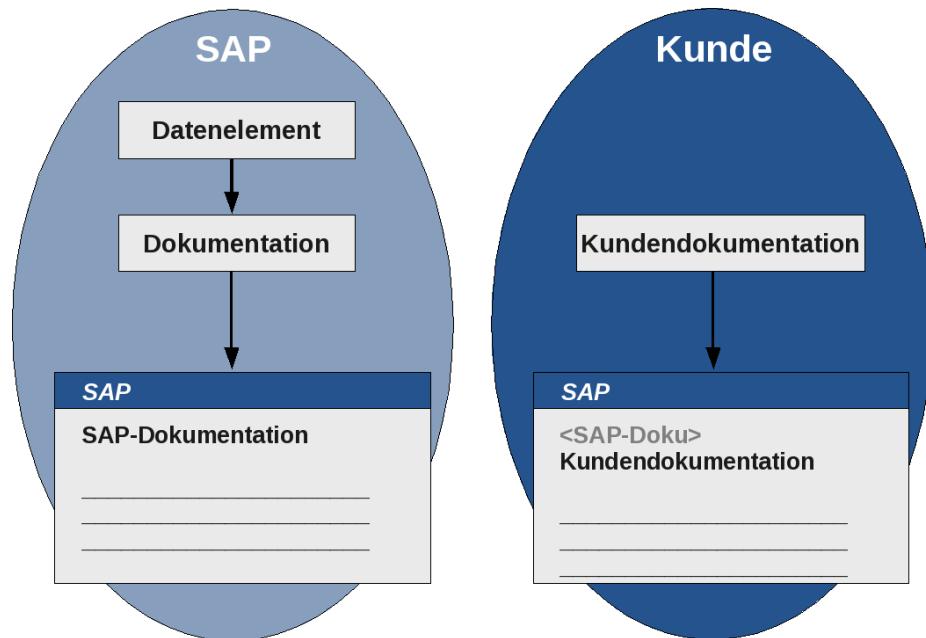


Abbildung 8: Erweiterung der Dokumentation eines SAP-Datenelements

Um eine solche Modifikation durchzuführen ist in der Transaktion **CMOD** der Menüpfad **Springen -> Glob. Erweiterungen -> Datenelemente -> DA-Kundendoku neu** zu wählen. Glob. Erweiterungen steht für globale Erweiterungen. Das bedeutet, dass eine solche Erweiterung in allen Programmen wirksam wird, die das betroffene Datenelement verwenden. Die Auswirkungen können bei zentralen Elementen also weitreichend sein.

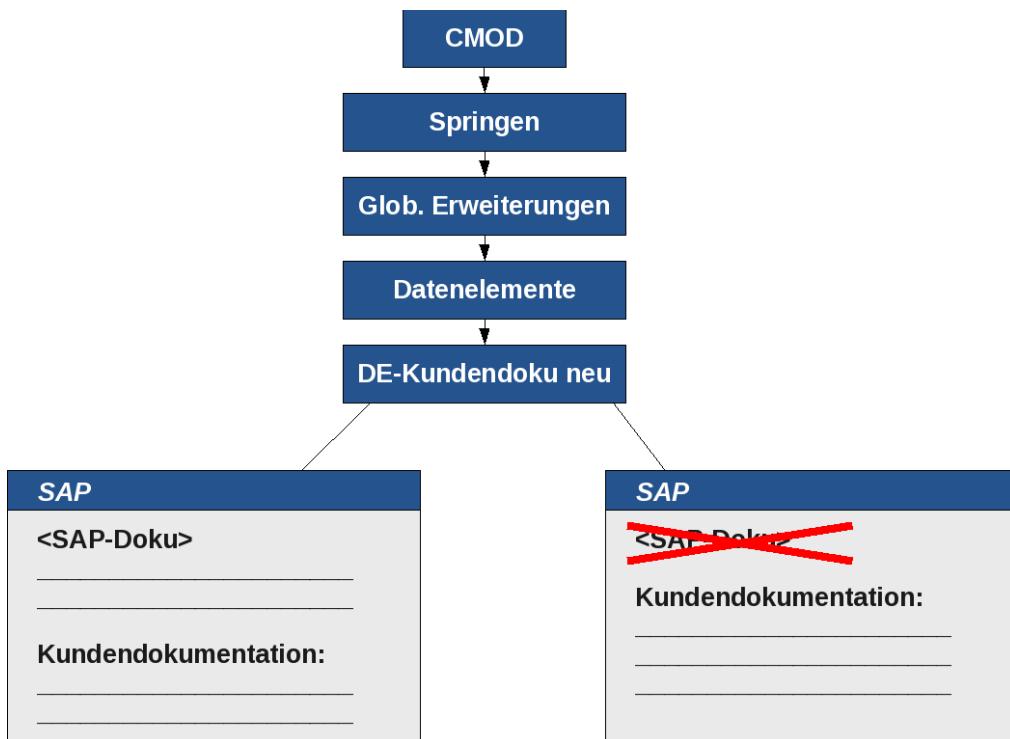


Abbildung 9: Ersetzen oder Ergänzen der Dokumentation?

Beim beschriebenen Menüpfad kann auf Basis einer Abfrage des Systems gewählt werden, ob der Originaltext ergänzt oder ein leeres Template zugrundegelegt werden soll.

Bei der Überlagerung von Feldbezeichnern können die im Dictionary gepflegten kurzen, mittleren, langen und Überschriften-Bezeichner angepasst werden.

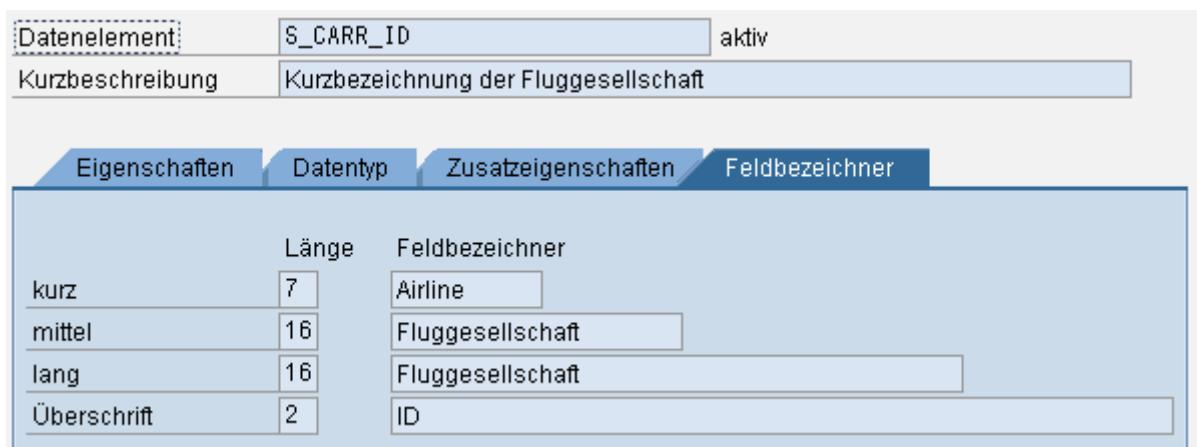


Abbildung 10: Pflege von Feldbezeichnern im Dictionary: SAP-System-Screenshot

Die entsprechende Erweiterung wird auch über die Transaktion **CMOD** vorgenommen. Der Menüpfad lautet dort dann **Springen -> Glob. Erweiterungen -> Schlüsselworte -> ändern**. Die Bezeichnung im Menü ist etwas unglücklich gewählt, da der Originaltext nicht verloren geht, sondern nur überlagert wird.

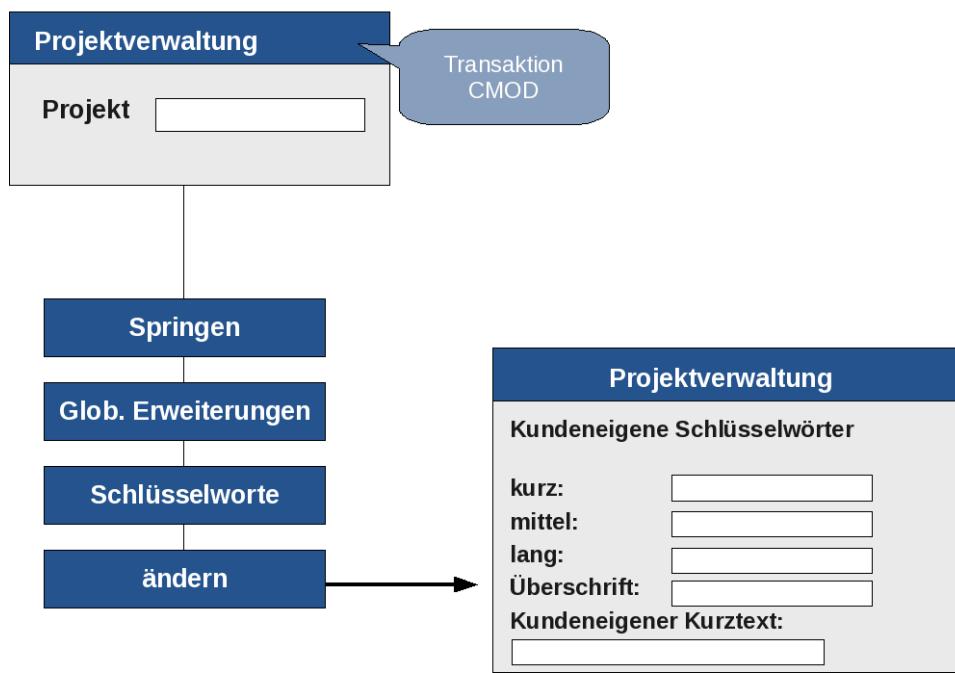


Abbildung 11: SAP-Feldbezeichner Überlagern

Da es sich auch hier um eine globale Erweiterung handelt, werden die Bezeichner in allen Programmen wirksam, die das Element verwenden. Eine Ausnahme stellen jedoch Programme dar, die den Feldbezeichner nicht aus dem Dictionary beziehen. Dies wird im Attribut „Dict modifiziert“ eines Dynpro-Feldes spezifiziert. Dieses kann entweder die Werte

1-4 für den kurzen, mittleren, langen bzw. Überschriften-Bezeichner, den Wert V oder Leerzeichen für eine variable Auswahl in Abhängigkeit von der verfügbaren Länge, oder den Wert F für einen fixen Wert haben. Letzterer sorgt dafür dass das Dictionary umgangen wird, so dass diese Stelle des Dynpros von der Erweiterung nicht berührt wird.

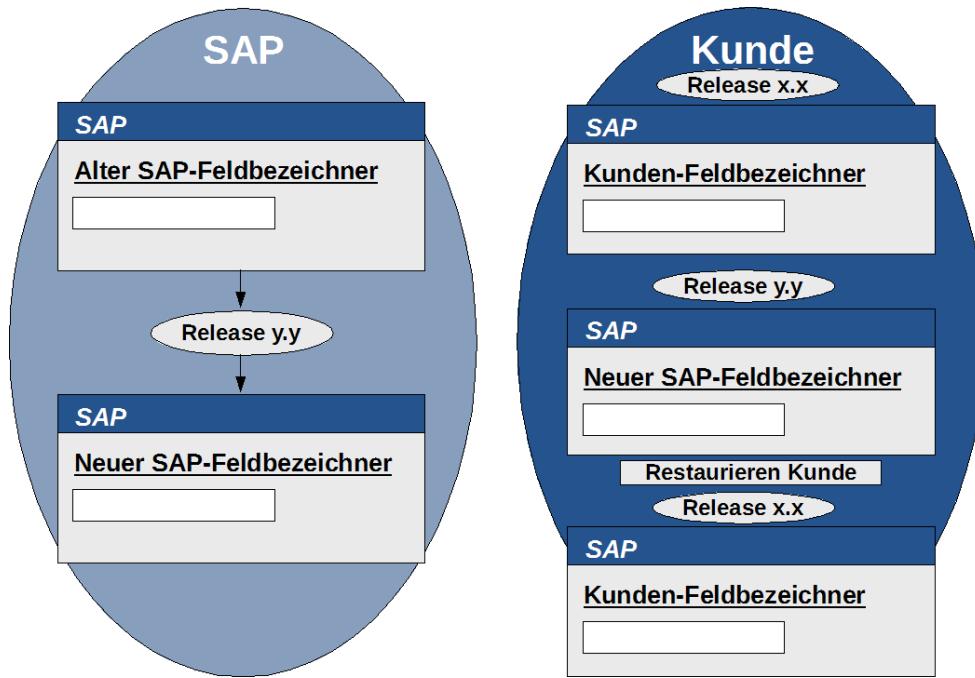


Abbildung 12: Überlagerte Feldbezeichner bei Releasewechsel

Beim Wechsel des SAP-Releases kann es vorkommen, dass SAP vorhandene Bezeichner noch einmal ausliefert. Der Kunde kann dann seine selbstdefinierten Bezeichner aus dem alten Release restaurieren. Dies wird durch ein Programm im Hintergrund durchgeführt. Die Durchführung dieser Feldbezeichnerfortschreibung wird von SAP empfohlen. Vorsicht ist allerdings bei der Restaurierung von Feldbezeichnern von Feldern geboten, die an sehr vielen Stellen im System verwendet werden, etwa die Mandantennummer oder der Buchungskreis. SAP empfiehlt, die Restaurierung nicht parallel zu schreibenden Zugriffen auf Datenbanktabellen durchzuführen, die diese Datenelemente verwenden, da die Aktivierung sonst scheitern kann und die Elemente dann nur teilweise aktiviert sind.

### 6.1.2 Erweiterungen an Strukturen und Tabellen

Zur Erweiterung von Strukturen und Datenbanktabellen gibt es im SAP-System zwei Möglichkeiten: Append-Strukturen und Customizing Includes.

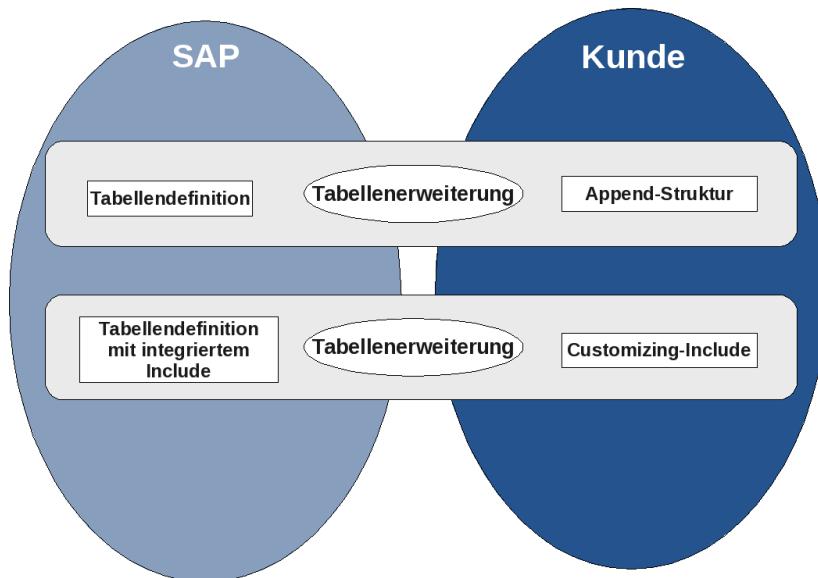


Abbildung 13: Tabellenerweiterungen: Append-Strukturen und Customizing Includes

**Customizing-Includes** sind von SAP vorgesehene Erweiterungen, die in die entsprechenden SAP-Tabellen integriert sind, und vom Kunden mit Feldern gefüllt werden. **Append-Strukturen** werden hingegen vom Kunden zu einer beliebigen Tabelle angelegt und erfordern keine Vorarbeit seitens SAP.

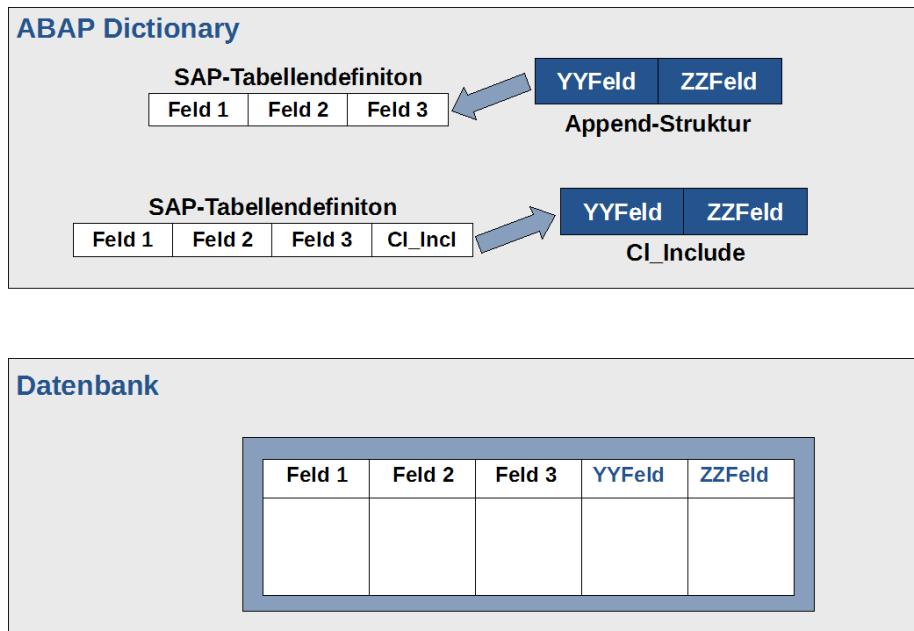


Abbildung 14: Append-Struktur vs. Customizing Include

Ein weiterer Unterschied zwischen Append-Strukturen und Customizing Includes liegt in der Richtung der Referenzierung. Ein Append wird genau einer Tabelle zugeordnet, und beim Aktivieren werden die Felder der Tabelle hinzugefügt. Es können mehrere Apps zu einer Tabelle definiert werden. Bei Includes geht die Beziehung von der Tabelle aus: Diese enthält eine zusätzliche „INCLUDE“-Zeile die das Include angibt. Der Unterschied ist in obiger Abbildung veranschaulicht.

Customizing-Includes sind in SAP-Tabellen an Stellen vorhanden, an denen von vornherein kundenspezifische Felder vorgesehen sind. Mit der umgekehrten Richtung der Beziehung ist auch eine Mehrfachverwendung verbunden: Ein Customizing-Include kann in mehreren Tabellen oder Strukturen eingebunden sein. So können kundenspezifische Felder, die sich auf mehrere Tabellen oder Strukturen beziehen, zentral gewartet werden und so die Konsistenz gewährleistet werden. Zum Anlegen werden spezielle Customizing-Transaktionen verwendet. Einige Customizing Includes sind auch Bestandteil von SAP-Erweiterungen und werden in der Projektverwaltung für Erweiterungsprojekte angelegt (zu Erweiterungsprojekten siehe nächstes Unterkapitel).

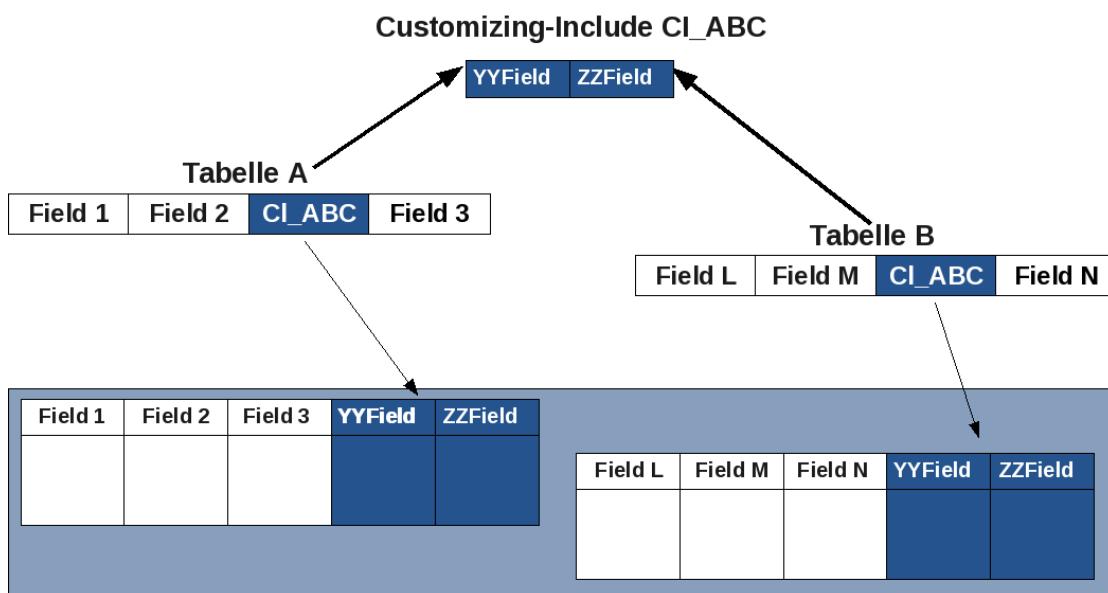


Abbildung 15: Customizing-Includes

Der Name eines Customizing-Includes liegt im Kundennamensraum, beginnt aber stets mit **CI\_**. Über diese Namenskonvention wird sichergestellt, dass kein Fehler entsteht, weil ein Customizing-Include nicht existiert. Die Felder des Includes müssen den gewöhnlichen Namensrestriktionen für den Kundennamensraum genügen (sie beginnen mit YY oder ZZ). So wird sichergestellt, dass es keine Namenskonflikte zwischen SAP-Feldern und vom Kunden durch das Include definierte Felder gibt.

Genau wie bei Append-Strukturen werden die Felder von Customizing-Includes von Tabellen bei Aktivierung auf der Datenbank angelegt bzw. bei Änderungen bearbeitet. Ein Hinzufügen von Feldern erfordert keine Umsetzung der Datenbanktabelle, da die Feldreihenfolge zwischen Dictionary und Datenbank seit Release 3.0 abweichen kann. So sehen Sie etwa in der obigen Abbildung, dass die Tabellenstrukturen im Dictionary von der im unteren Bereich der Abbildung dargestellten Repräsentation in der Datenbank bezüglich der Reihenfolge der Felder abweichen.

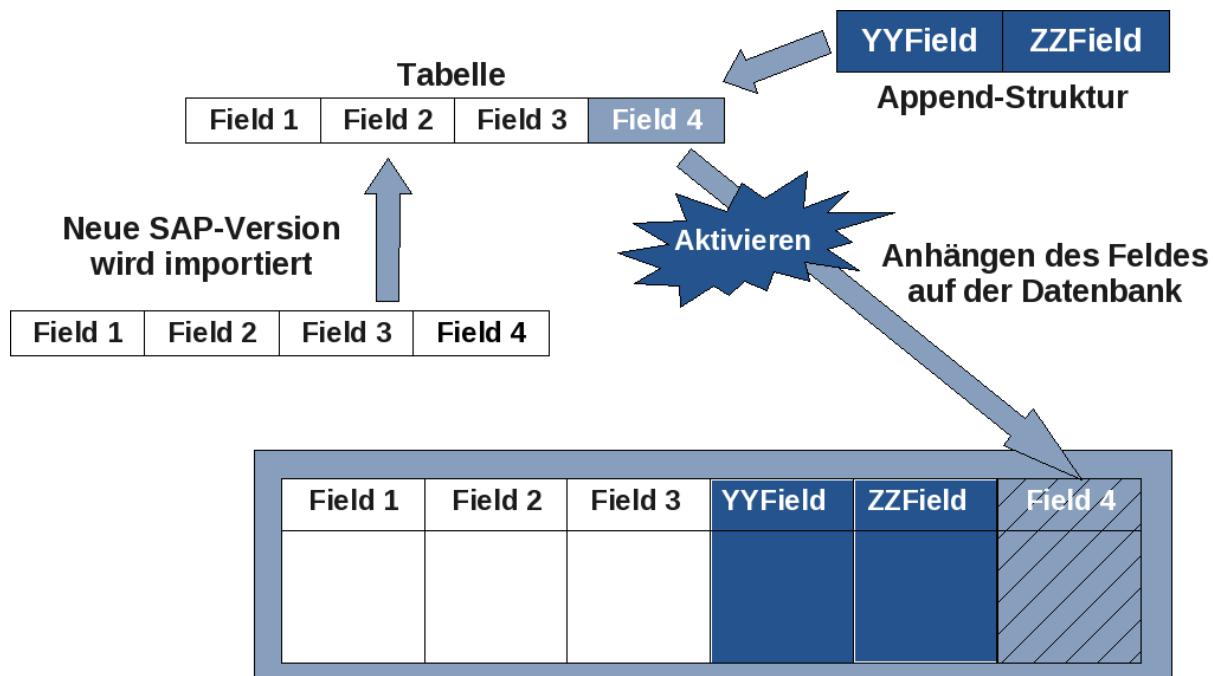


Abbildung 16: Neue Felder und Append-Strukturen

Die Abbildung zeigt das Ihnen bereits bekannte Verhalten bei Upgrades in Zusammenhang mit Appends. Die SAP-Tabelle enthält zunächst die Felder Field 1, Field 2 und Field 3. Der Kunde hat diese über die Append-Struktur um die Felder YYField und ZZField erweitert. Mit der neuen Version der SAP kommt ein neues Feld Field 4 hinzu. Beim Aktivieren der Tabelle wird dieses Feld im Datenbanksystem an das Ende der Tabelle angehängt. So wird ein Umsetzen der Tabelle vermieden.

Append-Strukturen kennen Sie bereits aus dem Kurs „Einführung in ABAP“ und haben diese dort auch praktisch angewendet. Diese werden daher hier nicht erneut detailliert behandelt.

## 6.2 Customer-Exits

Customer-Exits sind von SAP in SAP-eigenen Programmen vorgesehene Erweiterungsmöglichkeiten. Diese umfassen sowohl Programmcodeerweiterungen als auch Anpassungen der Oberfläche durch die Veränderung von Menüs und Dynpros.

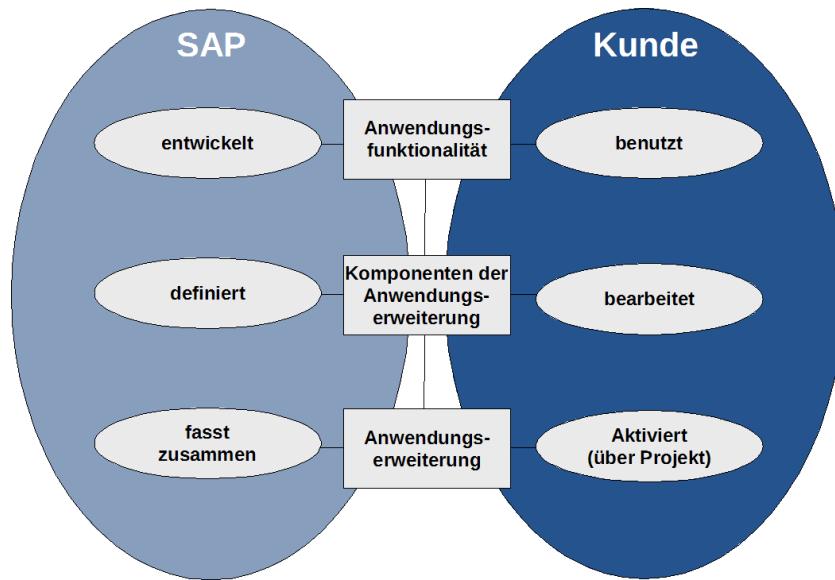


Abbildung 17: SAP-Anwendungserweiterungen

Customer-Exits sind Komponenten einer **Anwendungserweiterung**. Die Anwendungserweiterungen werden **von SAP zusammengesetzt** und können vom Kunden in seinem System über ein Erweiterungsprojekt (dazu unten mehr) aktiviert (und bei Bedarf auch wieder deaktiviert) werden. Der Kunde bearbeitet vor der Aktivierung die einzelnen Komponenten und fügt so die von Ihm benötigte Funktionalität ein. Da die Erweiterungen von SAP vorgedacht sind bleiben Sie auch bei Upgrades erhalten und erfordern keine besonderen Aktivitäten. Dem Kunden steht eine genau definierte Schnittstelle zur Erweiterung zur Verfügung. Im Gegensatz etwa zu einer manuellen Modifikation muss er sich so nicht mit Details der SAP-Seitigen Implementierung auseinandersetzen.

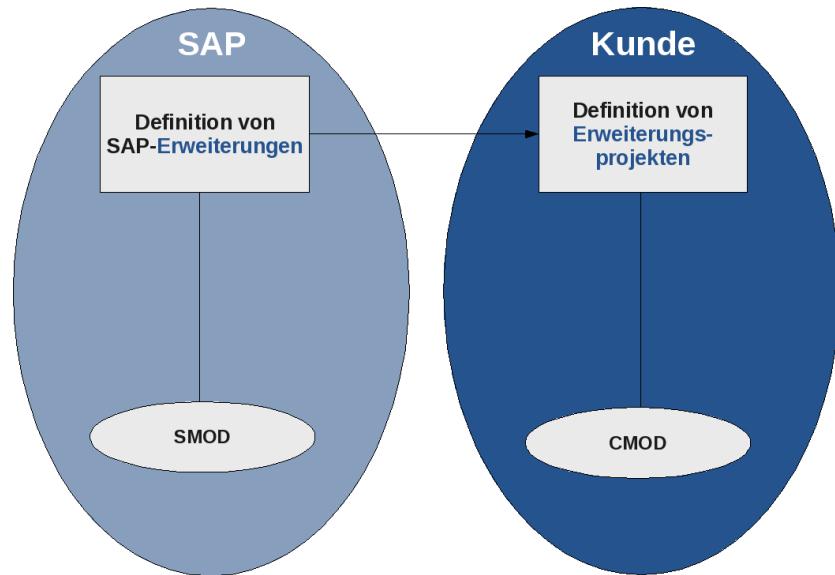


Abbildung 18: Erweiterungen und Erweiterungsprojekte

Zur Definition einer Anwendungserweiterung verwendet der SAP-Anwendungsentwickler die Transaktion **SMOD**. Dort stellt er die Programm-, Menü- und Dynpro-Exits zusammen. Der Kunde informiert sich über die angebotenen Erweiterungen und verwendet die Transaktion **CMOD**, um die Erweiterungen zu einem **Erweiterungsprojekt** zusammen zu fassen.

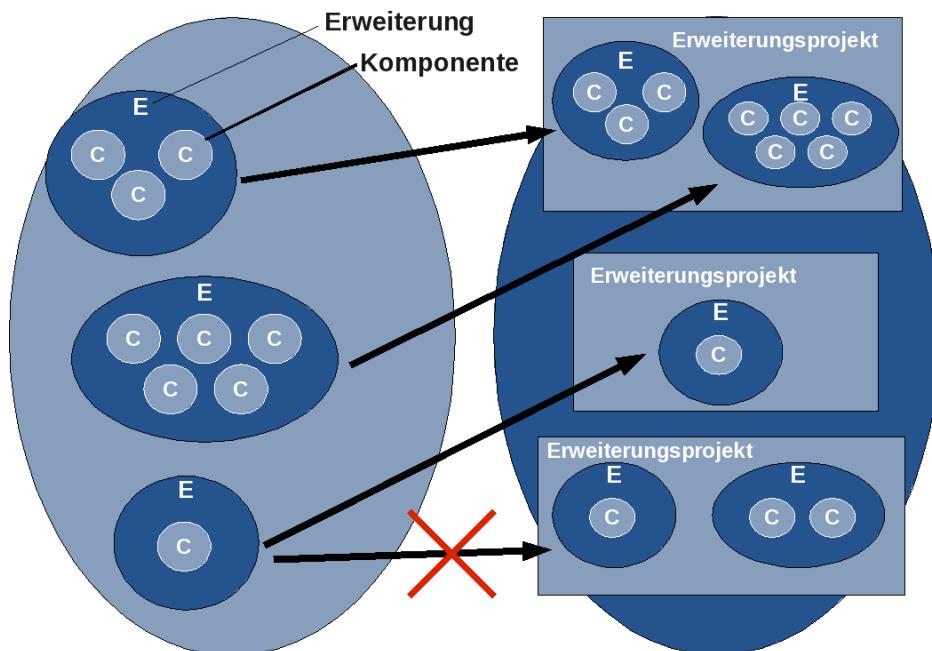


Abbildung 19: Aufbau von Erweiterungen und Erweiterungsprojekten

Die obige Abbildung zeigt den Aufbau von Erweiterungen und Erweiterungsprojekten: Der Kunde setzt seine Erweiterungsprojekte aus Erweiterungen zusammen, die der SAP-Entwickler definiert hat. Die Erweiterungen setzen sich aus den einzelnen Erweiterungskomponenten (Programm-, Menü- und Dynpro-Exits) zusammen. Dabei muss sowohl die Zuordnung von Erweiterungen zu Erweiterungsprojekten als auch die Zuordnung von Komponenten zu Erweiterungen stets eindeutig sein: Eine Komponente darf nie in mehreren Erweiterungen auftreten, und eine Erweiterung darf vom Kunden nie in mehreren Erweiterungsprojekten verwendet werden. Man spricht auch von der Eindeutigkeit einer SAP-Erweiterung bzw. der Eindeutigkeit eines Kundenprojekts.

### 6.2.1 Vorgehensweise für Erweiterungsprojekte

Die Komponenten für Erweiterungen werden vom SAP-Anwendungsentwickler bei der Entwicklung eingeplant und die benötigten Komponenten in den Anwendungsfunktionen definiert. Anschließend fasst er die Komponenten zu einer SAP-Erweiterung zusammen, und erstellt schließlich eine Dokumentation über die Erweiterung.

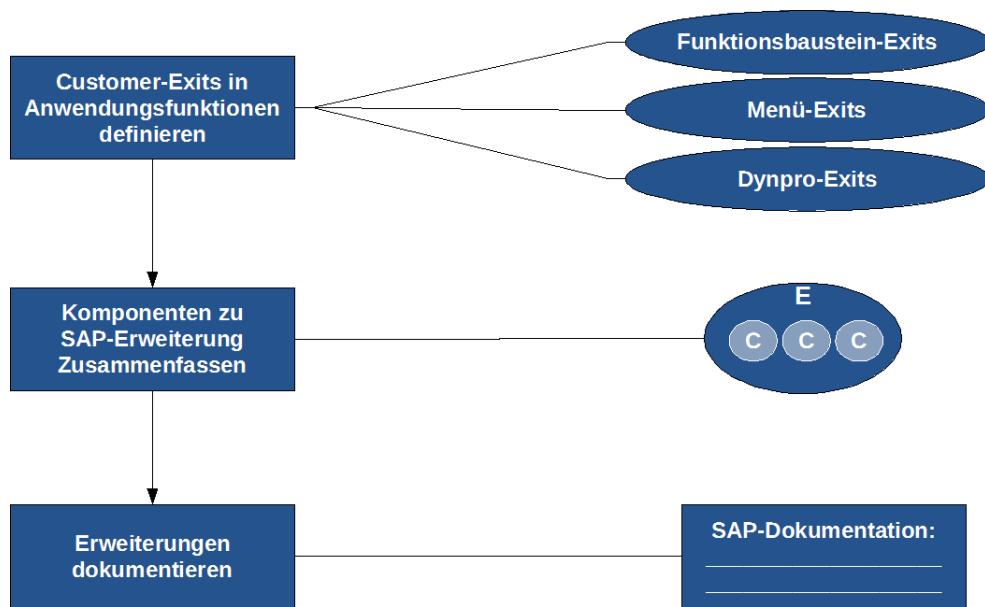


Abbildung 20: Vorgehensweise des SAP-Anwendungsentwicklers

Diese Dokumentation steht dem Entwickler auf Kundenseite zur Verfügung. Er kann so ermitteln, ob eine Erweiterung für die von ihm gewünschte Funktionalität geeignet ist oder nicht. Aus den Erweiterungen des SAP-Entwicklers baut nun der Kunde sein Erweiterungsprojekt zusammen:

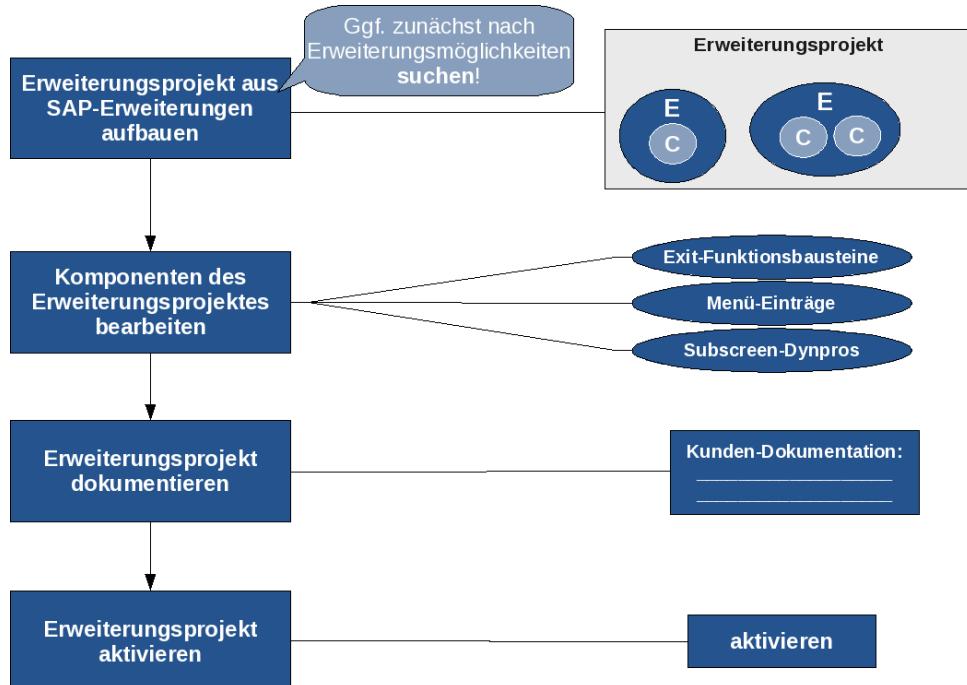


Abbildung 21: Vorgehen auf Kundenseite

Das Erweiterungsprojekt wird vom Kunden über die Transaktion **CMOD** angelegt:



Abbildung 22: Anlegen eines Erweiterungsprojekts: SAP-System-Screenshot

Dort ist zunächst als Attribut ein Kurztext zu pflegen. Um ein Erweiterungsprojekt aufzubauen zu können, muss der Kunde wissen, welche Erweiterungen bzw. Erweiterungskomponenten überhaupt von SAP im jeweiligen Programm angeboten werden. Dazu kann aus der Transaktion **CMOD** unter **Teilobjekte** der Punkt **Zuordnung von Erweiterungen** gewählt werden (siehe auch obige Abbildung). Er führt zu einer Liste der Erweiterungen im Erweiterungsprojekt. Durch die F4-Hilfe kann eine Suchmaske geöffnet werden, mit der global nach Erweiterungen gesucht werden kann:

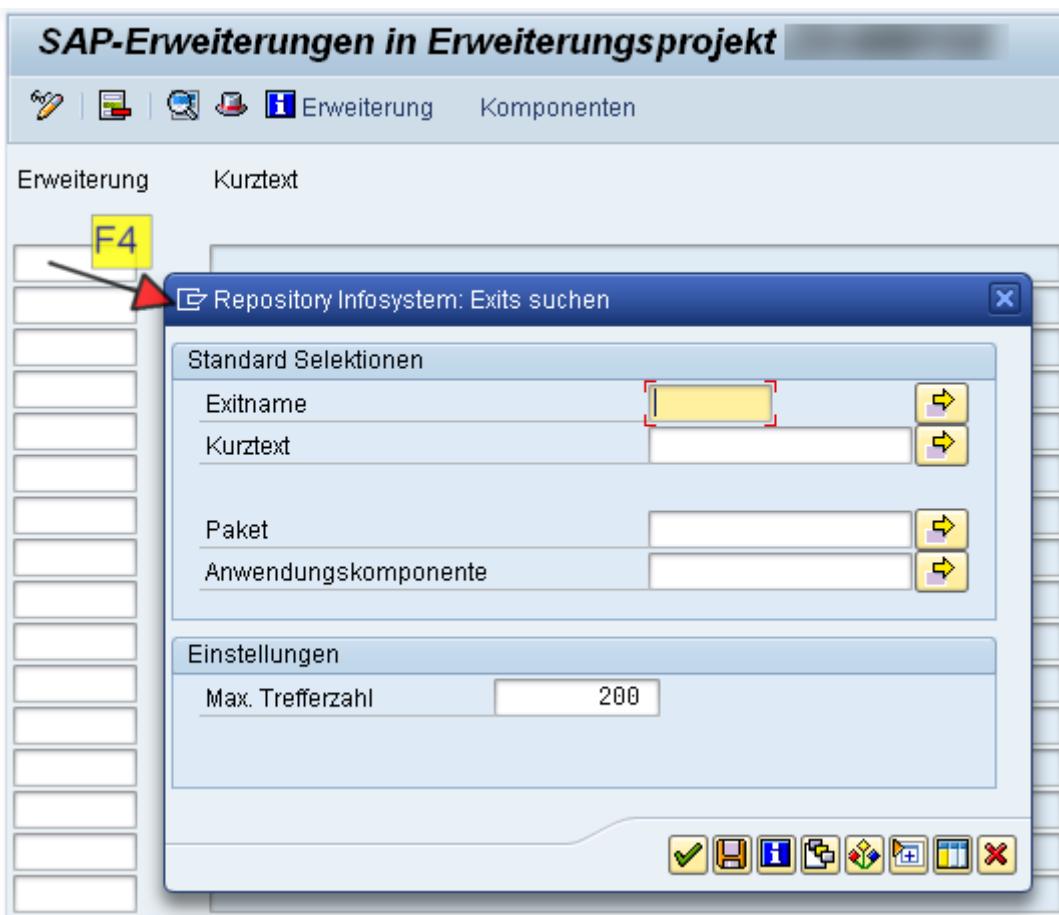


Abbildung 23: Suche nach Erweiterungen: SAP-System-Screenshot

Alternative Möglichkeiten zur Suche nach Erweiterungsmöglichkeiten werden Ihnen später noch vorgestellt. Hat der Kunde eine geeignete Erweiterung gefunden, kann er anschließend die Komponenten bearbeiten. Dafür kann er auf dem Einstiegsbild der Transaktion **Komponenten** auswählen (siehe Abbildung 22). Je nachdem ob es sich um die Erweiterung eines Menüs, eines Funktionsbausteins oder eines Dynpros handelt, wird zum Bearbeiten das entsprechende Werkzeug der ABAP Workbench geöffnet, also der Menu Painter, Function Builder oder der Screen Painter verwendet.

Über den Auswahlpunkt **Dokumentation** (siehe Abbildung 22) pflegt der Kunde eine Dokumentation seines Erweiterungsprojekts.

Während der Entwicklung der einzelnen Komponenten durch den Kunden wird das System noch nicht beeinflusst. Erst durch das **aktivieren** eines Erweiterungsprojekts werden die Änderungen wirksam. Das System generiert die betroffenen Dynpros und Menüs neu, und beim Start von Programmen auch diese.

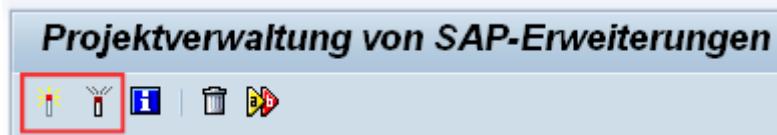


Abbildung 24: Aktivierung / Deaktivierung eines Projekts: SAP-System-Screenshot

Die Aktivierung des Projekts kann auch wieder zurückgenommen werden. Dafür steht eine entsprechende Schaltfläche auf dem Einstiegsbild der Transaktion CMOD bereit (siehe Abbildung). Ein **inaktives** Erweiterungsprojekt ist weiterhin im System vorhanden und kann bei Bedarf zu einem späteren Zeitpunkt wieder aktiviert werden.

Erweiterungsprojekte sind transportierbare Repository-Objekte, sie können also mithilfe eines Transportauftrags mitsamt ihren Komponenten vom Entwicklungssystem z. B. auf ein Testsystem und weiter auf ein Produktivsystem übertragen werden.

Nachdem Ihnen nun die Organisation von Erweiterungen in Erweiterungsprojekten bekannt ist, werden Ihnen im Folgenden die über Customer-Exits realisierten Programm-Exits, Menü-Exits und Dynpro-Exits vorgestellt.

## 6.2.2 Programm-Exits

Programm-Exits bieten die Möglichkeit, die Anwendungslogik eines SAP-Programms zu erweitern. Sie werden über Funktionsbausteine und Includes realisiert. Der SAP-Anwendungsentwickler legt die Programm-Exits an und stellt dem Kunden so eine genau definierte Schnittstelle zur Erweiterung bereit. Diese Schnittstelle wird in einer Dokumentation beschrieben. Im SAP-Programm fügt er Aufrufe der Exits ein, wodurch festgelegt wird, an welcher Stelle die Logik des Kunden zum Tragen kommt.

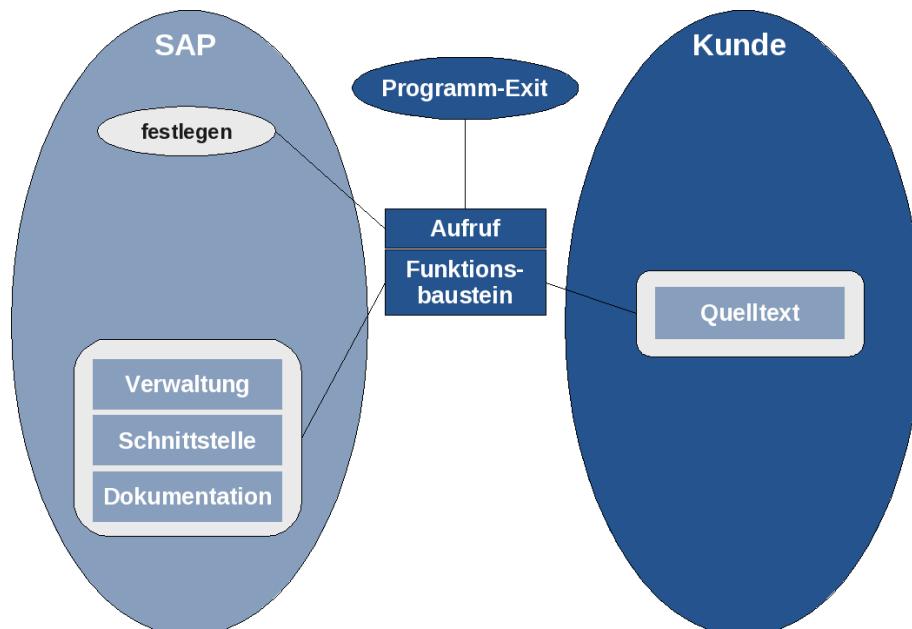


Abbildung 25: Programm-Exits

Die folgende Abbildung zeigt in Anlehnung an ein UML-Sequenzdiagramm die Architektur eines Programm-Exits:

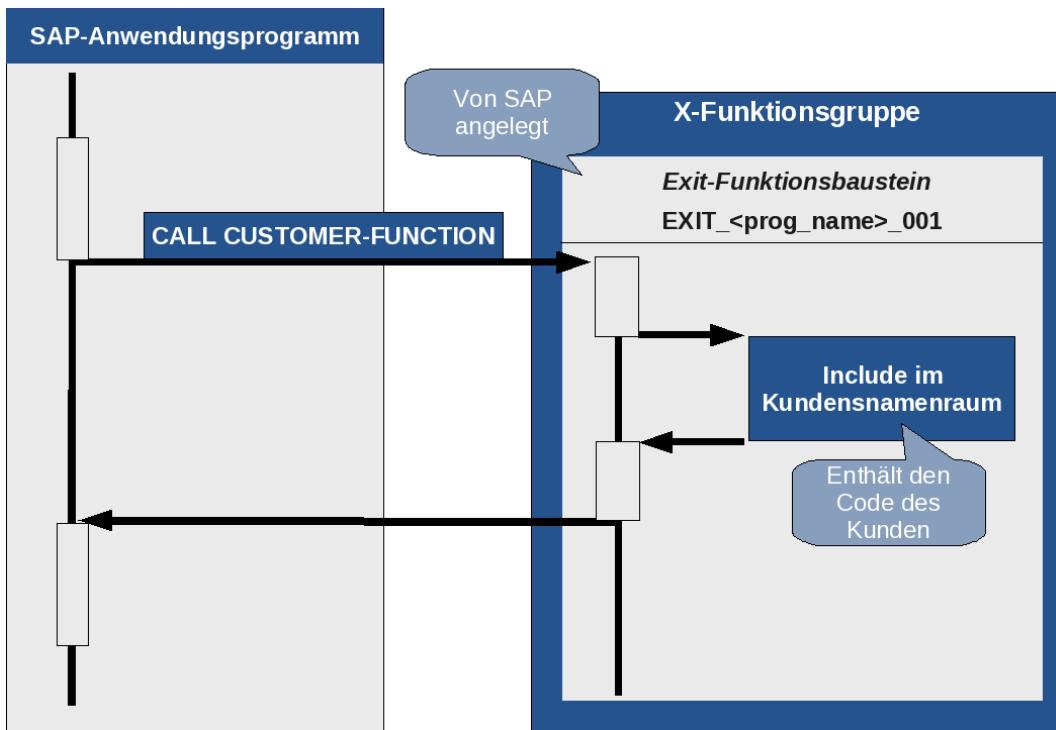


Abbildung 26: Architektur eines Programm-Exits

Das SAP-Anwendungsprogramm enthält den Befehl CALL CUSTOMER-FUNCTION, der analog zum CALL FUNCTION-Befehl einen Funktionsbaustein aufruft. Hier handelt es sich um einen speziellen, **von SAP** angelegten Exit-Funktionsbaustein aus einer X-Funktionsgruppe. Innerhalb des Funktionsbausteins wird ein Include eingebunden. Dieses befindet sich im Kundennamensraum und kann so **vom Kunden** angelegt, bearbeitet und mit der durch den Kunden gewünschten Programmlogik gefüllt werden. Diese wird dann an jeder Stelle ausgeführt, an der der Funktionsbaustein gerufen wird, sofern das Erweiterungsprojekt des Kunden, zu dem die Erweiterung gehört, im System aktiv ist. Andernfalls wird der Aufruf ignoriert.

Exit-Funktionsbausteine haben eine innerhalb eines Programms eindeutige, dreistellige Nummer. In der Syntax des CALL CUSTOMER-FUNCTION-Befehls wird lediglich diese Nummer angegeben:

```
PROGRAM <program_name>.  
...  
CALL CUSTOMER-FUNCTION '001'  
EXPORTING  
...  
IMPORTING  
...  
...
```

Abbildung 27: Aufruf des Exitfunktionsbausteins aus dem SAP-Programm

Der Funktionsbaustein selbst liegt in einer Funktionsgruppe deren Name mit X beginnt. Der Name des Funktionsbausteins setzt sich, jeweils durch einen Unterstrich getrennt, aus exit, dem Namen des SAP-Programms und der dreistelligen Nummer zusammen.

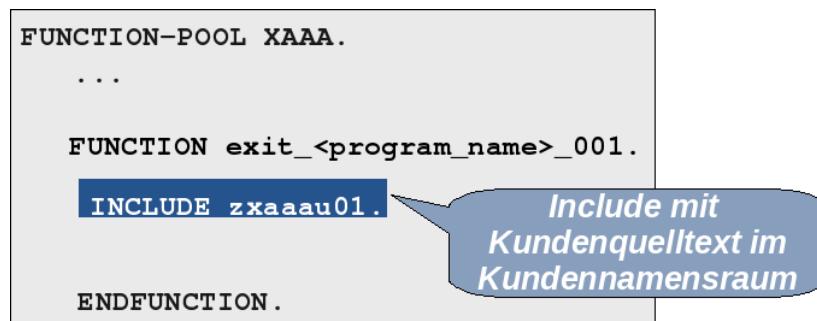


Abbildung 28: Definition eines Exit-Funktionsbausteins

Auf den ersten Blick mag es für Sie als nicht-SAP-interne Entwickler wenig interessant sein, wie der Aufruf eines Exit-Funktionsbausteins aus dem SAP-Programm erfolgt. Auf den zweiten Blick ist jedoch zu erkennen, dass die Kenntnis des Befehls hilfreich sein kann, um in einem SAP-Programm nach Erweiterungsmöglichkeiten zu suchen.

In einem ersten Schritt kann dabei der Quellcode des SAP-Programms durchsucht werden. Dieser kann angezeigt werden, indem Sie aus dem Menü den Pfad **System -> Status** wählen, und dort doppelt auf den Programmnamen klicken:

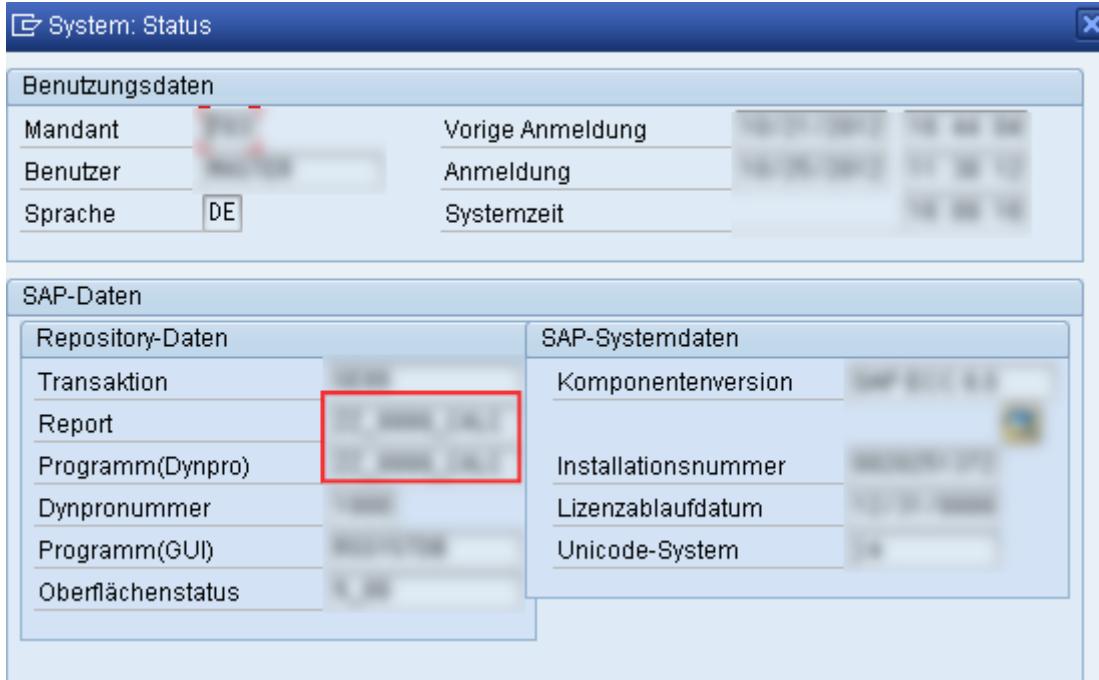


Abbildung 29: Aufruf des Quelltextes per System-Status: SAP-System-Screenshot

Dort können Sie den Menüpfad **Bearbeiten -> Suchen/Ersetzten...** wählen und global im Programm nach der Zeichenkette **CALL CUSTOMER** suchen.

Die zweite Möglichkeit, nach Programm-Exits zu suchen, ist die Transaktion **CMOD**. Dort gibt es unter dem Menüpfad **Hilfsmittel -> SAP-Erweiterungen** eine Verzweigung ins

Repository-Infosystem. Durch die Suche nach **EXIT\_programmname\_\*** als Komponentenname können Exits zum Programm **programmname** gefunden werden.

Weiterhin können Sie direkt über die Tools wie das Repository Infosystem suchen.

### 6.2.3 Praxis: Beispiel-Suche nach Customer-Exits

Öffnen Sie das **Repository Infosystem**. Sie finden es im Object Navigator oberhalb des Navigationsbaums. Wählen Sie anschließend aus dem Navigationsbaum des Repository Infosystems den Pfad **Repository-Infosystem -> Erweiterungen -> Customer-Exits -> Erweiterungen**. Es erscheint diese Maske:

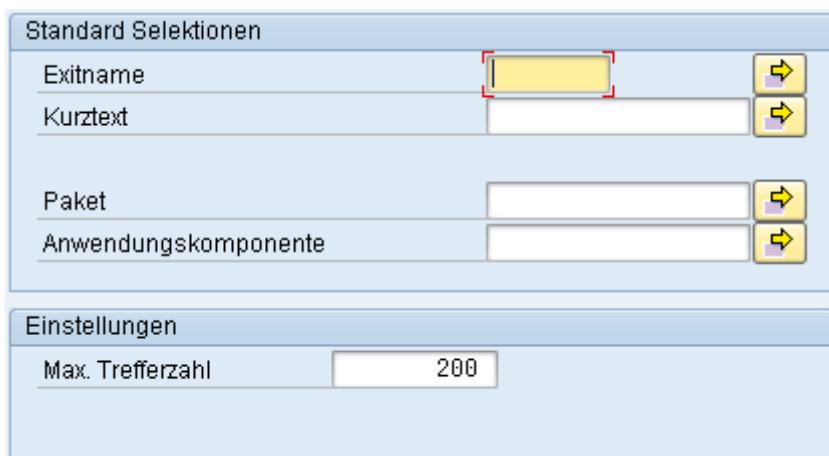


Abbildung 30: Suchmaske für Erweiterungen: SAP-System-Screenshot

Klicken Sie auf . Hierdurch werden alle Felder angezeigt, die für eine Suche zur Verfügung stehen. Als Beispiel sollen Exits für SAPLBD11 gesucht werden. Geben Sie daher in das Feld **Komponentenname** den Wert **EXIT\_SAPLBD11\_\*** ein. Gemäß der Namensgebung von Exit-Funktionsbausteinen werden so alle Bausteine für SAPLBD11 gefunden. Bestätigen Sie die Suche. Sie finden ein Ergebnis:

Exitname	Kurzbeschreibung
ALE00001	ALE User-Exit

Abbildung 31: Ergebnis der Suche: SAP-System-Screenshot

Doppelklicken Sie auf ALE00001. Sie sehen nun die Komponenten der Erweiterung. In diesem Fall handelt es sich um eine Exit-Funktion. Positionieren Sie den Cursor auf der Exit-Funktion und wählen Sie den Menüpfad **Springen -> Dokumentation**. Sie sehen nun Informationen zur Verwendung, insbesondere wird die Schnittstelle beschrieben.

Schließen Sie die Dokumentation wieder und klicken Sie diesmal doppelt auf die Exit-Funktion. Sie gelangen zum Code des Funktionsbausteins. Dieser besteht nur aus einer einzigen Anweisung, die ein Include einbindet. Der Name dieses Includes beginnt mit Z und liegt im Kundennamensraum. Ein Kunde, der das Exit verwenden möchte, könnte dieses Include anlegen und seine kundenspezifischen Befehle dort implementieren.

Da das Include zwischen FUNCTION und ENDFUNCTION gerufen wird, dürfen dort keine Definitionen von Unterprogrammen oder Modulen vorgenommen werden (FORM...ENDFORM bzw. MODULE...ENDMODULE), weiterhin dürfen keine Ereignisse der Reportverarbeitung (AT SELECTION-SCREEN usw.) angegeben werden.

Verlassen Sie das Exit und das Repository Infosystem nun wieder.

#### 6.2.4 Aufbau von Exit-Funktionsgruppen

Die folgende Abbildung skizziert den Aufbau einer Exit-Funktionsgruppe:

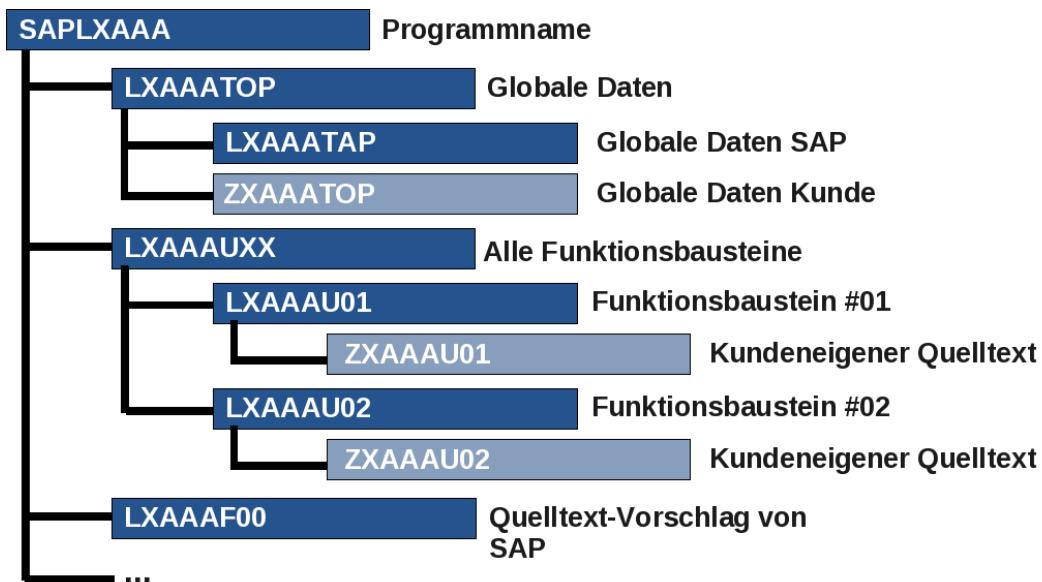


Abbildung 32: Aufbau einer Exit-Funktionsgruppe

Eine Exit-Funktionsgruppe ist wie eine normale Funktionsgruppe aus Includes zusammengesetzt: Globale Daten werden in einem TOP-Include definiert, Funktionsbausteine liegen in Nummerierten Includes, die wiederum in einem Include mit der Endung UXX liegen. Das Besondere an der Exit-Funktionsgruppe sind die Kunden-Includes: Sie enthält neben Includes des SAP-Namensraums, deren Namen mit LX beginnen, Includes die mit ZX beginnen und so im Kundennamensraum liegen und somit vom Kunden bearbeitet werden dürfen. Es ist dem Kunden nicht erlaubt, zusätzliche Funktionsbausteine innerhalb der Exit-Funktionsgruppe zu definieren.

Das in der Abbildung dargestellt Include LXAAAF00 ist ein Quelltext-Vorschlag. Ein solcher Vorschlag kann vom SAP-Entwickler angelegt werden. Der Kunde hat dann im Rahmen der Projektverwaltung diesen Quelltext-Vorschlag in sein Include zu übernehmen.

Sie haben bereits gesehen, dass im Exit-Funktionsbaustein ein Include des Kundennamensraums eingebunden wird. In diesem Include hinterlegt der Kunde seinen Quellcode. Es ist aber auch möglich, als Kunde globale Daten zu definieren. Das TOP-Include der Exit-Funktionsgruppe inkludiert ein Include LX...TAP und ein Include ZX...TOP. Ersteres enthält die Globalen Datendefinitionen der SAP, letzteres die des Kunden.

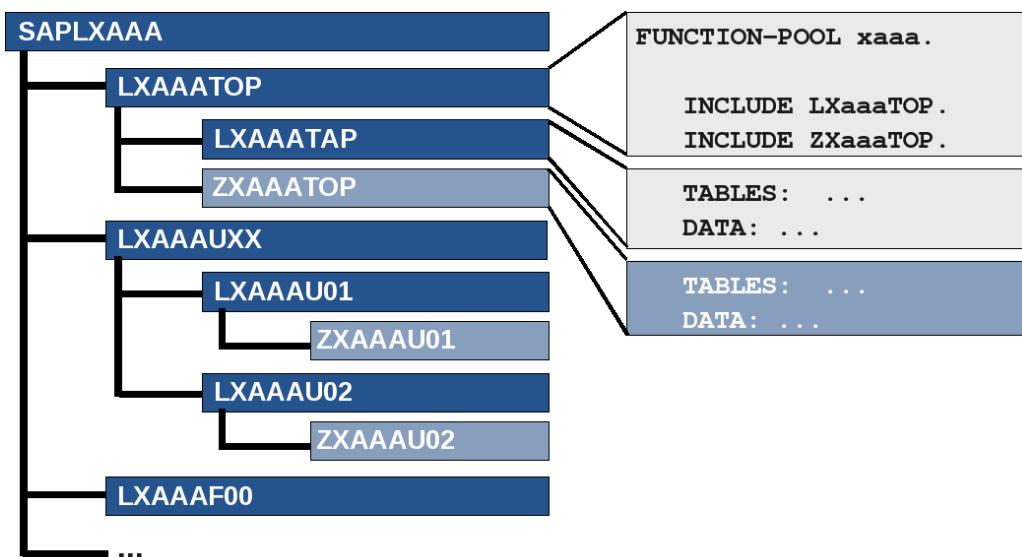


Abbildung 33: Globale Datendefinitionen in einer Exit-Funktionsgruppe

Wie Sie in der Abbildung sehen können, befindet sich die FUNCTION-POOL-Anweisung im TOP-Include im SAP-Namensraum. Sie kann daher durch den Kunden nicht verändert werden, um etwa eine Nachrichtenklasse anzugeben. Die Nachrichtenklasse muss daher ggf. im MESSAGE-Befehl explizit angegeben werden.

Es gibt einige weitere SAP-Objekte, die in einer Exit-Funktionsgruppe auftauchen können. Dies sind Unterprogramme (LX...F01), PBO-Module (LX...O01), PAI-Module (LX...I01) und Ereignisse (LX...E01). Mit **Ereignissen** sind hier nicht die Ereignisse im Sinne von ABAP Objects gemeint, sondern die Ereignisse der Reportverarbeitung wie etwa AT SELECTION-SCREEN.

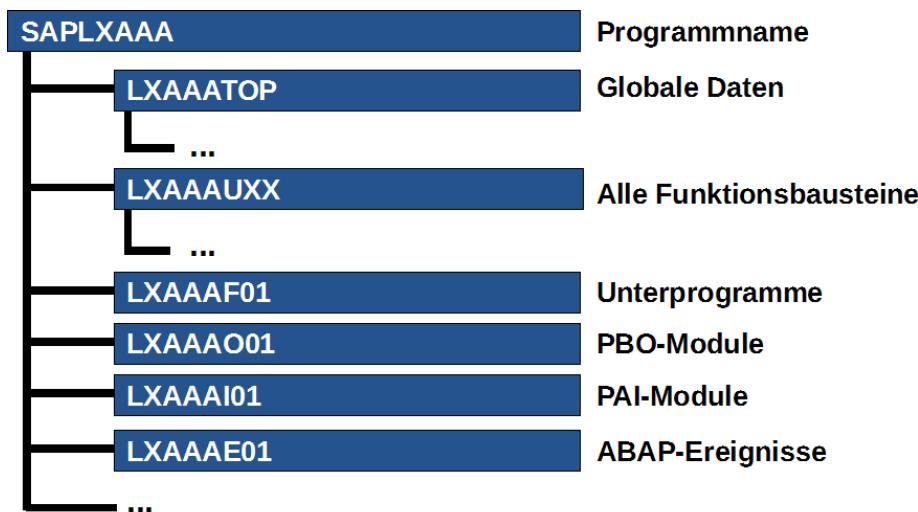


Abbildung 34: SAP-Objekte in einer Exit-Funktionsgruppe

Analog dazu kann auch der Kunde weitere Objekte in einer Exit-Funktionsgruppe anlegen. Dazu steht ihm ein Include ZX...ZZZ zur Verfügung. In diesem werden durch den INCLUDE-Befehl weitere Includes im Kundennamensraum zur Definition von Unterprogrammen, PBO-/PAI-Modulen und Ereignissen eingebunden. Der Aufbau der Namen dieser Includes ist analog zu den SAP-Includes, nur dass er mit Z beginnt (siehe auch folgende Abbildung).

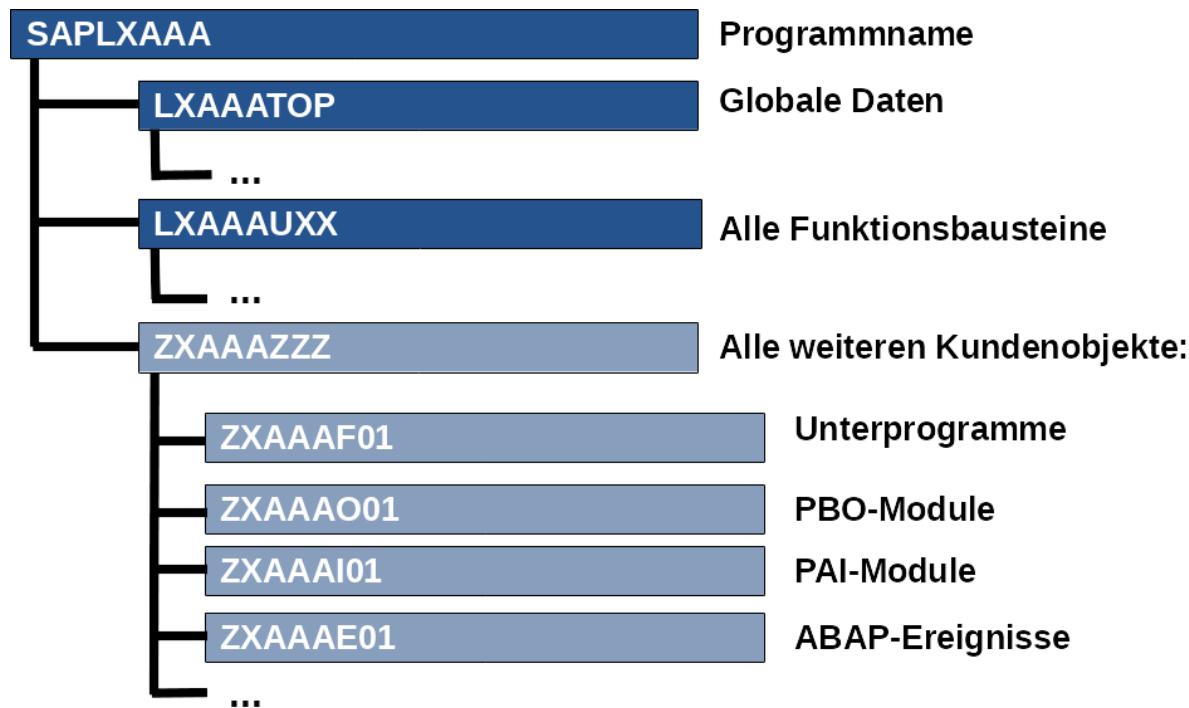


Abbildung 35: Kundeneigene Objekte in einer Exit-Funktionsgruppe

In einem Programm-Exit können Sie nicht nur z. B. Berechnungen durchführen, sondern auch eigene Dynpros definieren und aufrufen. Diese werden dann an der Stelle des Exit-Aufrufs in die Dynprofolge eingebunden. So können etwa zusätzliche Daten vom Benutzer eingegeben werden, die anschließend in einem PAI-Modul des Kunden verarbeitet werden (beachten Sie,

dass dies nicht als Dynpro-Exit bezeichnet wird, dieser Begriff bezeichnet Exits in SAP-Dynpros und wird Ihnen später vorgestellt). Der Aufruf geschieht wie in normalen Programmen mit dem CALL SCREEN-Befehl. Das Anlegen des Dynpros selbst wird durch Vorwärtsnavigation, d. h. Doppelklick auf die Dynpronummer, unterstützt. Die PAI-/PBO-Module befinden sich in den oben dargestellten Includes.

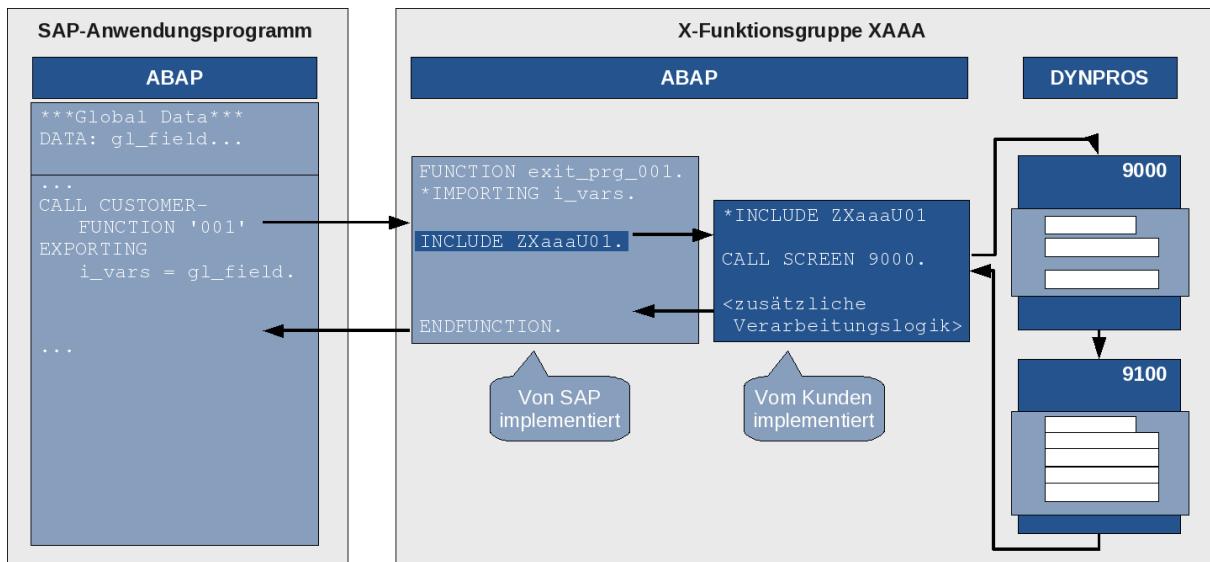


Abbildung 36: Beispiel für die Einbindung von Dynpros durch Programm-Exits

## 6.2.5 Menü-Exits

Menü-Exits werden dazu verwendet, in die Menüs von SAP-eigenen Programmen kundenspezifische Einträge hinzufügen zu können. Hinter diesen Einträgen kann der Kunde dann entweder eine vorgegebene Funktionalität hinterlegen oder ein Programm-Exit implementieren. Die Entscheidung darüber trifft der SAP-Entwickler.

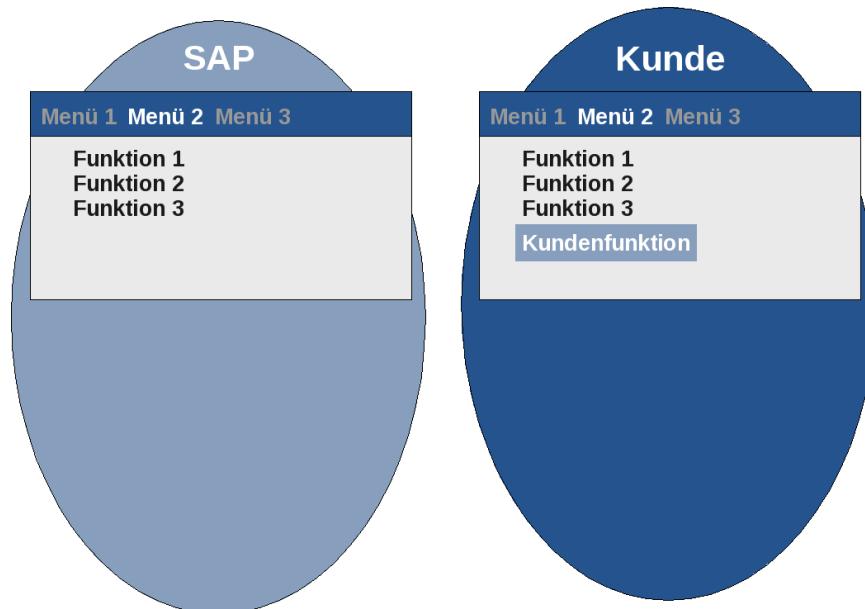


Abbildung 37: Menü-Exits

Der SAP-Entwickler reserviert im Menü eines Programms bestimmte Menüeinträge. Diese sind zunächst nicht sichtbar. Erst wenn Sie vom Kunden im Rahmen eines Erweiterungsprojekts aktiviert werden, erscheinen sie im Menü und führen den Anwender zur hinterlegten Funktionalität. Die reservierten Menüpunkte für Menü-Exits haben spezielle Funktionscodes die mit einem Pluszeichen (+) beginnen.

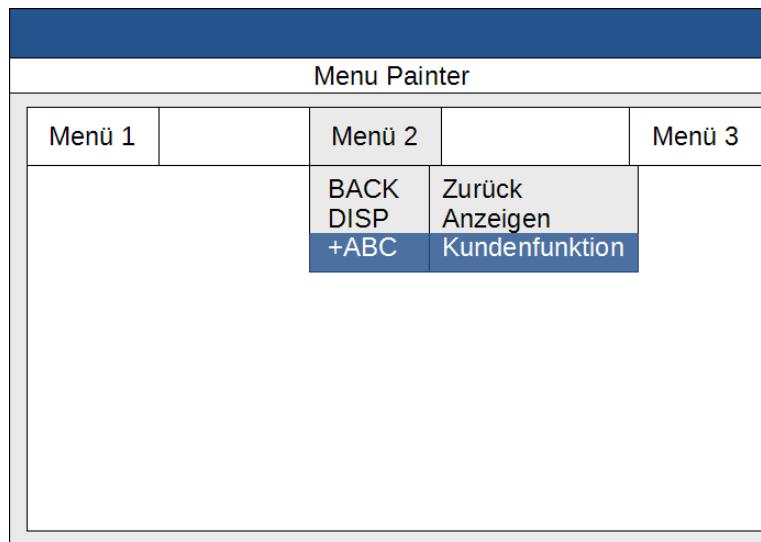


Abbildung 38: Menü mit reservierter Kundenfunktion

Im PAI des Dynpros verarbeitet der SAP-Anwendungsentwickler den reservierten Funktionscode. Hier kann mit dem CALL CUSTOMER-FUNCTION-Befehl ein Programm-Exit verwendet werden. Im betreffenden Funktionsbaustein kann der Kunde dann – wie zuvor gesehen – seine Befehle über ein Include im Kundennamensraum einbinden.

PROGRAM <program\_name>.  
DATA ok\_code LIKE sy\_ucomm.  
...  
\* PAI-Modul :  
CASE ok\_code.  
 WHEN 'DISP'.  
 ...  
 WHEN '+ABC'.  
 CALL CUSTOMER-FUNCTION '001'  
 EXPORTING  
 <i\_variables>  
 IMPORTING  
 <e\_variables>.  
 ...  
ENDCASE.  
...

A callout bubble points to the 'WHEN '+ABC'.' line of code with the text: 'Durch Programm-Exit realisierte Kunden-Funktionalität'.

Abbildung 39: Menü-Exit, dass ein Programm-Exit verwendet

Um ein Menü-Exit zu verwenden, bindet der Kunde die zugehörige Erweiterung in der Transaktion CMOD in sein Erweiterungsprojekt ein. Hierbei wird auch die Bezeichnung für

den Menüpunkt durch den Kunden festgelegt. Anschließend implementiert er, sofern vorhanden, den Programm-Exit durch Anlegen des entsprechenden Includes und hinterlegen der Funktionalität. Nach Aktivierung des Includes und des Erweiterungsprojekts wird der neue Menüpunkt in der Anwendung sichtbar.

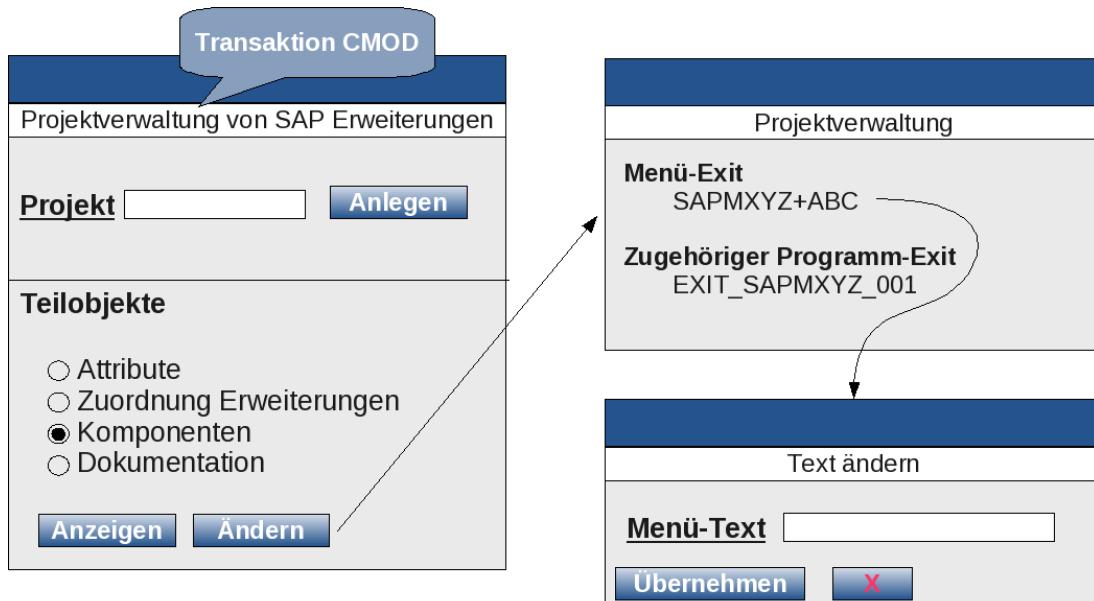


Abbildung 40: Festlegen des Menütextes für ein Menü-Exit

### 6.2.6 Dynpro-Exits

Die dritte Art von Customer-Exits sind Dynpro-Exits. Sie haben bereits bei den Programm-Exits kundenspezifische Dynpros kennen gelernt, die aus dem Include des Kunden aufgerufen wurden. Bei Dynpro-Exits geht es dagegen darum, innerhalb eines bestehenden SAP-Dynpros Ergänzungen vorzunehmen.

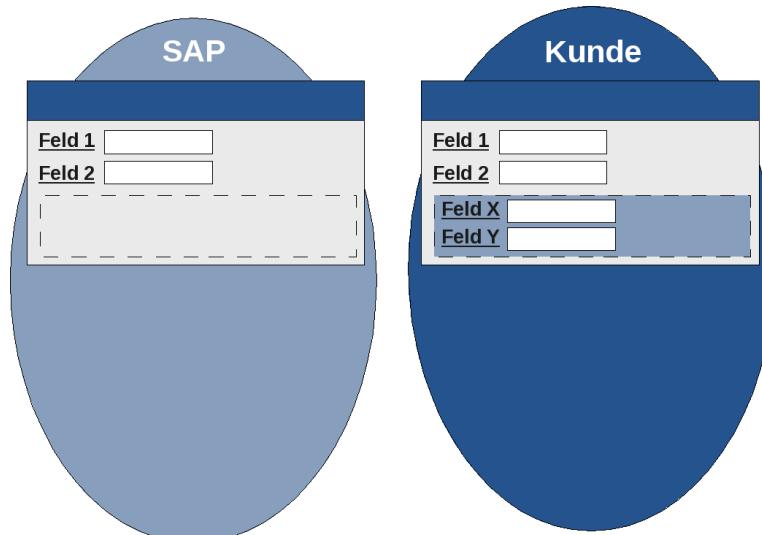


Abbildung 41: Dynpro-Exits

Diese Ergänzungen werden mit Subscreen-Bereichen und Subscreens realisiert. Der SAP-Entwickler definiert einen Subscreen-Bereich auf dem Dynpro des SAP-Programms, welches der Kunde erweitern können soll. Der Kunde definiert den Subscreen mit den von ihm gewünschten Feldern und der von ihm benötigten Logik.

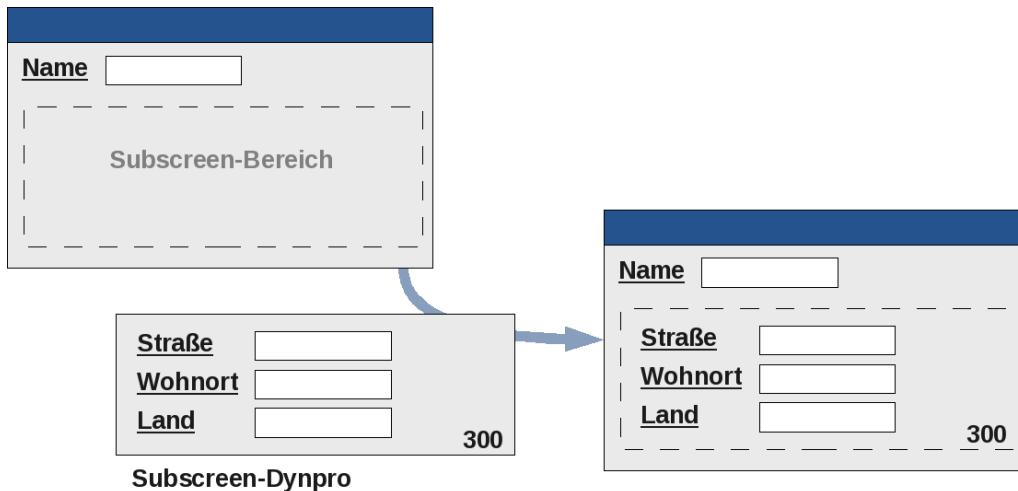


Abbildung 42: Subscreen Exits

Subscreens kennen Sie bereits aus dem Kurs „Einführung in ABAP“. Auf einem gewöhnlichen Dynpro wird ein Bereich reserviert, in den zur Laufzeit ein anderes Dynpro geladen wird. Dieses Dynpro muss explizit als Subscreen-Dynpro gekennzeichnet sein.

Im PAI und PBO des Dynpros mit dem Subscreen-Bereich wird der entsprechende Block des Subscreen-Dynpro gerufen. Dies geschieht bei herkömmlichen Subscreens mit dem CALL SUBSCREEN-Befehl, dem im PBO neben dem Namen des Subscreen-Bereiches auch der Name des einzubindenden Subscreens und des Programms, in dem dieser definiert ist, mitgegeben werden. Durch die Definition des Subscreen-Bereichs ist also noch nicht festgelegt, welcher Subscreen in diesen Bereich tatsächlich geladen wird.

```

PROCESS BEFORE OUTPUT.
  MODULE ...  
  

  CALL SUBSCREEN abcd
    INCLUDING sy_cprog '1234'.
  MODULE ...  
  

PROCESS AFTER INPUT.
  MODULE ...  
  

  CALL SUBSCREEN abcd.  
  

  MODULE user_command_0100.

```

Abbildung 43: Syntax zum Aufruf eines (normalen) Subscreens

Dynpro-Exits basieren auf dem gleichen Konzept: Auf dem Dynpro eines SAP-Programms werden Subscreen-Bereiche definiert (es sind mehrere Bereiche auf einem Dynpro möglich), und in diese Bereiche werden vom Kunden entwickelte Subscreens geladen. Technisch bestehen die gleichen Restriktionen wie bei der herkömmlichen Verwendung von Subscreens: Diese dürfen das OK-Code-Feld nicht benennen, keinen GUI-Status setzen, kein Folgedynpro setzen und führen keine Verarbeitung des Funktionscodes durch. Die Syntax zum Aufruf der Subscreens in PBO und PAI ist bei Dynpro-Exits etwas anders als bei herkömmlichen Subscreen-Einbindungen:

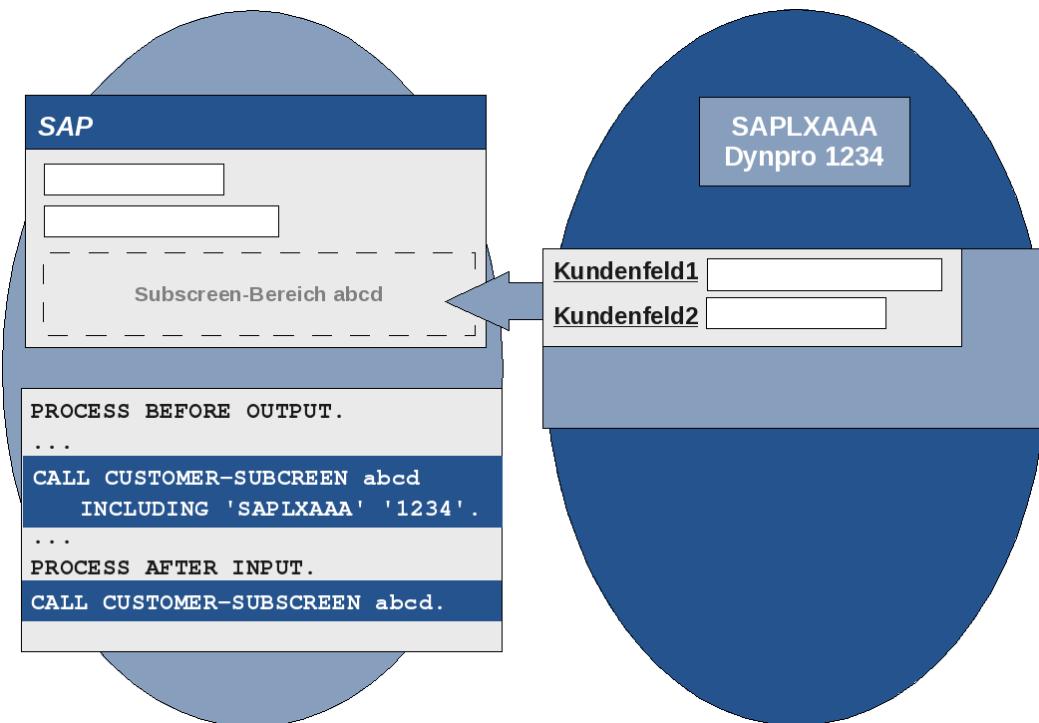


Abbildung 44: Beispiel zur Einbindung per Dynpro-Exit

Es wird hier statt des Befehls CALL SUBSCREEN, den Sie von herkömmlichen Subscreens kennen, von SAP der Befehl CALL CUSTOMER-SUBSCREEN verwendet. Der Name des Subscreen-Bereiches wird hier wie beim CALL SUBSCREEN-Befehl gewohnt im PBO angegeben. Das eingebundene Subscreen-Dynpro wird vom Kunden in einer X-Funktionsgruppe angelegt. Der Name der Funktionsgruppe muss statisch in Hochkommata angegeben werden (in der obigen Abbildung 'SAPLXAAA'), während für die vierstellige Dynpronummer auch eine Variable angegeben werden darf.

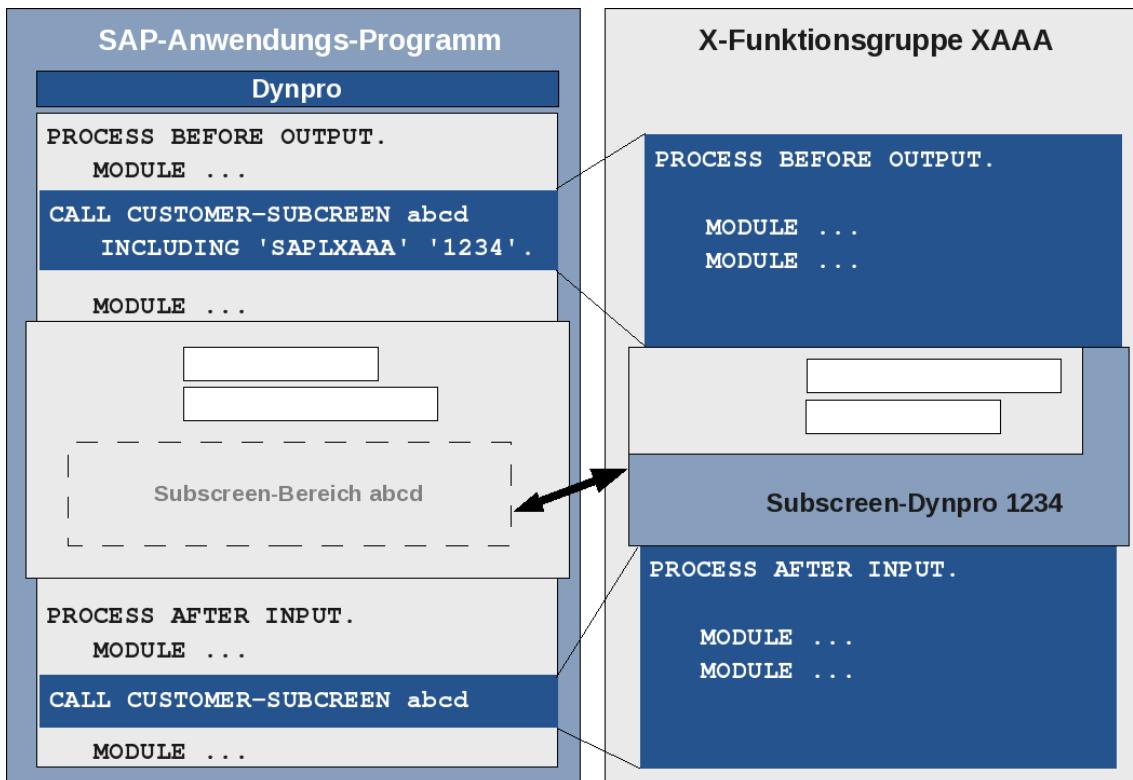


Abbildung 45: Aufruf der PBO- / PAI-Module

Analog zu herkömmlichen Subscreens werden auch durch den CALL CUSTOMER-SUBSCREEN-Befehl PBO bzw. PAI des Subscreen ausgeführt. Die entsprechenden Module werden in Includes in der jeweiligen X-Funktionsgruppe durch den Kunden definiert. Dies führt zu einem Problem beim Transport der Datenobjekte. In der Funktionsgruppe sind die globalen Daten des SAP-Anwendungsprogramms nicht bekannt, und so könnte in den Modulen nicht auf diese Daten zugegriffen werden. Um dieses Problem zu lösen, definiert der SAP-Entwickler ein Programm-Exit. Über diesen Exit werden die Daten aus den globalen Feldern des SAP-Anwendungsprogramms an die X-Funktionsgruppe übertragen und stehen dort in globalen Datenobjekten zur Verfügung. Die folgende Abbildung veranschaulicht diesen Vorgang.

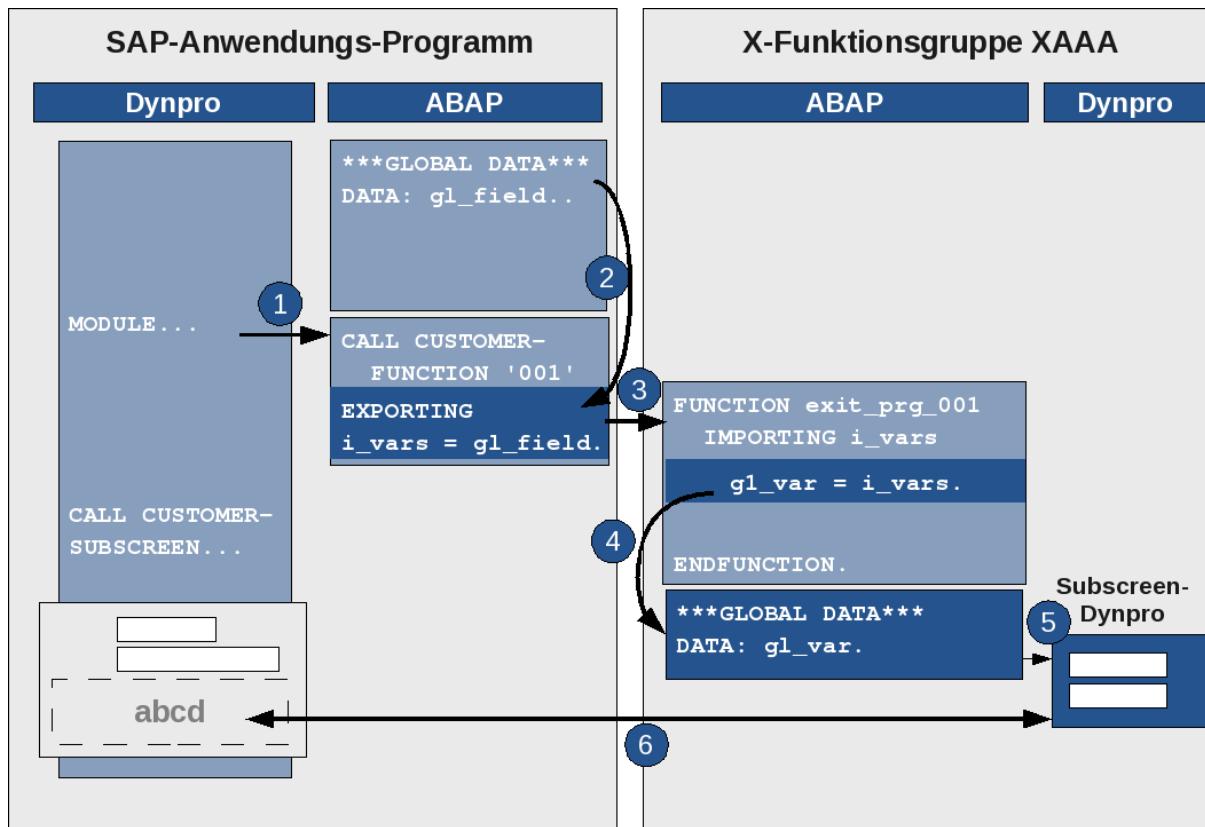


Abbildung 46: Datentransport zum Subscreen-Dynpro

Der Datentransport geschieht hier auf folgendem Weg:

1. Aufruf eines PBO-Moduls des SAP-Dynpros (vor dem CALL CUSTOMER-SUBSCREEN-Befehl).
2. Die globalen Daten des SAP-Anwendungsprogramms werden für den Aufruf eines Programm-Exits benutzt.
3. Die Daten landen so in der entsprechenden X-Funktionsgruppe und werden...
4. ...in die globalen Datenfelder dieser X-Funktionsgruppe geschrieben.
5. Im Subscreen-Dynpro stehen diese globalen Daten zur Verfügung.
6. Das Subscreen-Dynpro wird (durch den CALL CUSTOMER-SUBSCREEN-Befehl) gerufen und im Subscreen-Bereich des SAP-Dynpro angezeigt.

Im PAI besteht ein ähnliches Problem: Die Daten, die der Benutzer im Subscreen-Dynpro eingegeben hat, landen in den globalen Datenobjekten der X-Funktionsgruppe. Der SAP-Programmierer verwendet daher ein zweites Programm-Exit, um die Daten zurück zu holen. Die folgende Abbildung veranschaulicht den Vorgang.

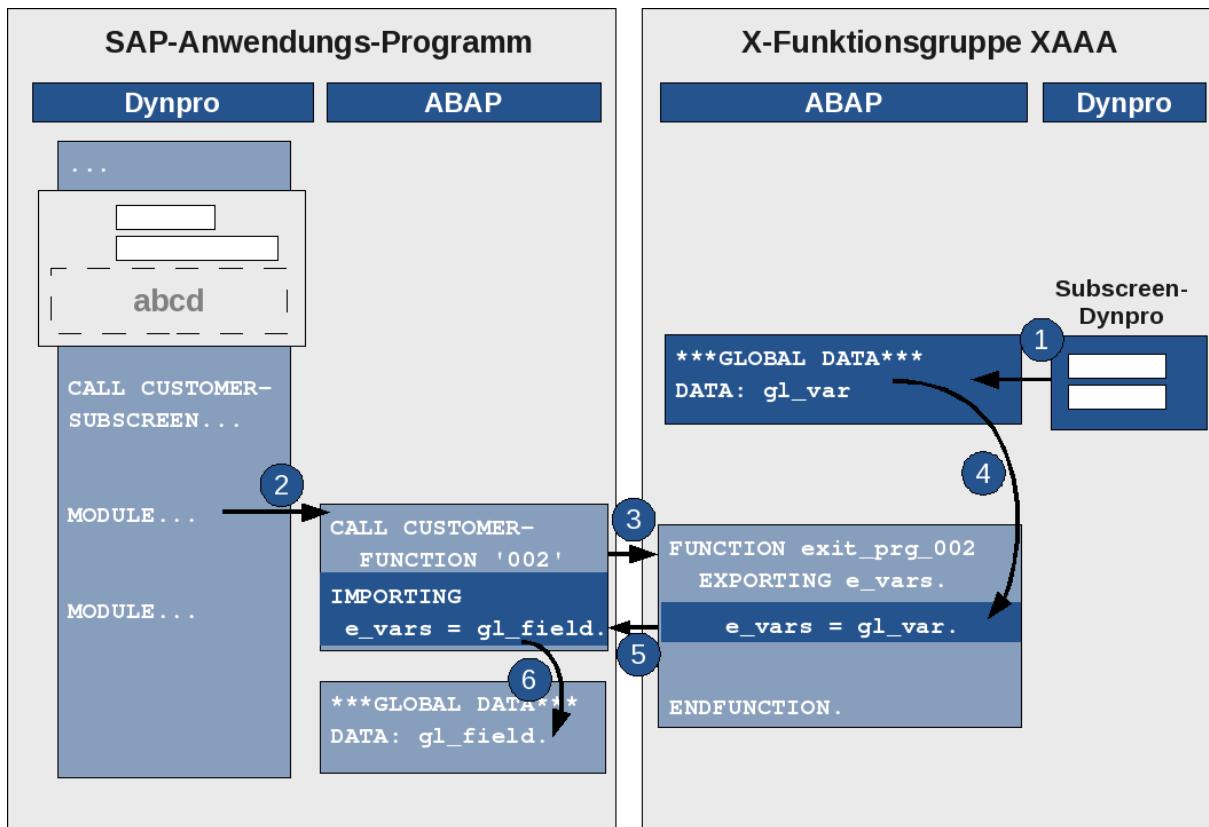


Abbildung 47: Datentransport vom Subscreen-Dynpro

Der Datentransport nimmt hier folgenden Weg:

- Die vom Benutzer eingegebenen Daten werden im (vom CALL CUSTOMER-SUBSCREEN-Befehl ausgelösten) PAI des Subscreens in den globalen Datenbereich der X-Funktionsgruppe transportiert.
- Im PAI des SAP-Dynpros wird ein Modul für das holen der Daten eingebunden.
- Im Modul wird ein Programm-Exit aufgerufen.
- Das Programm-Exit greift auf die globalen Daten der X-Funktionsgruppe zu und...
- ...liefert diese zurück.
- Die Daten werden in den globalen Feldern des SAP-Programms abgelegt und können für die weitere Verarbeitung verwendet werden.

Dynpro-Exits sind – wie die andern Customer-Exits auch – Komponenten von Erweiterungen, die der Kunde in Erweiterungsprojekten zusammenfasst und über die Transaktion **CMOD** verwaltet.

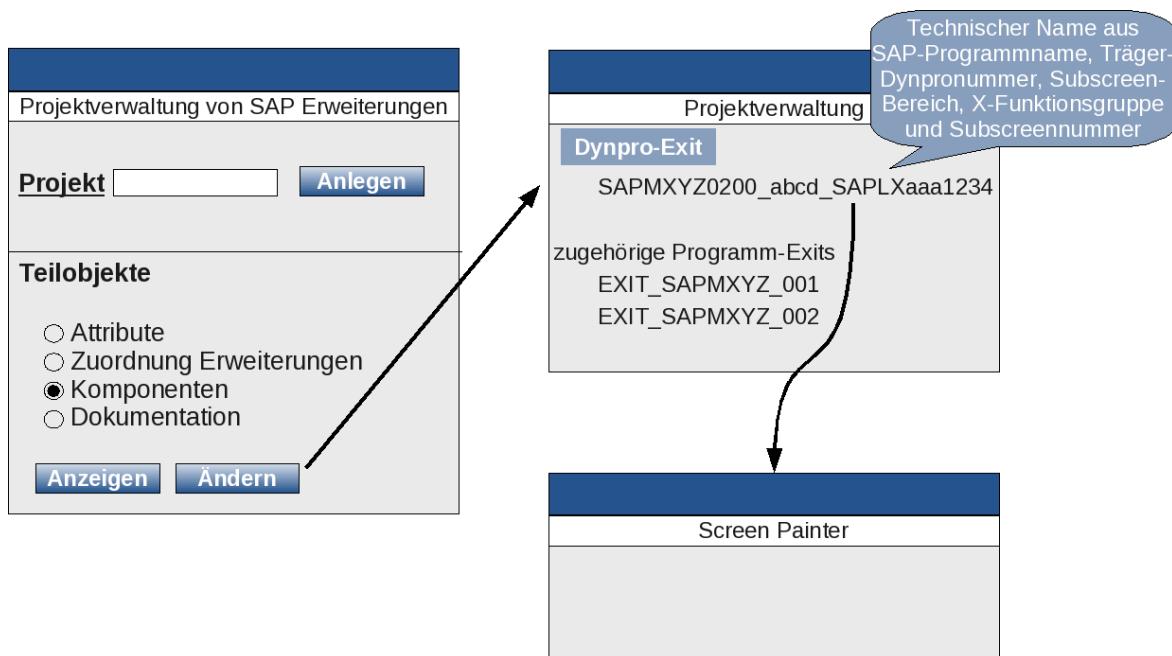


Abbildung 48: Name und Bearbeitung eines Dynpro-Exits

Aus der Projektverwaltung kann das Dynpro-Exit direkt durch Vorwärtsnavigation aus der Komponentenpflege (siehe obige Abbildung) umgesetzt werden.

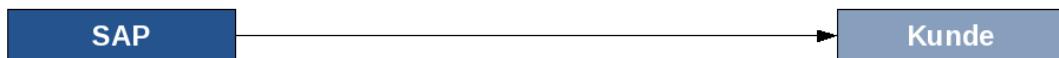
Um bei einem gegebenen Programm herauszufinden, ob dieses Dynpro-Exits anbietet, kann die Ablauflogik des betreffenden Dynpros nach der Anweisung CALL CUSTOMER-SUBSCREEN durchsucht werden.

### 6.3 Klassische Business Add Ins

Zum Release R/3 4.6 ist im SAP-System eine neue Erweiterungstechnik hinzugekommen. Es handelt sich um die klassischen Business Add Ins (kurz BAdIs). Es wird hier der Begriff „klassisch“ verwendet, weil es mittlerweile auch ein neues BADI-Konzept gibt, das Ihnen in einem späteren Abschnitt vorgestellt wird.

Der Bedarf nach einem neuen Erweiterungskonzept erwuchs aus den Mängeln bisheriger Konzepte und neuen Wegen bei der Auslieferung von Software.

#### Klassischer Weg der Auslieferung



#### Heutiger Weg der Auslieferung



Abbildung 49: Veränderte Software-Auslieferungswege

Während früher SAP die Produkte direkt an den Kunden ausgeliefert hat, gibt es heute vielfach Intermediäre. Dies können wie in der Abbildung dargestellt IBUs (Industrial Business Units) und sonstige Partner sein. Durch diese Intermediäre werden Anpassungen vorgenommen, um die Software etwa auf bestimmte Branchen auszurichten oder anderweitige Verbesserungen an der Software zu erreichen. Dies ist zunächst ein Vorteil für den Kunden, der ein Produkt erhält, das besser auf die Bedürfnisse von ihm bzw. seiner Branche zugeschnitten ist.

Mit den bisherigen Erweiterungskonzepten vor Einführung der BAdIs kann es auf diesem Weg aber zu Problemen kommen. Das ist dann der Fall, wenn entlang der Auslieferungskette mehrere Akteure Anpassungen an derselben Komponente des SAP-Systems vornehmen wollen. Die im letzten Abschnitt beschriebenen Customer-Exits (Programm-Exits, Dynpro-Exits und Menü-Exits) sind jeweils nur einmal verwendbar. Hat z. B. der Partner ein bestimmtes Programm-Exit für seine Anpassung benutzt, steht dieses für den Kunden nicht mehr zur Verfügung und er kann an dieser Stelle keine Erweiterung der Software mehr vornehmen. Die (hier nicht näher behandelte) BTE-Technik bietet zwar die Möglichkeiten der Mehrfachverwendung und Filterung an, kann jedoch nicht für Menü- und Dynpro-Exits verwendet werden und besitzt keine Verwaltungsebene. Um die neuen Problemstellungen meistern zu können, wurde nach einem neuen Erweiterungskonzept gesucht, dass die Anforderung der Mehrfachverwendung unterstützt und gleichzeitig für dieselben Einsatzgebiete wie Customer-Exits (Programm-, Dynpro- und Menü-Exits) angewendet werden kann, über eine Verwaltungsebene verfügt und die neuesten Technologien nutzt. Das Ergebnis waren die (klassischen) BAdIs, die die geforderte Funktionalität konsequent objektorientiert umsetzen.

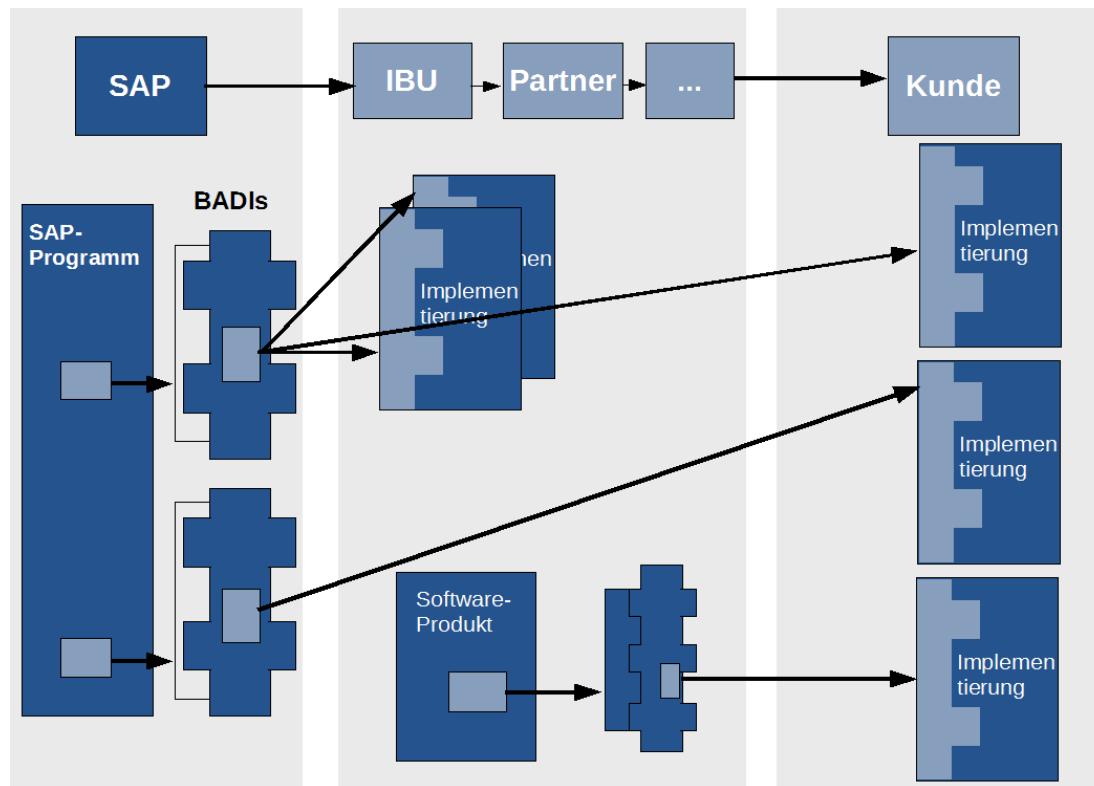


Abbildung 50: Architektur klassischer BAdIs im Kontext der Auslieferungskette

Ein BAdI besteht aus objektorientierter Sicht aus einem BAdI-Interface und einer BAdI-Klasse. Die BAdI-Klasse implementiert das BAdI-Interface. Zur Definition eines BAdI wird nur das BAdI-Interface benötigt. Das SAP-System generiert daraus automatisch eine BAdI-Klasse. Auch der Verwender implementiert das BAdI-Interface. Wie die obige Abbildung skizziert, kann es zu einem BAdI mehrere Implementierungen geben. Die wesentliche Anforderung der Mehrfachverwendbarkeit wurde bei diesem Konzept also umgesetzt. So kann ein weiter am Ende der Auslieferungskette arbeitender Entwickler ein BAdI unabhängig davon verwenden, ob ein weiter am Anfang der Auslieferungskette beteiligter Entwickler dasselbe BAdI bereits für eine Erweiterung der Funktionalität verwendet. BAdIs können auch von Partnern in deren eigenen Produkten angeboten werden, beschränken sich also nicht auf die von SAP vorgegebenen Instanzen.

Wie gefordert können mithilfe von BAdIs Programme, Menüs und Dynpros erweitert werden. Es stehen somit die aus den Customer-Exits bekannten grundlegenden Features zur Verfügung.

### 6.3.1 Programm-Exits mit BAdIs

Auf Seiten des SAP-Programms (oder des Partner-Programms, wenn es sich um ein Produkt eines Partners handelt welches das BAdI anbietet) werden für die Verwendung eines BAdIs im Wesentlichen zwei Schritte benötigt. Als erstes wird eine Hilfsklasse **cl\_exithandler** benutzt, um eine Instanz der BAdI-Klasse zu erhalten. Im zweiten Schritt wird dann eine Methode der BAdI-Klasse gerufen. Das BAdI arbeitet dann alle aktiven Implementierungen ab.

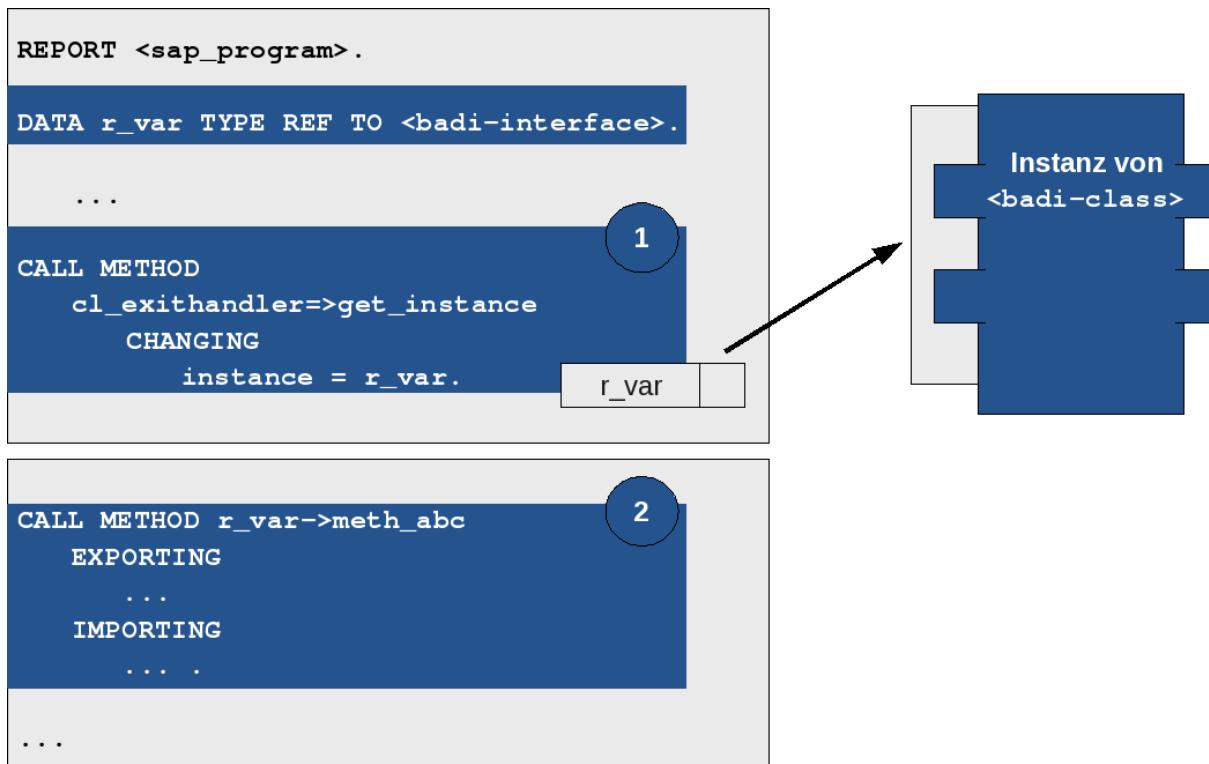


Abbildung 51: Aufruf eines BAdIs auf Seiten des SAP-Programms

Für den Zugriff auf die BAdI-Klasse wird eine Referenzvariable vom Typ des BAdI-Interfaces verwendet. Die Klasse `cl_exithandler` bietet eine statische Methode `get_instance`, die hier aufgerufen wird und die Instanz liefert. Über die Instanzmethode, die im Beispiel `meth_abc` heißt, wird dann das BAdI aufgerufen und dieses sorgt dafür dass die Implementierungen abgearbeitet werden. Die Methode `meth_abc` muss durch den SAP-Entwickler (oder Partner) im BAdI-Interface definiert worden sein.

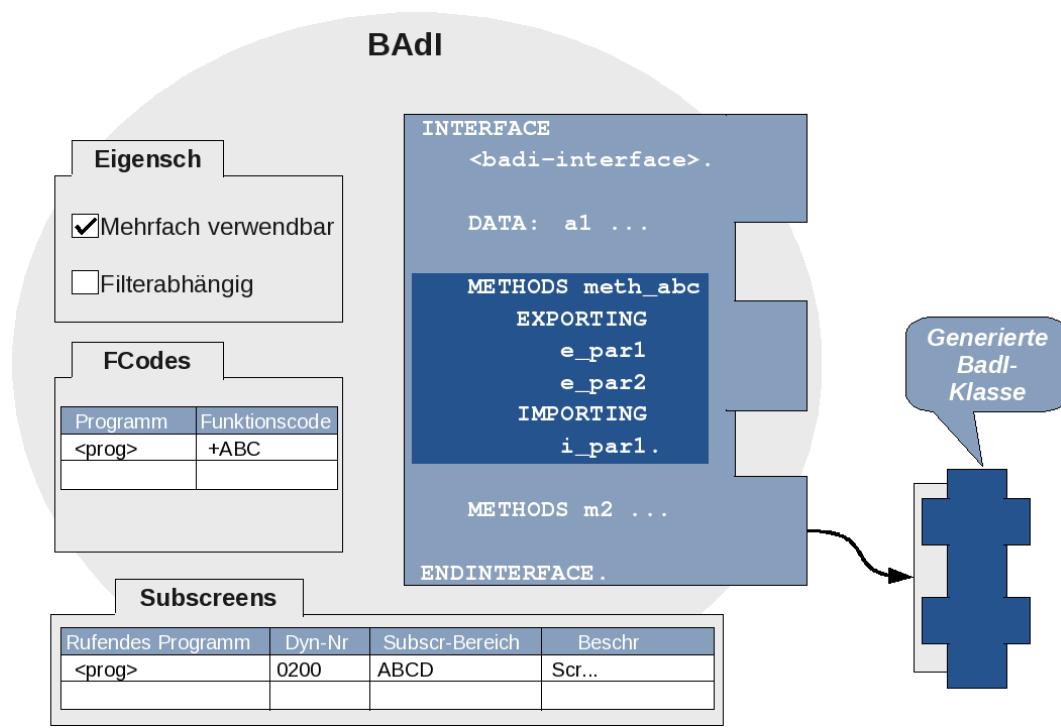


Abbildung 52: Komponenten von BAdIs

Das BAdI (obige Abbildung) enthält also für Programmerweiterungen ein Interface, das die Schnittstelle festlegt, über die die Erweiterung gerufen wird. Sie sehen in der Darstellung auch die generierte BAdI-Klasse, die das System beim Aktivieren erzeugt. Diese hat neben dem Aufrufen der aktiven Implementierungen die Aufgabe, Filter zu handhaben, über die konfiguriert werden kann, dass ein BAdI nur zu bestimmten Bedingungen ausgeführt werden soll. Weiterhin sind die Komponenten zur Erweiterung von Menüs (Funktionscodes) und Dynpros (Subscreens) abgebildet. Das Häkchen **Mehrfach verwendbar** gibt an, ob zu einem Zeitpunkt mehrere Implementierungen des BAdI aktiv sein dürfen. Ist es nicht gesetzt, dürfen zwar weitere Implementierungen im System vorhanden sein, es kann aber stets nur eine dieser Implementierungen aktiv sein.

Nicht abgebildet sind Default- und Beispielimplementierungen. Eine Defaultimplementierung wird ggf. durchlaufen, wenn keine aktive Implementierung existiert. Eine Beispielimplementierung kann hinterlegt werden, um dem Verwender des BAdI eine Vorlage als Basis für seine eigene Implementierung anzubieten. Die Default- und Beispielimplementierungen werden vom Ersteller des BAdI angelegt.

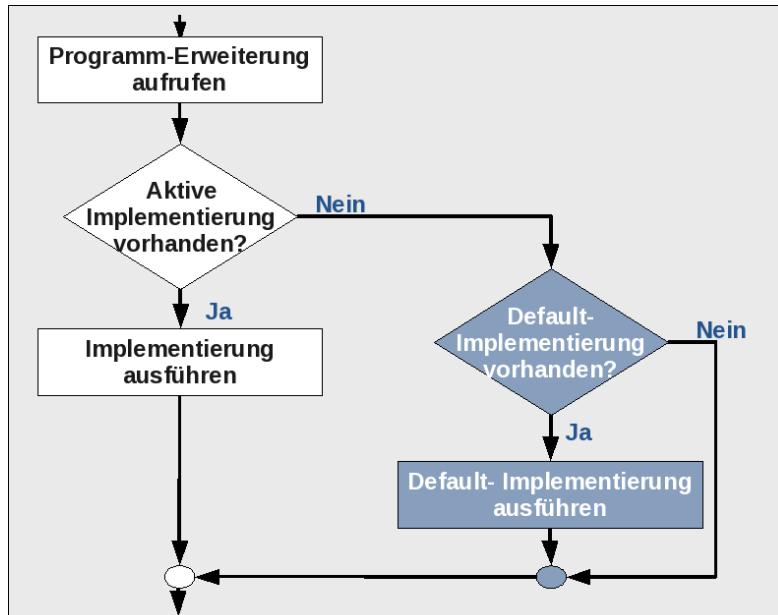


Abbildung 53: Ausführung der Default-Implementierung

Für **filterabhängige BAdIs** wird bei deren Erstellung ein **Filtertyp** angegeben. Hierbei handelt es sich um ein zeichenartiges Datenelement, das den Typ der Filterwerte beschreibt. Es muss eine Suchhilfe oder eine Domäne mit Festwerten oder eine Wertetabelle besitzen. Einen Spezialfall stellen erweiterbare Filtertypen dar. Diese Eigenschaft kann ebenfalls vom BADI-Ersteller gesetzt werden und wird verwendet, wenn das Anlegen einer BADI-Implementierung mit dem Hinzukommen eines neuen Filterwerts korrespondiert und dieser beim Löschen wieder entfernt werden soll. In diesem Fall gibt es einige Bedingungen an den Filtertyp: Die Domäne des Filtertyps muss mit einer mandantenunabhängigen Wertetabelle verbunden sein. Diese wiederum muss genau ein Schlüsselfeld besitzen und dieses muss mit dem Filtertypen-Datenelement typisiert sein. Ferner muss die Wertetabelle eine Texttabelle mit zwei Schlüsselfeldern haben, eines hat das Filtertypen-Datenelement als Typ, das andere ist ein Sprachenfeld. Ein weiteres Feld der Texttabelle enthält „TEXT“ oder „TXT“, um es als Textfeld kenntlich zu machen. Sowohl die Wertetabelle als auch die Texttabelle müssen die Auslieferungsklasse „E“ oder „G“ besitzen. Die folgende Abbildung veranschaulicht die Zusammenhänge.

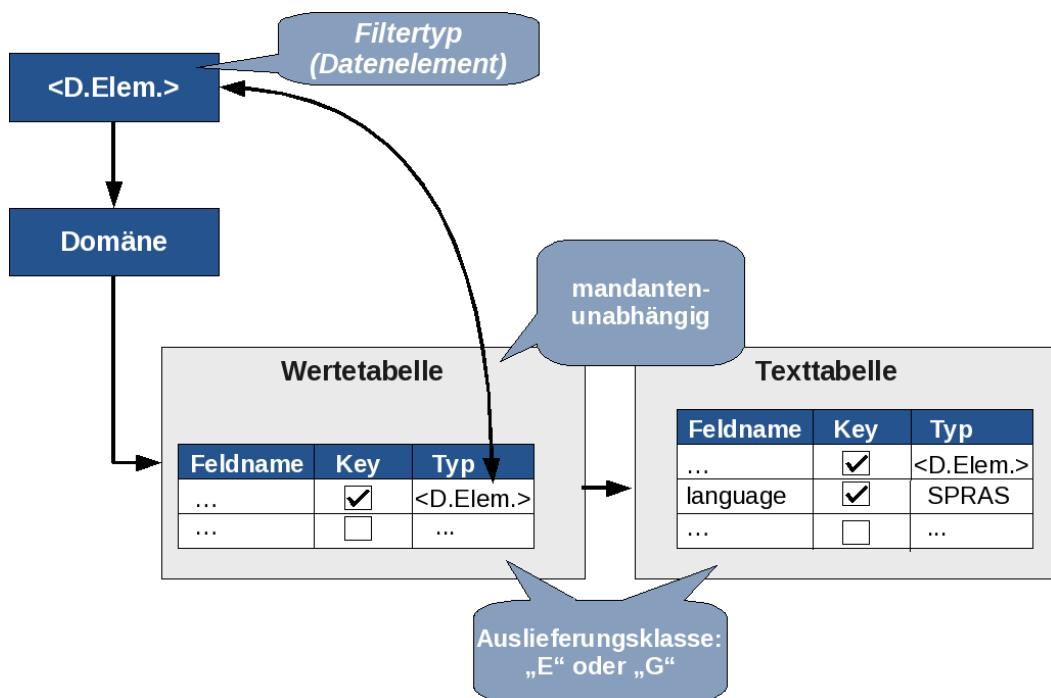


Abbildung 54: Erweiterbare Filtertypen

Um herauszufinden, ob ein gegebenes Programm ein BAdI anbietet, kann per Menüpfad **System -> Status** und Doppelklick auf den Programmnamen zu dessen Quellcode navigiert und dieser (global im Programm) nach der Zeichenkette **cl\_exithandler** durchsucht werden.

Im Fall eines BAdI-Aufrufs finden Sie einen Aufruf der statischen Methode `get_instance` dieser Klasse zum Holen der BAdI-Klasseninstanz. Durch Vorwärtsnavigation können Sie hierüber bis zur Definition des BAdIs gelangen und die Dokumentation abrufen.

Eine programmübergreifende Suche nach BAdIs ist über das Repository Infosystem möglich. Verwenden Sie zur Suche dort den Pfad **Repository-Infosystem -> Erweiterungen -> Business Add Ins -> Definitionen** aus dem Navigationsbaum, um zur entsprechenden Suchmaske zu gelangen.

Zur Implementierung eines BAdIs wird die Transaktion **SE19** verwendet, die sich auch im Easy Access-Menü unter dem Pfad **Werkzeuge -> ABAP Workbench -> Hilfsmittel -> Business Add Ins -> Implementierung** finden lässt.

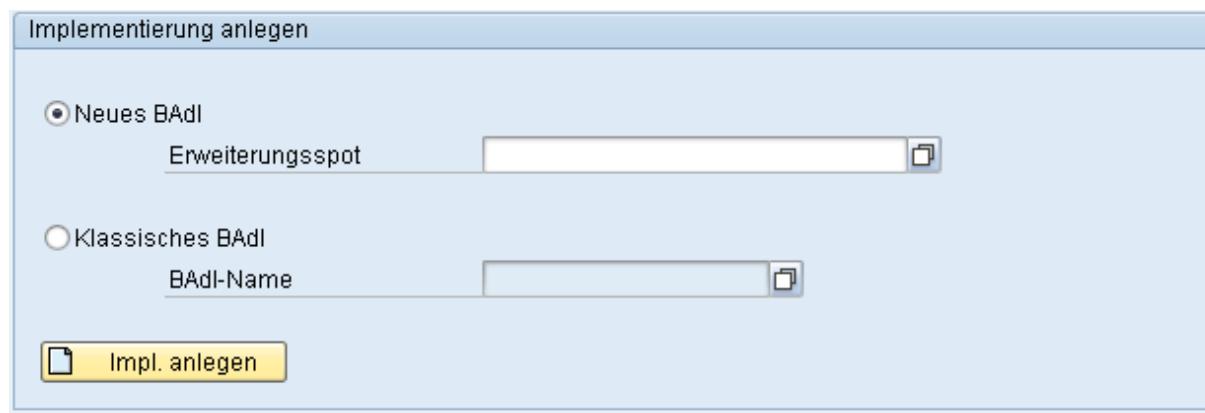


Abbildung 55: Anlegen von BAdIs: SAP-System-Screenshot

Nach Anlegen der Implementierung kann in einer implementierenden Klasse die eigentliche Implementierung der Methoden des BAdI vorgenommen werden.

The screenshot shows the SAP interface for creating a BAdI. On the left, the 'Interface' tab is selected, displaying the 'Interface-Name' as <badi-interface> and the 'Name der implementierenden Klasse' as <impl-class>. Below this, a table lists a single method: <meth-abc> with the description 'Methode des BAdI'. A large blue double-headed arrow points from this table to a separate window titled 'Class Builder: Editieren Methode <badi-interface>~<meth\_abc>'. This window contains the ABAP source code for the method:

```
METHOD <badi-interface>~<meth_abc>.  
--  
ENDMETHOD.
```

Abbildung 56: Implementierung von BAdI-Methoden

Der Name der implementierenden Klasse kann theoretisch frei gewählt werden. Praktisch ist dieser vorgegeben, und im Interesse einheitlicher Namenskonventionen sollte der vorgegebene Name nicht verändert werden. Er setzt sich aus dem Namensraumpräfix (Y oder Z), CL\_IM\_ (Für Class bzw. Implementation) und dem Namen Ihrer Implementierung zusammen.

Durch klare Namenskonventionen ist es leichter, z. B. Anhand eines Quelltextes herauszufinden, wie das verwendete BAdI heißt. SAP schlägt folgende Benennungen vor:

- BAdI-Definition: Zusammengesetzt aus Namenspräfix (z. B. Z) und dem eigentlichen Namen, der durch den SAP- oder Partnerentwickler, der das BAdI anlegt, frei wählbar ist
- BAdI-Interface: Zusammengesetzt aus Namenspräfix (z. B. Z), IF\_EX\_ und dem BAdI-Namen
- Methoden: Durch den SAP- oder Partnerentwickler, der das BAdI anlegt, frei wählbar
- BAdI-Klasse: Wird bei der Erstellung des BAdI durch den SAP- oder Partnerentwickler automatisch vom System benannt, Name setzt sich aus Namenspräfix, CL\_EX\_ und BAdI-Namen zusammen
- Implementierende Klasse: Wie soeben besprochen frei wählbar, sollte aber aus Namenspräfix, CL\_IM\_ und dem Namen der Implementierung bestehen und wird vom System so vorgeschlagen.

In der implementierenden Klasse können Sie auch private Methoden definieren, die dann von den Interfacemethoden die Sie implementieren verwendet werden können. So kann Ihr Code besser strukturiert werden. Die Definition der privaten Methoden geschieht wie gewohnt im Class Builder.

Wenn die Implementierungen fertig und aktiviert sind, kann auch die gesamte BAdI-Implementierung aktiviert werden. Dies geschieht aus dem BAdI-BUILDER (**SE19**) über die entsprechende Schaltfläche. Durch die komplementäre Schaltfläche zum Deaktivieren kann die Aktivierung wieder zurückgenommen werden.



Abbildung 57: Aktivieren / Deaktivieren von BAdI-Implementierungen: SAP-System-Screenshot

Gibt es keine aktive Implementierung zu einem BAdI, wird im SAP-Programm dennoch die Methode `get_instance` der Klasse `cl_exithandler` aufgerufen, dasselbe gilt auch für die Instanzmethode der BAdI-Klasse beim Aufruf eines BAdI. Die BAdI-Klasse stellt dann fest, dass es keine aktive Implementierung gibt und das Programm fährt fort. Dieser Ablauf stellt einen Unterschied zu den Programm-Exits aus dem Customer-Exit-Abschnitt dar: Der Aufruf `CALL CUSTOMER-FUNCTION` wird bei einer fehlenden Implementierung nicht ausgeführt.

Beachten Sie, dass die Aktivierung oder Deaktivierung nur im Originalsystem der Implementierung modifikationsfrei durchgeführt werden kann. In die nachgelagerten Systeme (z. B. Test- und Produktivsystem) muss diese Eigenschaft transportiert werden.

### 6.3.2 Praxis: Implementierung eines BAdI

Öffnen Sie den Object Navigator und öffnen Sie darin das vordefinierte Programm **ZZ\_ABAP2\_BADI####**, indem Sie in den Feldern oberhalb des Navigationsbaums

Programm aus der Drop-Down-Box auswählen, den Namen des Programms in das Feld darunter eingeben und mit Enter bestätigen.



Abbildung 58: Öffnen des Programms: SAP-System-Screenshot

Testen Sie das Programm. Es handelt sich um einen einfachen Report, um zu einer Flugverbindung vorhandene Buchungen anzeigen zu lassen.

Geben Sie Beispieldaten ein und schauen Sie sich die Ausgaben des Programms an. Von den Anwendern des Programms wird der Wunsch geäußert, auch den vollen Namen des Fluggasts und nicht nur seine Kundennummer angezeigt zu bekommen.

Finden Sie heraus, ob das Programm Erweiterungsmöglichkeiten bietet, mit der die Kundenwünsche umgesetzt werden können, indem Sie den Quelltext nach der Zeichenkette **cl\_exithandler** durchsuchen.

*Hinweis: Das Programm liegt zwar im Übungssystem im Kundennamensraum, es sei für diese Übung jedoch angenommen, dass es sich nicht um ein Kundenprogramm handelt, so dass ein direktes Bearbeiten ausscheidet. Verändern Sie also in dieser Übung **nicht** den Quelltext des Programms!*

Klicken Sie doppelt auf die Referenzvariable, der durch die Methode des **cl\_exithandler** die Instanz der BAdI-Klasse zugewiesen wird. Sie gelangen dadurch zu deren Definition. Klicken Sie dort doppelt auf das BAdI-Interface, mit dem die Referenzvariable typisiert ist. Sie gelangen so zur Definition des BAdI-Interfaces.

Klicken Sie dort oberhalb der Registerkartenreiter auf den **Namen des Interfaces** und führen Sie einen Verwendungsnachweis in Klassen durch. Sie sollten ein Ergebnis erhalten, dem Sie den Klassennamen entnehmen können. Gemäß der in diesem Kapitel erläuterten **Namenskonvention** können Sie aus dem Klassennamen den Namen des BAdI ermitteln. Sollte der Verwendungsnachweis nichts anzeigen, was leider vorkommen kann, verwenden Sie die Lösung in der Fußnote<sup>1</sup>. Öffnen Sie die Transaktion **SE18** und lassen Sie sich dort die Dokumentation zum BAdI anzeigen, falls diese existiert (dies ist im Schulungssystem nicht der Fall, wäre aber normalerweise zu erwarten).

Öffnen Sie dann die Transaktion **SE19**, und legen Sie eine Implementierung des BAdI an, indem Sie im unteren Bereich (*Implementierung Anlegen*) den Punkt **Klassisches BAdI** auswählen, und in das Feld den **BAdI-Namen** angeben sowie den Button darunter betätigen. Anschließend werden Sie nach einem Namen der Implementierung gefragt. Benennen Sie diese **ZZ\_#####\_IMPL**, pflegen Sie eine passende Kurzbeschreibung und sichern Sie. Bestätigen Sie die Nachfragen nach Paket und Transportauftrag wie gewohnt.

<sup>1</sup> Das gesuchte BAdI heißt ZBADI#### (mit ihrer entsprechenden Benutzernummer)

Wechseln Sie nun zur Registerkarte **Interface**. Ein Doppelklick auf eine der Methoden führt Sie zum Class Builder und dort zum Code der Methode Ihrer BAdI-Implementierungsklasse. Implementieren Sie das BAdI: Schreiben Sie den benötigten Code der Methode und speichern und aktivieren Sie diese. Prüfen Sie in einem zweiten Modus, ob sich das Programm schon verändert hat. Dies sollte nicht der Fall sein, da die BAdI-Implementierung noch aktiviert werden muss. Kehren Sie daher aus der Methodendefinition zurück zur BAdI-Implementierung und klicken Sie auf die Schaltfläche zur Aktivierung. Testen Sie nun erneut das Programm. Neben allen Kundennummern sollten nun jeweils auch die entsprechenden Namen stehen.

### 6.3.3 Menü-Exits mit BAdIs

Auch Menü-Exits sind über BAdIs möglich, sofern der SAP- bzw. Partner-Anwendungsentwickler eine solche Möglichkeit bei der Erstellung der Anwendung vorgesehen hat.

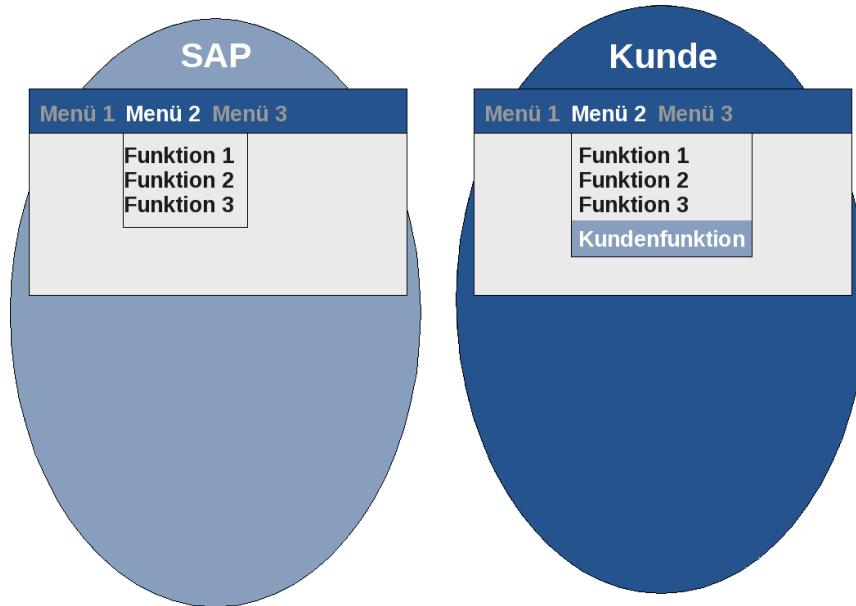


Abbildung 59: Menü-Exit mit BAdI

Die Erweiterung muss – wie bei der Technik mit Customer-Exits – auch hier vom Entwickler vorgedacht sein, indem er einen Menüpunkt für den Kunden reserviert. Dieser Menüpunkt erhält auch hier einen mit dem Pluszeichen beginnenden Funktionscode.

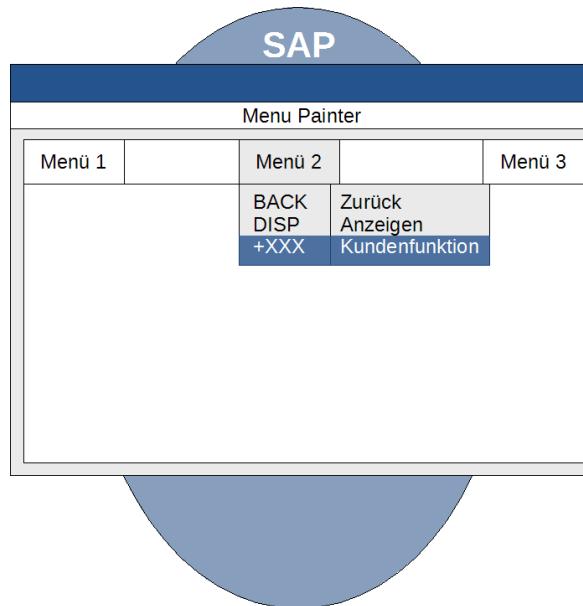


Abbildung 60: Für Kundenfunktion reservierter Funktionscode im Menü

Im Code des SAP-Entwicklers wird beim Eintreten des Funktionscodes eine Methode des BAdI gerufen. Dieser Methodenaufruf ist unbedingt notwendig, um eine Funktionalität für den Menüpunkt hinterlegen zu können. Der Menüpunkt wird dann sichtbar, wenn es zu der

BAdI-Methode eine entsprechende, aktive Implementierung gibt. Existiert keine solche Implementierung, ist der Menüpunkt ausgebendet.

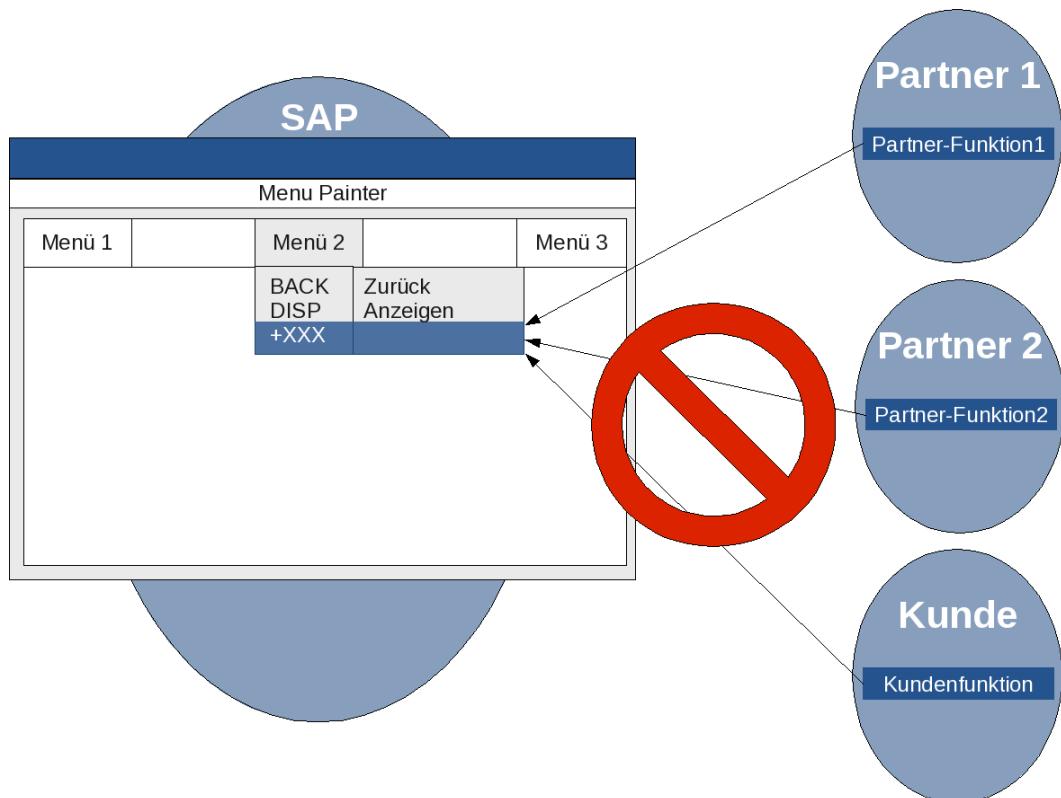


Abbildung 61: Mehrfachverwendung?

Ein BAdI, das Funktionscodes zur Erweiterung eines Menüs enthält, muss als nicht mehrfach-verwendbar gekennzeichnet werden. Das hängt damit zusammen, dass sonst unklar wäre, welche Menüfunktion hinter dem Menüpunkt steckt. Auch eine Filterabhängigkeit ist an dieser Stelle nicht zulässig. Durch Filter kann im Allgemeinen die Ausführung des BAdI an Bedingungen geknüpft werden, die durch die BAdI-Klasse geprüft werden. Diese ruft dann nur die Implementierungen auf, die die Bedingungen erfüllen.

Für die Filterung wird ein Datenelement verwendet, das eine Wertetabelle besitzt. Die Angabe des Filterwertes wird dann ein zusätzliches, obligatorisches Argument der jeweiligen Methode.

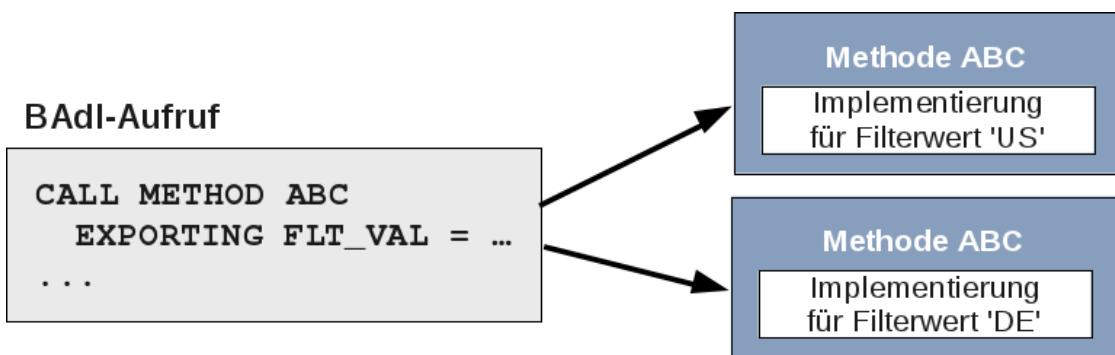


Abbildung 62: Filterabhängiger BAdI-Aufruf

## 6.4 Modifikationen

Bisher haben Sie mit den Customer-Exits und den klassischen Business Add Ins zwei Erweiterungstechniken kennen gelernt, mit denen das SAP-System durch Kunden bzw. Partner angepasst werden konnte. Diese Erweiterungstechniken besitzen die Gemeinsamkeit, dass der Ersteller der jeweiligen Software, in der Regel also der Anwendungsentwickler auf Seiten von SAP, die Erweiterung vorgedacht hat. Der SAP-Anwendungsentwickler muss diese Erweiterungsmöglichkeiten also explizit in seinem Programm anlegen.

Ein Vorteil dieser Techniken ist die Kontrolle: Es sind nur an den festgelegten Stellen Anpassungen möglich, weitere Teile des Programms bleiben stets unverändert. Darüber hinaus existiert eine klar definierte Schnittstelle zwischen dem Originalprogramm und den Anpassungen, die auch bei Updates des Originalprogramms stabil gehalten werden kann, so dass die Anpassungen des Programms ohne weiteres bestehen bleiben können.

Der Vorteil der Kontrolle wandelt sich in einen Nachteil, wenn der Kunde eine Anpassung am Programm vornehmen möchte, die vom SAP-Entwickler nicht vorgesehen wurde. Der Kunde ist bei den Erweiterungskonzepten der Customer-Exits und klassischen BAdIs also auf die Weitsicht des SAP-Entwicklers angewiesen.

In diesem Unterkapitel werden Ihnen **Modifikationen** vorgestellt. Durch diese ist es möglich, auch außerhalb von vorgegebenen Erweiterungsmöglichkeiten Anpassungen des Systems vorzunehmen, indem Elemente außerhalb des Kundennamensraums bearbeitet werden. Damit es durch solche Anpassungen nicht zum Chaos kommt, sind einige Dinge zu beachten, die im Folgenden thematisiert werden.

### 6.4.1 Grundbegriffe

Die Entwicklung von Software für das SAP ERP-System spielt sich im Normalfall unter Verwendung mehrerer Systeme ab. Eine typische Konfiguration besteht etwa aus einem Entwicklungs-, einem Test- und einem Produktivsystem. Hinzu kommen die fremden Systeme in der Auslieferungskette der Software, so natürlich auch ein System das SAP selbst verwendet.

Alle Entwicklungsobjekte liegen in jeweils genau einem System als **Original** vor. In allen anderen Systemen, in die sie transportiert werden, gelten sie hingegen als **Kopie**.

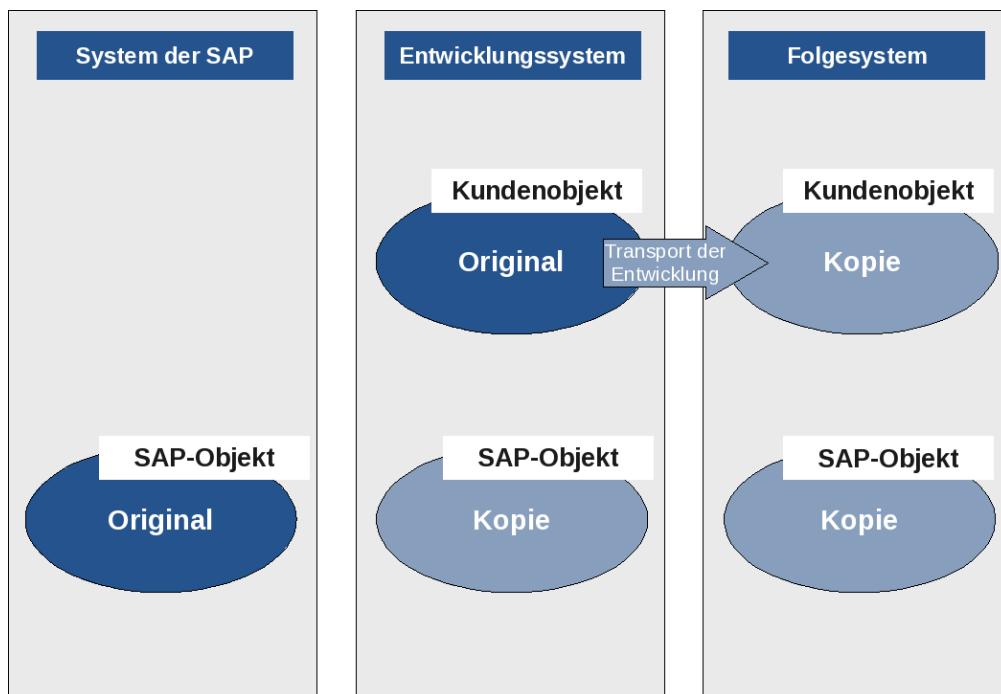


Abbildung 63: Original vs. Kopie

Die SAP-Objekte liegen in den Systemen des Kunden somit als Kopie vor, während die Eigenentwicklungen des Kunden auf dessen Entwicklungssystem als Original vorliegen. Die in der Transportkette folgenden Systeme des Kunden enthalten dann sowohl von den eigenen Entwicklungsobjekten als auch von den SAP-Entwicklungsobjekten Kopien.

Sowohl das Original als auch die Kopie eines Entwicklungsobjekts können bearbeitet werden. Bei Änderungen des Originals spricht man von einer **Korrektur**, dementsprechend werden diese in Transportaufträgen mit Aufgaben des Typs **Entwicklung/Korrektur** festgehalten. Wird hingegen eine Kopie bearbeitet, handelt es sich um eine **Reparatur**. Die in diesem Kursteil behandelten **Modifikationen** sind Reparaturen von SAP-Objekten. Sie können auch als Kunde Reparaturen an Ihren eigenen Objekten vornehmen. Diese können etwa dann erforderlich werden, wenn sich auf dem Produktivsystem Fehler finden die kurzfristig dort behoben werden müssen. Diese Änderungen müssen anschließend auch im Original des Entwicklungsobjekts durchgeführt werden, da es ansonsten bei weiteren Entwicklungen zu Problemen kommen kann. Vor dem Einspielen eines Upgrades oder Support Packages müssen alle Aufträge, die Reparaturen enthalten, freigegeben werden.

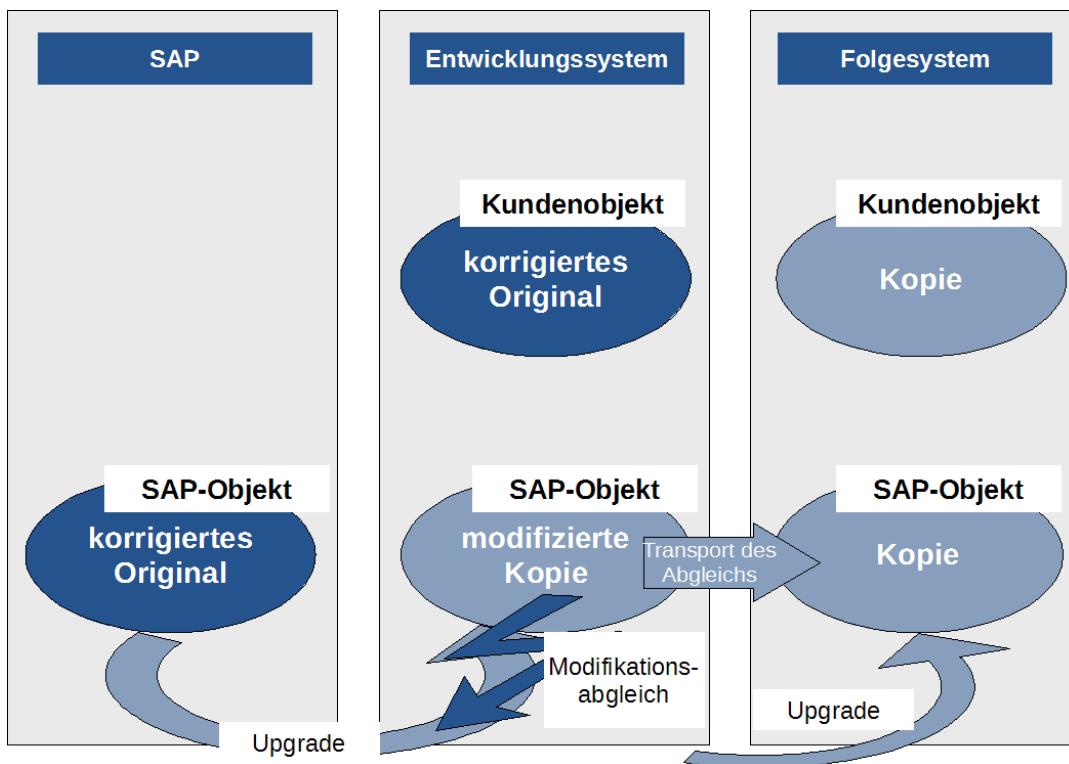


Abbildung 64: Notwendigkeit des Modifikationsabgleichs

Wenn Sie per Modifikation Änderungen an einem SAP-Objekt durchführen, kommt es zu einem Konflikt, wenn SAP dieses Objekt verändert und neu ausliefert, etwa in Form eines Upgrades oder Support Packages. Ihre Modifikation wurde an der alten Kopie vorgenommen, durch die Aktualisierung seitens SAP befindet sich in Ihrem System nun aber die neue Kopie, die Ihre Änderungen nicht enthält, und wird zum aktiven Objekt Ihres Systems. Ihre Änderungen müssen dann auf diese neue Version übertragen werden, wenn Sie weiterhin gelten sollen. Dieser Vorgang wird als **Modifikationsabgleich** bezeichnet.

Wie in der obigen Abbildung dargestellt wird der Modifikationsabgleich auf dem Entwicklungssystem durchgeführt, und anschließend auf die folgenden Systeme transportiert. Theoretisch wäre auch ein Modifikationsabgleich auf einem Produktivsystem möglich, davon wird jedoch abgeraten. Je nach Umfang der von Ihnen vorgenommenen Reparaturen kann der Modifikationsabgleich einen sehr hohen Aufwand darstellen. Dies spricht dafür, Anpassungen des SAP-Systems möglichst **modifikationsfrei** unter Verwendung einer der bereitgestellten Erweiterungstechniken durchzuführen. Zur Modifikation sollte nur gegriffen werden, wenn die modifikationsfreie Erweiterung nicht infrage kommt.

Modifikationen werden von SAP im Rahmen der SAP Software Change Registration (SSCR) erfasst. Um ein SAP-Objekt modifizieren zu können, wird ein Zugangsschlüssel (**SSCR-Schlüssel**, auch **Modifikationsschlüssel** genannt) benötigt. Dieser wird bei SAP beantragt. Möglicherweise sind Sie schon einmal zur Eingabe eines solchen Schlüssels aufgefordert worden, weil Sie versehentlich versucht haben, ein SAP-Objekt zu bearbeiten:



Abbildung 65: Abfrage des Modifikationsschlüssels

Versehentliche Änderungen an SAP-Objekten werden durch diese Abfrage vermieden, und der Aufwand der Registrierung sorgt in gewissem Maße auch dafür, dass ein Entwickler beim Kunden eher eine modifikationsfreie Technik zur Anpassung des Systems verwendet. So werden weniger Modifikationen vorgenommen und die Probleme im Rahmen des Modifikationsabgleichs verringert. Weiterhin liegen SAP durch die Registrierung der Modifikationen Informationen vor, welche Kunden welches SAP-Objekt angepasst haben. Diese Information kann vom Support genutzt werden: Meldet sich der Kunde mit einem Problem, kann SAP einfacher abschätzen, ob das Problem mit einer Anpassung zusammenhängen könnte.

Die Modifikation wird wie auch die herkömmlichen Entwicklungsarbeiten in einem Transportauftrag und darin in einer Aufgabe organisiert. Modifikationen werden in speziellen Aufgaben des Typs Reparatur verwaltet. Der Auftrag sorgt dafür, dass das bearbeitete Objekt gegen **Änderungen** und **Importe** gesperrt wird. So kann es weder von anderen Anwendern des Systems geändert werden, noch durch den Import einer neuen Version von SAP überschrieben werden. Bei Freigabe der Aufgabe muss der Entwickler seine Änderungen dokumentieren und die Sperren gehen an den Auftrag über und werden bei dessen Freigabe ganz aufgehoben. Bestätigt der Entwickler die Modifikation bei der Aufgabenfreigabe nicht, bleibt die Importsperre bestehen, sie kann nur durch den Entwickler aufgehoben werden. Die Auftragsfreigabe führt zu einem Kopiervorgang der Objekte von der Datenbank in ein Verzeichnis des zugrundeliegenden Betriebssystems, von wo aus ein Administrator sie in das Zielsystem importiert.

Für die Durchführung des Modifikationsabgleichs ist es erforderlich, eine Versionierung durchzuführen. Bei der Freigabe des Auftrags wird automatisch eine Vollversion aller Objekte in der Versionsdatenbank hinterlegt. Werden weitere Änderungen durchgeführt, hinterlegt das System bei Freigabe des nächsten Auftrags zusätzlich die Differenz, also die Unterschiede zur Vorversion. Wird ein SAP-Objekt in eine Aufgabe eingefügt, wird dessen

Version aus der Vollkopie in der Versionsdatenbank geladen, sofern bereits eine Vollkopie existiert.

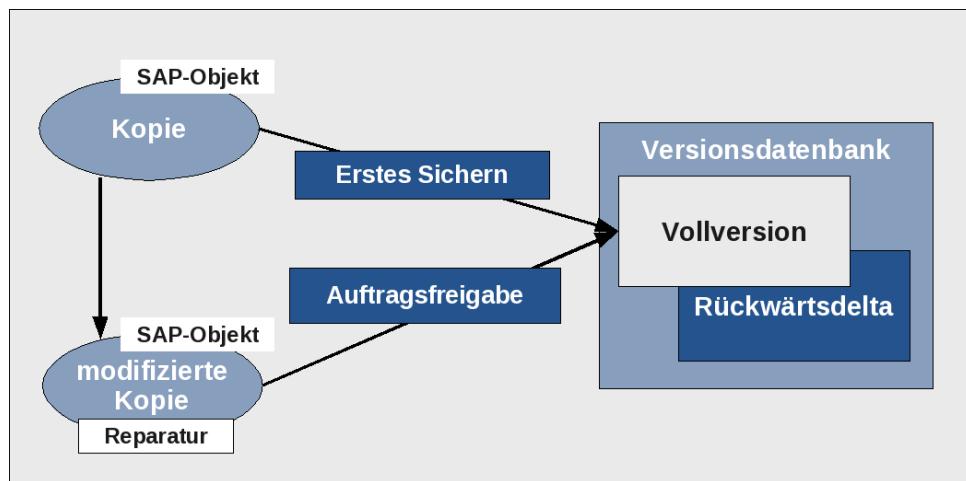


Abbildung 66: Versionsverwaltung beim modifizierten von SAP-Objekten

#### 6.4.2 Der Modifikationsassistent

Bei der Durchführung von Modifikationen steht Ihnen der **Modifikationsassistent** als unterstützendes Hilfsmittel zur Verfügung. Er arbeitet dabei mit den wesentlichen Werkzeugen zusammen, mit denen Modifikationen realisiert werden: Den Werkzeugen der ABAP Workbench. Im Detail sind das der ABAP Editor, für den ein spezieller Modus für Modifikationen angeboten wird, der Screen Painter, der Menu Painter, die Pflege von Textelementen, der Class Builder, der Function Builder, das ABAP Dictionary und die Pflege von Dokumentationen.

Der Zweck des Modifikationsassistenten ist die Vereinfachung des Modifikationsabgleichs. Um diesem Zweck gerecht zu werden, ermöglicht der Assistent eine feinere Granularität bei der Aufzeichnung von Änderungen. Vor der Einführung des Assistenten war es nicht möglich, auf einer feineren Ebene als der von Includes zu arbeiten, deren Änderungen dann beim Abgleich der Modifikation zeilenweise mit wenig Systemunterstützung in die neue SAP-Version übertragen wurden. Der Modifikationsassistent hingegen erlaubt die Aufzeichnung auf Ebene einzelner Programmbausteine wie Module, Routinen und Funktionsbausteine aufzuzeichnen und dann auch auf dieser Ebene den Abgleich durchzuführen.

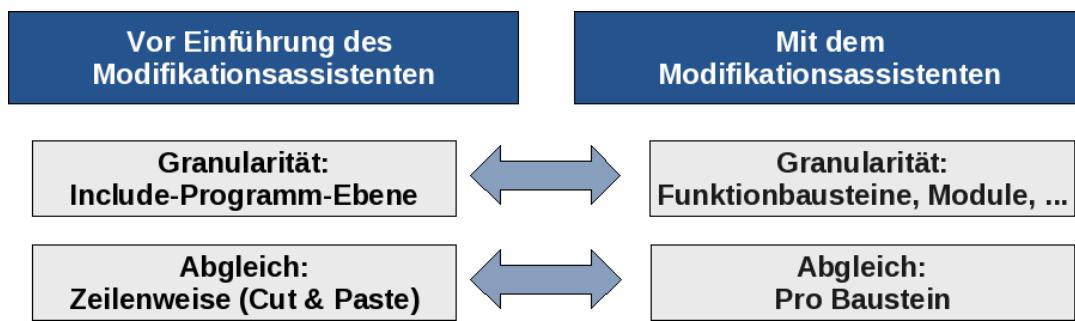


Abbildung 67: Vorteile des Modifikationsassistenten

Die feinere Granularität macht in vielen Fällen einen Modifikationsabgleich überflüssig. Eine Änderung der SAP an einem SAP-eigenen Include machte bislang einen Abgleich des Moduls erforderlich, sobald durch den Kunden irgendeine Modifikation an diesem Include vorgenommen wurde. Mit dem Modifikationsassistenten wird hingegen berücksichtigt, dass der Programmbaustein, in dem SAP eine Änderung durchgeführt hat, gar nicht von der Modifikation des Kunden betroffen ist. In diesem Fall ist kein Modifikationsabgleich mehr nötig: Der entsprechende Baustein wird von SAP übernommen, während die vom Kunden modifizierten Teile des Includes unverändert bleiben.

Der Modifikationsassistent zeichnet die Änderungen zunächst auf einer speziellen Ebene auf, ohne dass dabei die ursprünglichen Entwicklungsobjekte verändert werden. Erst wenn die Generierung angestoßen wird, werden die Änderungen übernommen und ein ausführbares Programm erzeugt, das aus dem ursprünglichen Code und den aufgezeichneten Änderungen besteht.

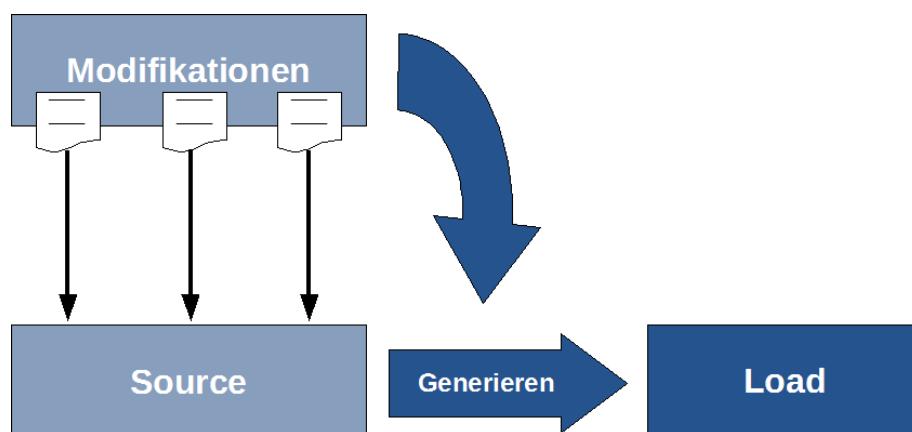


Abbildung 68: Funktionsweise des Modifikationsassistenten

Entlang der Auslieferungskette der Software können mehrere Modifikationsschichten auftreten: Die von SAP ausgelieferte Software wird bspw. durch einen Partner modifiziert. Diese modifizierte Software stellt für den nachfolgenden Softwarenutzer das Original dar und kann von ihm in einer weiteren Schicht modifiziert werden.

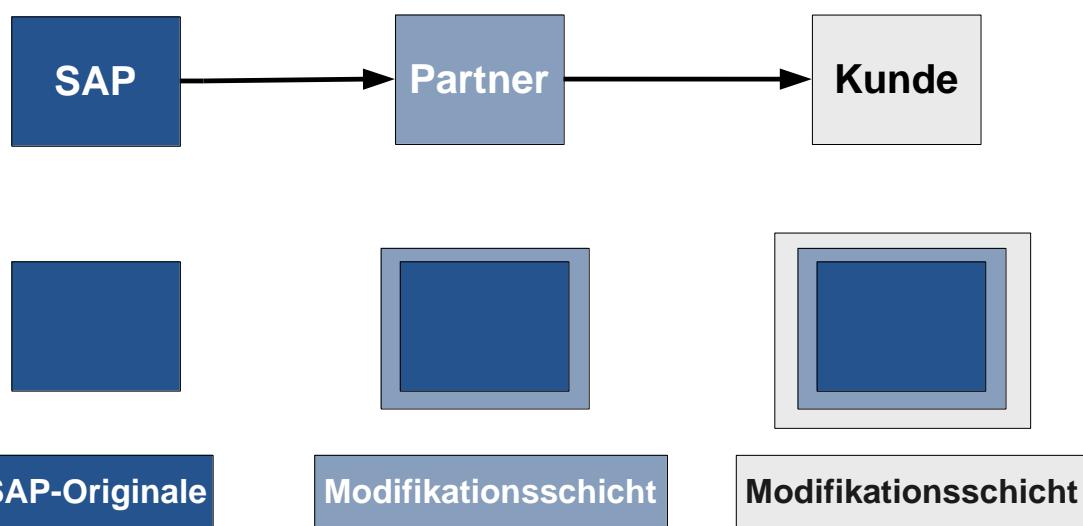


Abbildung 69: Modifikationen entlang der Software-Auslieferungskette

Beim Bearbeiten von Quelltexten in der ABAP Workbench stellt der Modifikationsassistent einen speziellen Änderungsmodus bereit. Dieser schränkt die Funktionalität des Editors deutlich ein, um die Änderungen detailliert protokollieren zu können. Es werden Schaltflächen zum Einfügen, Ersetzen und Löschen von Quellcode sowie zum Zurücknehmen dieser Änderungen angeboten, außerdem kann eine Modifikationsübersicht aufgerufen werden, die die durchgeführten Änderungen auflistet. Eine zurückgenommene Änderung geht verloren und kann nicht wieder sichtbar gemacht werden.

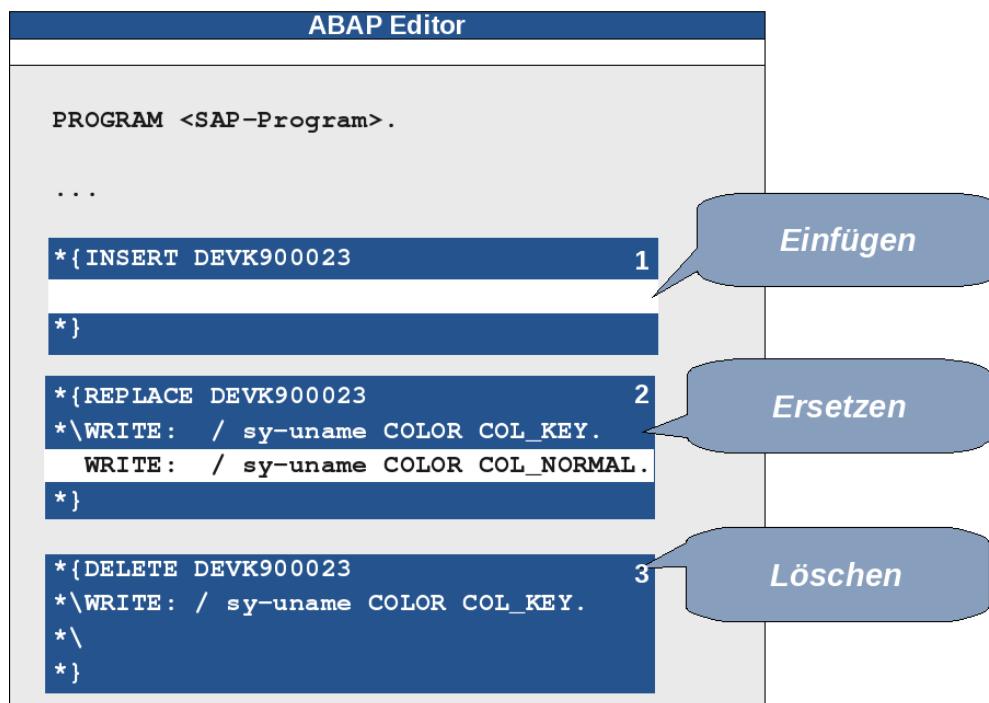


Abbildung 70: Änderungen mit dem Modifikationsassistenten

Die Änderungen, die über die Schaltflächen des Modifikationsassistenten durchgeführt werden, werden im Quelltext automatisch mit Kommentaren versehen. Diese geben an, ob es sich um einen Einfüge-, Ersetzungs- oder Löschvorgang handelt, und enthalten ggf. den vorher vorhandenen Code. Weiterhin sind der Änderungsauftrag, innerhalb dessen die Änderung durchgeführt wurde, und eine Nummer zur internen Verwaltung sichtbar.

Die Modifikationsübersicht listet alle Modifikationen innerhalb des Programms auf. Die Gliederung erfolgt hier anhand der Modularisierungseinheiten des Programms, also der dem Modifikationsassistenten zugrundeliegenden Granularität entsprechend.

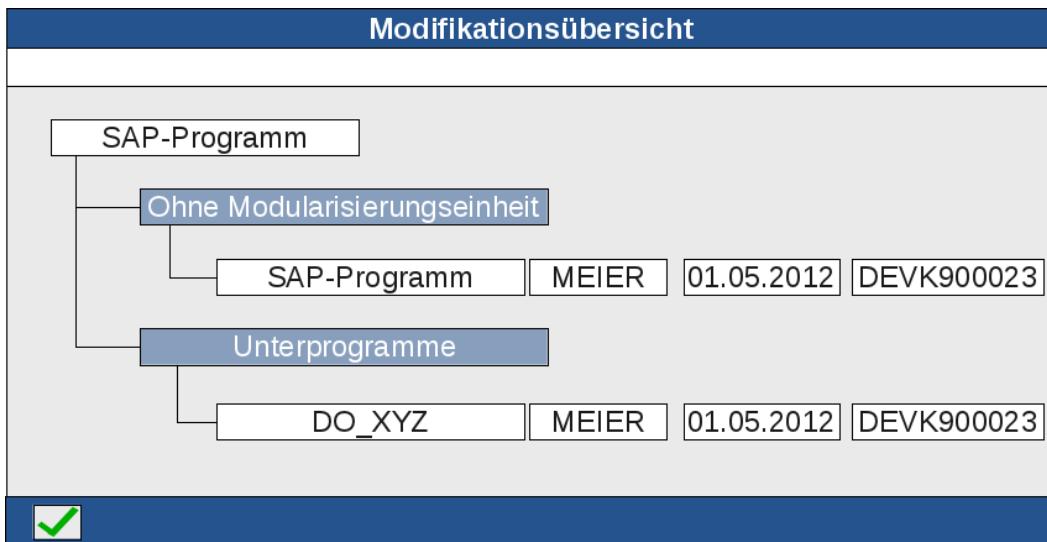


Abbildung 71: Modifikationsübersicht

Der Modifikationsassistent kann Systemweit ein- oder ausgeschaltet werden. Dafür wird der Profilparameter **eu/controlled\_modification** verwendet. Darüber hinaus kann er für einzelne Repository-Objekte deaktiviert werden. Die Verwendung des Modifikationsassistenten wird empfohlen.

Die Darstellungen zum Bearbeiten von Code bedeuten nicht, dass der Modifikationsassistent nur die Bearbeitung von Quelltexten unterstützt. Vielmehr bietet er eine Unterstützung in verschiedenen Entwicklungswerkzeugen und kann so z. B. auch die Modifikation von Dictionary-Elementen unterstützen.

#### 6.4.3 Der Modification Browser

Um systemweit modifizierte Objekte zu finden, stellt SAP den **Modification Browser** zur Verfügung. Sie können diesen über den Menüpfad **Umfeld -> Modification Browser** aus der ABAP Workbench aufrufen, oder den Transaktionscode **SE95** benutzen.

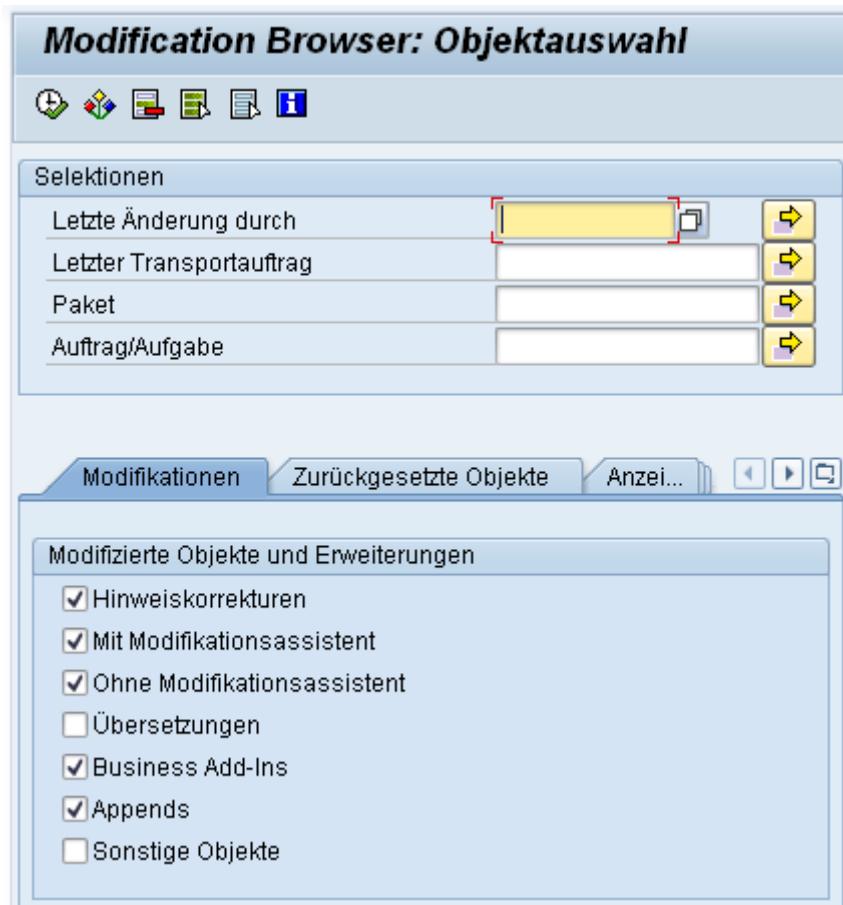


Abbildung 72: Einstiegsbild des Modification Browsers: SAP-System-Screenshot

Das Einstiegsbild des Modification Browsers erlaubt die Filterung nach verschiedenen Kriterien, so dass gezielt Modifikationen in bestimmten Bereichen ausgewählt werden können. Die Ergebnisse werden dann in Form eines Baums angezeigt:



Abbildung 73: Ergebnisanzeige des Modification Browsers: SAP-System-Screenshot

Die Funktionalität des Modification Browsers ist nicht auf die reine Anzeige beschränkt. Aus der Ergebnisdarstellung heraus können ganze Teilbäume zurückgenommen werden, indem diese markiert und die Schaltfläche  (Rücksetzen auf Original) verwendet wird.

#### 6.4.4 Der Note Assistant

Ein weiteres Hilfsmittel im Bereich der Modifikationen ist der Note Assistant. Dieser findet sich unter dem Transaktionscode **SNOTE** und widmet sich den sogenannten **SAP-Hinweisen**. Ein SAP-Hinweis ist eine Beschreibung von Fehlern im SAP-System, die die Symptome des Fehlers, die Ursachen, und die betroffenen Releasestände enthalten. Je nach Art des Fehlers ist im Hinweis auch beschrieben, wie dieser durch Workarounds umgangen, durch Korrekturen beseitigt oder durch das Einspielen von Support Packages behoben werden kann. Support Packages werden durch den Note Assistant jedoch nicht ersetzt.

Vor Einführung des Note Assistant konnten SAP-Hinweise, die eine Korrektur an Objekten des SAP-Systems erforderten, lediglich manuell umgesetzt werden. Dies erforderte die Eingabe eines Schlüssels für das betroffene Objekt und war sehr anfällig für Fehler. Der Kunde musste selbstständig aus der Beschreibung des SAP-Hinweises ermitteln, welche Wechselwirkung die Fehlerkorrektur mit anderen Fehlerkorrekturen im System hat, und beim späteren Einspielen eines Support Packages, das die Behebung des Problems umfasst, war ein manueller Abgleich erforderlich.

Der Note Assistant unterstützt den Anwender durch Automatisierungen bei der Handhabung von SAP-Hinweisen. Zunächst können die Hinweise mithilfe des Note Assistant direkt vom SAP Service Marktplatz heruntergeladen werden. Dann können die Hinweise automatisiert mithilfe des Modifikationsassistenten in das System eingespielt werden, ohne dass dafür die Eingabe des entsprechenden Schlüssels erforderlich ist. Diese Möglichkeit ist allerdings auf Korrekturen am ABAP-Quelltext beschränkt. Die Abhängigkeiten der Hinweise untereinander werden vom Note Assistant erkannt und berücksichtigt, indem auf abhängige Hinweise aufmerksam gemacht wird und der Anwender auch zu deren Einspielung aufgefordert wird.

Mit dem Note Assistant können Hinweise betrachtet werden. Dabei wird angezeigt, welche Hinweise obsolet sind, weil sie inzwischen in einem Support Package berücksichtigt wurden. Obsolete Hinweise werden auch beim Modifikationsabgleich berücksichtigt. Hier werden Modifikationen, die zu Hinweisen gehören, besonders hervorgehoben und die Korrekturen von obsolet gewordenen Hinweisen hervorgehoben. So können die betroffenen Entwicklungsobjekte wieder in die Standardversion überführt werden.

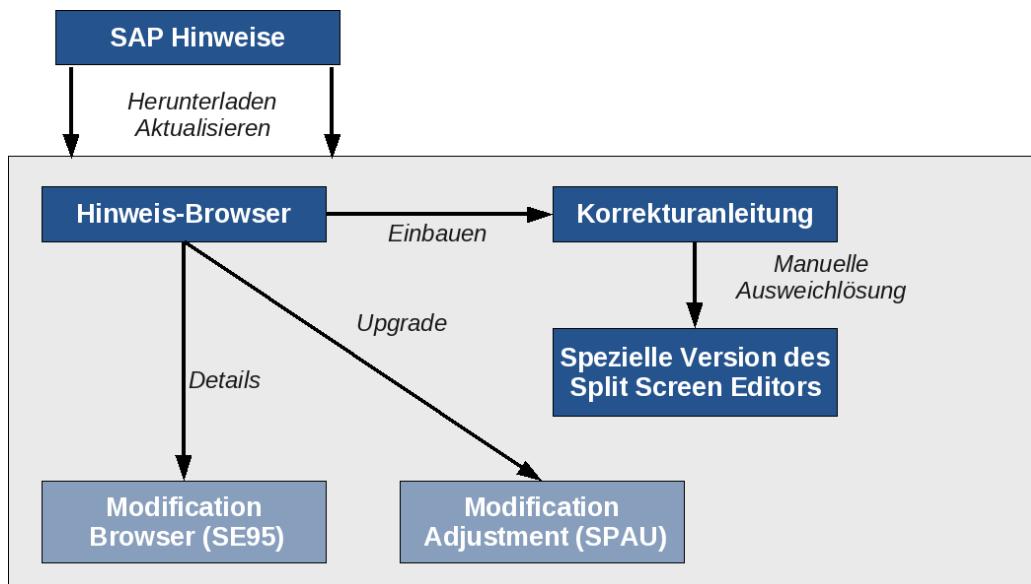


Abbildung 74: Übersicht über den Note Assistant

Der Note Assistant zeigt nach dem Aufruf über die Transaktion **SNOTE** eine Liste von Hinweisen, die sich in die Teile **Inkonsistent**, **Neu** und **In Bearbeitung** gliedert. Bei ersterem handelt es sich um die Liste der Hinweise, die sich in einem inkonsistenten Zustand befinden. Dies ist etwa der Fall, wenn eine veraltete Version des Hinweises eingebaut ist, oder wenn der Hinweis nicht vollständig eingebaut wurde. Hinweise in Bearbeitung sind die Hinweise, die Ihnen als Bearbeiter zugeordnet wurden.

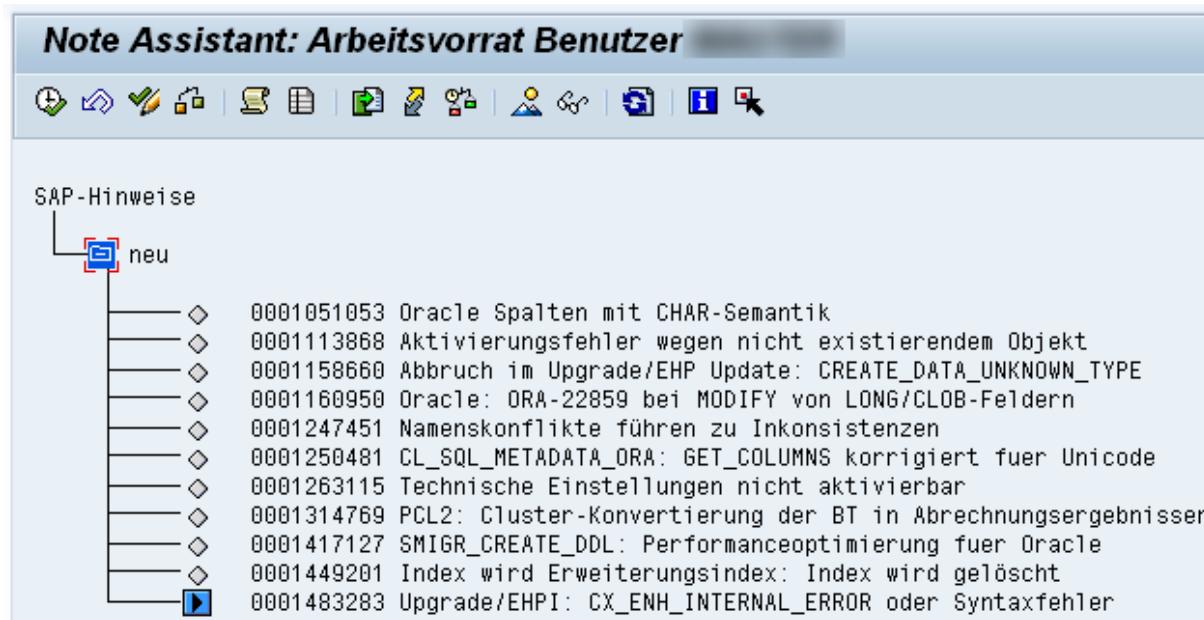


Abbildung 75: Neue Hinweise im Note Assistant: SAP-System-Screenshot

Über die Symbole vor den Hinweisen wird dargestellt, ob der Hinweis automatisch eingespielt werden kann. Wählen Sie den Menüpfad **Hilfsmittel -> Farblegende** um sich anzeigen zu lassen, welche Bedeutung die einzelnen Symbole besitzen.

Zum Laden von Hinweisen gibt es zwei Möglichkeiten:

- **Hinweis-Download:** Hierbei wird der Hinweis direkt über den Note Assistant aus dem SAP-Service-Marktplatz geladen.
- **Hinweis-Upload:** Hierbei wird der Hinweis zunächst auf das lokale Dateisystem heruntergeladen, um anschließend von dort in das SAP-System hochgeladen zu werden.

Beide Möglichkeiten werden vom Einstiegsbild des Note Assistant als Untermenüpunkte von **Springen** aufgerufen. In demselben Menü finden Sie auch den **Hinweis-Browser**. Dieser bietet die Möglichkeit, nach bestimmten Hinweisen zu suchen. Wenn Sie auf dem Einstiegsbild des Note Assistant oder in den Suchergebnissen des Hinweis-Browsers doppelt auf einen Hinweis klicken, wird Ihnen das entsprechende Dokument angezeigt. Vor dem Einbau eines Hinweises empfiehlt es sich, dieses Dokument sorgfältig zu lesen.

Wird der Einbau vom Benutzer angestoßen, prüft der Note Assistant die Abhängigkeiten und lädt ggf. vorausgesetzte Hinweise. Die vom Benutzer ausgewählten und vom System als abhängig bestimmten Hinweise werden dem Benutzer dann zur Installation angeboten. Dabei ist die Reihenfolge fest vorgegeben. Es gibt an dieser Stelle dennoch Wahlmöglichkeiten.

Es kann versucht werden, so viele Hinweise wie möglich durch den Note Assistant auf einmal einzubauen zu lassen. Wie viele Hinweise auf diese Weise auf einen Schlag eingebaut werden können, hängt davon ab, ob Hinweise einzubauen sind, die Includes betreffen, in denen bereits eigene Modifikationen vorgenommen wurden. Trifft der Note Assistant auf solche Hinweise, ist unter Umständen ein Abgleich erforderlich, so dass der Vorgang unterbrochen wird, um die Gelegenheit für den Abgleich der eigenen Modifikationen zu bieten.

Statt der Möglichkeit des Masseneinbaus können auch alle Hinweise einzeln eingebaut werden. Dies bietet sich dann an, wenn die am System vorgenommenen Änderungen Schritt für Schritt nachvollzogen werden sollen. Als dritte Option bietet sich an, den Einbau der Hinweise zu stoppen.

Eingangs wurde erwähnt, dass der Note Assistant bzw. dessen Verwendung zum Einspielen von Hinweisen keinen Ersatz für das Einspielen von Support Packages ist. In Support Packages werden verschiedene Fehlerkorrekturen gesammelt und können als ganzes in das System eingespielt werden. Diese Korrekturen werden über Hinweise dokumentiert. Wird ein Support Package eingespielt, müssen die bereits eingespielten Hinweise berücksichtigt werden.

- Bei Korrekturen aus dem Support Package, die bereits durch Hinweise im System eingebaut sind, müssen die enthaltenen Objekte auf das SAP-Original zurückgesetzt werden. Der Hinweis ist daraufhin als obsolet gekennzeichnet.
- Wenn Korrekturen, die im System durch Hinweise eingebaut sind, durch das Support Package überschrieben werden, ohne dass das Support Package diese Hinweise enthält, müssen die Hinweiskorrekturen erneut eingebaut werden. Die Hinweise tauchen im Note Assistant im Abschnitt **inkonsistent** auf und werden als erneut einzubauen gekennzeichnet.
- Es kann zudem vorkommen, dass Teile eines eingebauten Hinweises im Support Package enthalten sind. Das kann etwa dann vorkommen, wenn der Hinweis mehrere Korrekturanleitungen unterschiedlicher Gültigkeitsdauer enthält. Eine dieser Korrekturanleitungen könnte durch das Support Package obsolet werden, während die andere weiter gültig bleibt.

Wenn der Note Assistant erst nachträglich in ein System installiert wird, besitzt er keine Kenntnis über die bereits ohne dieses Werkzeug eingebauten Hinweiskorrekturen. Dadurch werden diese nicht im Hinweis-Browser dargestellt und Abhängigkeiten werden nicht korrekt behandelt, wenn ein benötigter Hinweis vom Note Assistant als fehlend und somit als noch einzubauen betrachtet wird, obwohl er eigentlich zuvor bereits im System eingebaut wurde. Die vor der Installation eingebauten Hinweise können dem Note Assistant jedoch nachträglich bekannt gemacht werden. Dafür steht das Programm SCWN\_REGISTER\_NOTES zur Verfügung. Dieses erwartet die Eingabe der Nummern von Hinweisen, die vollständig eingebaut sind. Das Programm prüft dann ob die Hinweise für das verwendete Systemrelease und Support Package Level gültig sind und registriert sie im Note Assistant, sofern sie dort nicht bereits als eingebaut registriert sind.

#### 6.4.5 User-Exits

Eine ursprünglich im Modul SD (Sales & Distribution) entwickelte Technik zur Erweiterung von SAP-Programmen sind **User-Exits**. Es handelt sich hierbei jedoch im Gegensatz zu Techniken wie BAdIs oder Customer-Exits nicht um eine modifikationsfreie Erweiterungstechnik. User-Exits bieten lediglich insoweit eine Entlastung für den Kunden, dass die Schwierigkeiten eines Modifikationsabgleichs umgangen werden.

Technisch realisiert werden User-Exits auf SAP-Seite durch spezielle Includes in Modulpools. Ein Modulpool ist ein Programm, das ähnlich wie eine Funktionsgruppe aus Includes zusammengesetzt ist. So gelten auch bei Modulpools bestimmte Endungen in den Includenamen als Zeichen, dass sich im Include ein bestimmter Inhalt befindet. Diese stimmen mit denen der Funktionsgruppen, die Ihnen im Kurs bereits vorgestellt wurden, überein. Die speziellen Includes für User-Exits unterscheiden sich technisch gesehen nicht von den anderen Includes. Insbesondere aber liegen Sie im SAP-Namensraum.

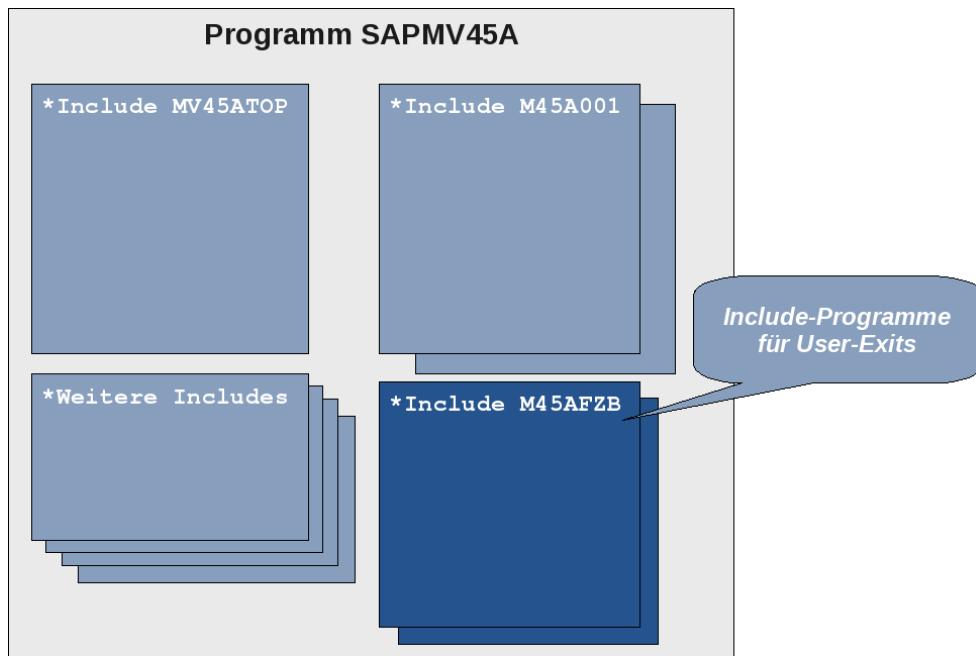


Abbildung 76: Aufbau eines Modulpools mit User-Exits

Die Umgehung des Aufwands beim Modifikationsabgleich wird dadurch erreicht, dass die Includes, in denen der User seinen Code implementieren soll, von SAP nur ein einziges Mal ausgeliefert werden. So kann es nicht passieren, dass eine veränderte Version der SAP auf eine veränderte Version des Kunden trifft.

Der Inhalt der Includes für User-Exits sind bei der Auslieferung durch SAP leere Unterprogramme (FORM...ENDFORM.) Diese werden dann vom Kunden durch dessen Code gefüllt. In den anderen Includes des Programms, in denen sich der SAP-Programmcode befindet, werden die Unterprogramme aufgerufen.

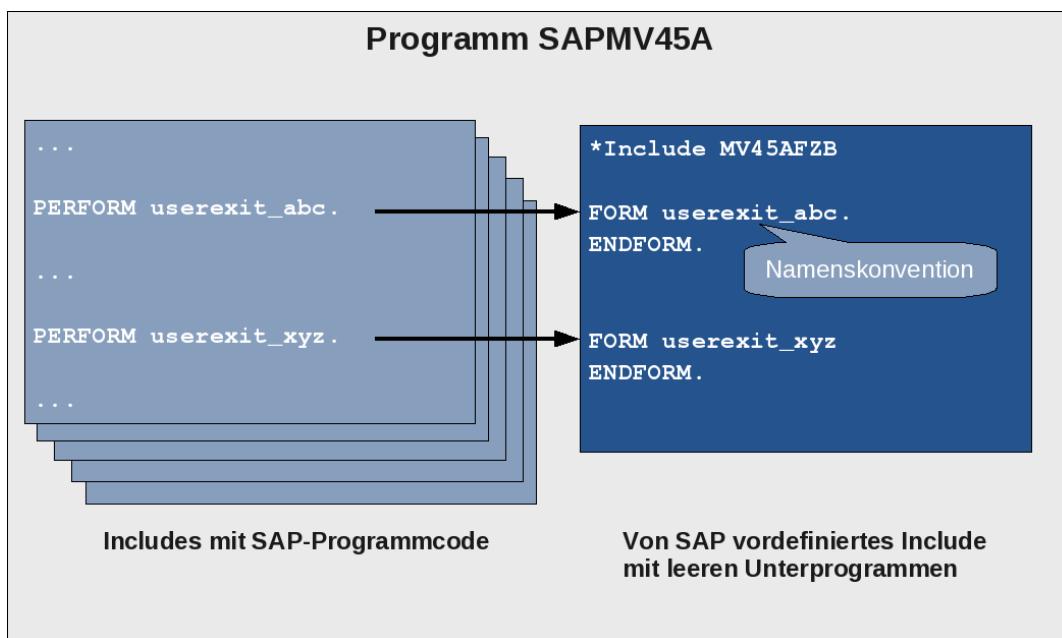


Abbildung 77: Technische Realisierung von User-Exits

Anhand von Namenskonventionen erkennt der Entwickler der SAP, dass ein Include vom Kunden für User-Exits verwendet wird und bearbeitet dieses nicht. Insbesondere dürfen neue User Exits nicht in dasselbe Include eingebaut werden, da dieses ja nur einmal ausgeliefert wird. Stattdessen würde in diesem Fall von SAP ein weiteres Include mit den entsprechenden Unterprogrammen angelegt und ausgeliefert, das ebenfalls bestimmten Namenskonventionen entspricht. Entscheidend sind dabei die letzten beiden Buchstaben des Includenamens.

Die Namen der Unterprogramme beginnen stets mit `userexit_`. Diese Namenskonvention kann für die programmlokale Suche nach User-Exits verwendet werden. Hierzu ist der Programmcode etwa nach der Zeichenkette `PERFORM userexit_` zu durchsuchen. Eine weitere Informationsquelle ist der SAP Referenz-Einführungsleitfaden.

#### 6.4.6 Modifikationen abgleichen

Bei Modifikationen ist ein Abgleich erforderlich, sobald SAP ein auf dem Kundensystem modifiziertes Objekt neu ausliefert. Objekte die vorher nicht im System vorhanden waren, oder Objekte die vom Kunden zwar modifiziert wurden, aber im eingespielten Upgrade oder Support Package von SAP nicht enthalten sind, müssen nicht abgeglichen werden.

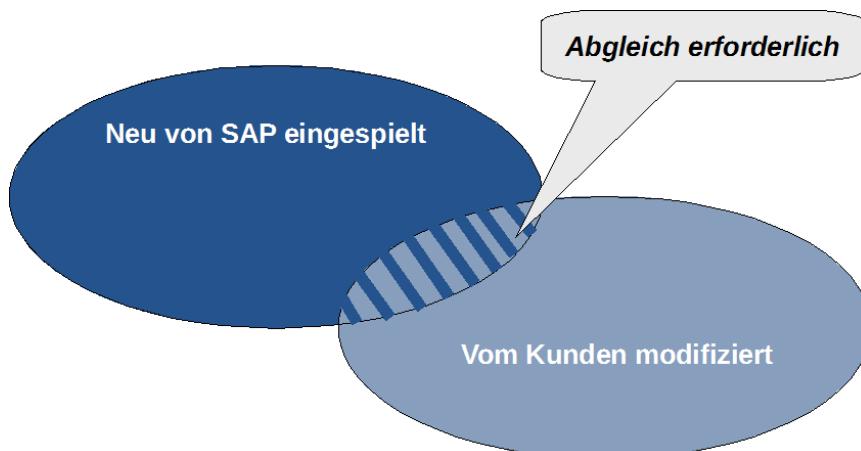


Abbildung 78: Schnittmenge, für die ein Abgleich erforderlich ist

Bei der Durchführung des Abgleichs werden die Transaktionen **SPDD** und **SPAU** verwendet. Diese unterscheiden sich nach ihrem Anwendungsgebiet:

- Die Transaktion SPDD führt den Modifikationsabgleich für Domänen, Datenelemente und Tabellen durch.
- Die Transaktion SPAU führt den Modifikationsabgleich für ABAP-Programmcode, Oberflächen (Menüs), Views, Sperrobjekte, Dynpros und Suchhilfen durch.

Der Abgleich mit SPDD wird direkt im Anschluss an den Import der entsprechenden Dictionary-Elemente durchgeführt. Für die entsprechenden Objekte hat zu diesem Zeitpunkt noch keine Generierung stattgefunden. Dies ist erforderlich, da sonst Kundendaten verloren gehen könnten, da sie modifizierte Felder benutzen (Beispiel: Der Kunde hat ein Feld einer Tabelle in einer Modifikation vergrößert und entsprechende Daten eingefügt, beim Einspielen der neuen SAP-Version würde dieses von der alten, kurzen Version überschrieben und Daten gingen verloren). Die Objekte, die von der Transaktion SPAU behandelt werden, können

hingegen nicht für einen Verlust von Kundendaten sorgen. Daher werden diese erst später abgeglichen. Wird dieser Abgleich innerhalb von 30 Tagen nach dem Upgrade durchgeführt, ist keine Angabe eines SSCR-Schlüssels erforderlich. Nach Ablauf dieser Zeit müssen für die zu modifizierenden Objekte entsprechende Schlüssel bei SAP beantragt werden.

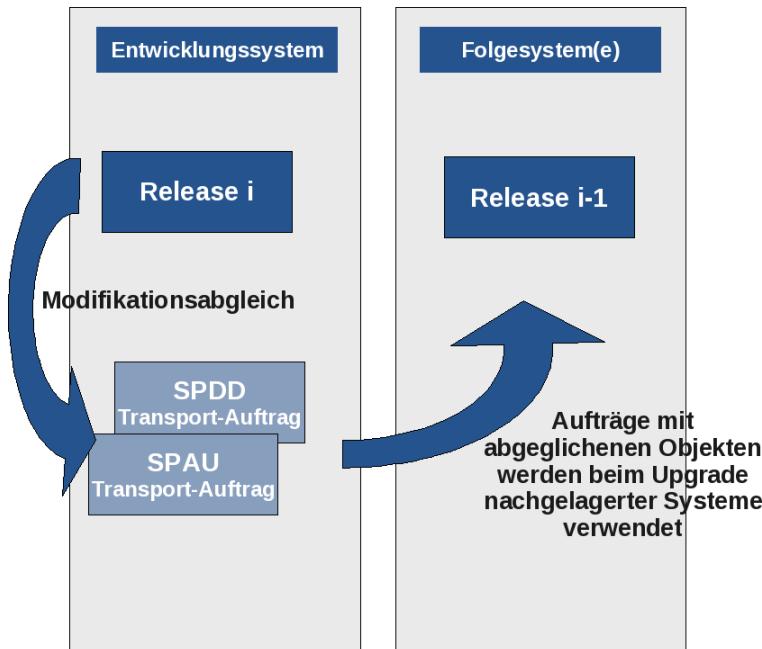


Abbildung 79: Abgleich und Transport

Wie bereits eingangs erläutert, wird der Modifikationsabgleich in aller Regel auf dem Entwicklungssystem vorgenommen. Von dort aus findet ein Transport zu den nachgelagerten Systemen (z. B. dem Produktivsystem) statt. Für den Abgleich mit der SPDD-Transaktion und den mit der SPAU-Transaktion werden in der Regel zwei getrennten Transportaufträgen verwendet. Auf dem nachgelagerten System können die vorgenommenen Abgleiche dann im Rahmen des Upgrade einzeln bestätigt oder abgelehnt werden. Danach wird für alle Modifikationen des nachgelagerten Systems geprüft, ob für diese Abgleiche aus den Transportaufträgen gefunden werden. Ist dies der Fall, ist kein Abgleich im Rahmen des Upgrades des Folgesystems erforderlich. Dies ist ein Grund dafür, dass Modifikationen stets auf dem Entwicklungssystem vorgenommen werden sollten: Hierdurch wird der nötige Abgleich bereits dort durchgeführt, während nur auf dem Produktivsystem durchgeführte Anpassungen nicht durch den Abgleich des Entwicklungssystems abgedeckt werden.



Abbildung 80: Einstiegsbild des Modifikationsabgleichs (SPAU): SAP-System-Screenshot

Die Transaktion **SPAU** bietet eine Suchfunktion zum Auffinden bzw. Filtern von abzugleichenden Objekten. Diese werden dann als Ergebnisbaum strukturiert, nach der Unterstützung des Modifikationsassistenten (mit / ohne) und der Art des Objekts angezeigt.

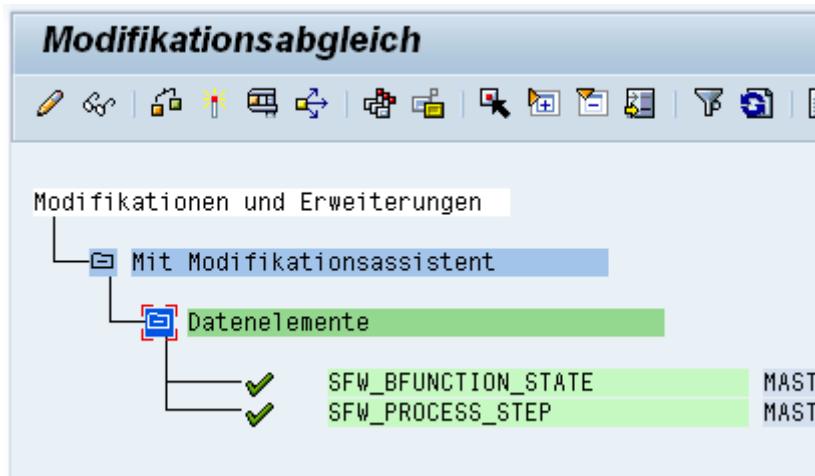


Abbildung 81: Liste abzuleitender Objekte: SAP-System-Screenshot

Über die Symbole vor dem jeweiligen Objekt wird kenntlich gemacht, ob der Abgleich bereits durchgeführt wurde bzw. wie dieser noch durchzuführen ist. Sie finden auch hier über den Menüpfad **Hilfsmittel -> Farblegende** eine genaue Angabe der Bedeutung jedes Symbols.

Objekte, die als **automatisch** abgleichbar gekennzeichnet sind, können ohne Eingreifen des Nutzers übernommen werden. Der Status **halbautomatisch** bedeutet, dass zwar keine unmittelbare Übernahme möglich ist, jedoch Hilfsmittel vom System angeboten werden. Darunter fällt beim Abgleich von Programmcode der **Split-Screen-Editor**. Dieser zweigt zwei Quellcodes nebeneinander an und ermöglicht so das Bequeme übernehmen von Änderungen. Der Status **manuell** sagt hingegen aus, dass keine Unterstützung angeboten wird und der Abgleich daher manuell im jeweiligen Werkzeug vorgenommen werden muss. Statt des Abgleichs ist mit einer entsprechenden Schaltfläche auch ein Rücksetzen auf das Original von SAP möglich. Hierdurch gehen die Modifikationen verloren und es ist zukünftig kein Abgleich des betroffenen Objekts mehr notwendig.

#### 6.4.7 Fazit: Modifikationen

Modifikationen bieten die Möglichkeit einer umfassenden Anpassung des SAP-Systems, besitzen jedoch den Nachteil dass in der Regel ein Modifikationsabgleich notwendig wird, der sehr viel Zeit kosten kann. Aus diesem Grund sollten Modifikationen so selten wie möglich durchgeführt werden und bei der Durchführung sorgsam vorgegangen werden.

Um die Wartbarkeit des Systems zu gewährleisten, sollte der in den SAP-Quellcode eingefügte Kunden-Code möglichst von Modularisierungseinheiten wie Funktionsbausteinen gekapselt werden, anstatt ihn in voller Breite unmittelbar in das SAP-Programm einzufügen.

Alle Änderungen sollten sorgfältig Dokumentiert sein,

- durch Kommentare im Code, wie sie vom Modifikationsassistenten unterstützt werden (hier insbesondere auch Kommentare die ggf. den entfernten SAP-Code enthalten),
- durch führen eines Modifikations-Logbuchs und
- durch die Definition von Unternehmensstandards für die Kommentierung im Quelltext.

Das Logbuch sollte alle Modifikationen enthalten und zu diesen jeweils angeben, welches Objekt von welchem Typ geändert wurde (Programm (ggf. Routine), GUI-Status, ...). Weitere empfohlene Angaben umfassen die Reparaturnummer, das Datum der Änderung, der Name des Mitarbeiters, der die Änderung durchgeführt hat, die Angabe, ob eine Vorabkorrektur mit der Transaktion SPDD notwendig ist, ggf. die SAP-Hinweisnummer nebst ihrer Gültigkeitsangabe sowie eine Aufwandsschätzung für den Abgleich der Modifikation, wenn diese beibehalten werden soll.

Dictionary-Elemente der zentralen Teile des SAP-Systems, wie etwa der ABAP Workbench, sollten durch Kunden nicht verändert werden, es sei denn Sie werden dazu explizit von SAP aufgefordert. Beim Austausch dieser Komponenten in einem Upgrade gehen die Modifikationen verloren, ohne dass ein Abgleich möglich wäre.

## 6.5 Erweiterungen durch das neue Erweiterungskonzept

*Achtung: Teile dieses Kapitels können nur mit SAP GUI für Windows bearbeitet werden, nicht aber mit der Java-GUI für Linux und Mac OS!*

In den vergangenen Unterkapiteln haben Sie sowohl unterschiedliche Erweiterungskonzepte als auch Modifikationen von SAP-Programmen kennen gelernt. Mit dem SAP NetWeaver 7.0 wurden mit dem neuen Erweiterungskonzept weitere Techniken aufgenommen, mit denen das SAP-System angepasst werden kann. Es mag auf den ersten Blick überflüssig erscheinen, noch zusätzliche Techniken anzubieten. Bei genauerem Hinsehen ergeben sich durch die neuen Techniken aber klare Vorteile gegenüber den bisher verwendeten Techniken. Als Entwickler im SAP-System müssen Sie sich trotzdem auch mit den älteren Techniken auseinandersetzen: Je nachdem, wann ein Element des SAP-Systems entwickelt wurde, bietet es eine bestimmte Erweiterungstechnik. Neue Entwicklungen verwenden bspw. keine klassischen BAdIs mehr, sondern die in diesem Unterkapitel vorgestellten kernelbasierten BAdIs. Dennoch gibt es im SAP-System Software, die noch klassische BAdIs enthält. Sie müssen also jeweils prüfen, welche Technik für ein bestimmtes Anpassungsproblem angemessen und anwendbar ist.

### 6.5.1 Enhancement Points

Eine neue Möglichkeit der Erweiterung wird im SAP-System durch **Enhancement Points** geschaffen. Hierbei handelt es sich um bestimmte Stellen im Code von Programmen, Funktionsbausteinen und Klassen, an denen der Kunde modifikationsfrei Ergänzungen vornehmen kann. Es wird dabei zwischen **expliziten Enhancement Points** und **impliziten Enhancement Points** unterschieden. Explizite Enhancement Points haben mit den bisher betrachteten Erweiterungstechniken die Eigenschaft gemein, dass sie vom SAP-Programmierer vorgedacht und explizit in das Entwicklungsobjekt integriert werden. An den so definierten Stellen kann dann der Kunde seinen Code einfügen. Implizite Enhancement Points müssen hingegen nicht vorgedacht werden: Sie existieren automatisch an bestimmten Stellen im Programmcode.

## SAP-Objekt

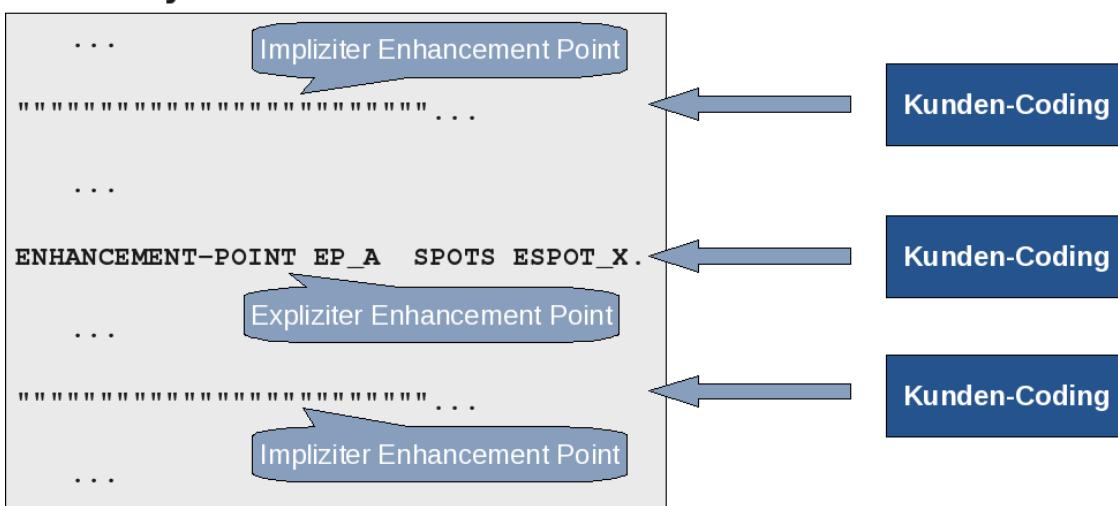


Abbildung 82: Implizite vs. explizite Enhancement Points

Implizite Enhancement Points existieren an den folgenden Stellen:

- Am Anfang und Ende von Unterprogrammen, Funktionsbausteinen und Methoden sowie am Ende von Includes. Zweck: Aufnahme zusätzlicher Funktionalität.
- Zu Methoden globaler Klassen können **Pre-** und/oder **Post-Methoden** definiert werden, die daraufhin vor bzw. nach Ausführung der Methode ausgeführt werden. Alternativ kann die Methode durch Definition einer **Overwrite-Methode** vollständig ersetzt werden.
- Am Ende der Definition eines Strukturtyps bzw. einer Struktur (vor END OF...). Zweck: Zusätzliche Felder zur Struktur hinzufügen.
- Am Ende des IMPORTING-, EXPORTING- und CHANGING-Blocks der Deklaration lokaler Klassen sowie in den Schnittstellendefinitionen von Funktionsbausteinen und Methoden globaler Klassen. Zweck: Aufnahme zusätzlicher Parameter in die Schnittstelle.
- Am Ende der PUBLIC-, PROTECTED- und PRIVATE-SECTION von Lokalen Klassen. Zweck: Definition weiterer Methoden und Attribute.
- Am Ende des IMPLEMENTATION-Blocks lokaler Klassen. Zweck: Implementierung der zusätzlich deklarierten Methoden.
- Analog dazu können in globalen Klassen beliebige zusätzliche Attribute definiert sowie Methoden definiert und implementiert werden.

Die Nutzung impliziter Enhancement Points ist bei Enhancement Points, bei denen ABAP-Code eingefügt wird, direkt in den ABAP Editor integriert.

Das betreffende Objekt, z. B. ein Programm, wird im **Anzeigemodus** geöffnet, und aus dem Menü der Pfad **Programm -> Erweitern** gewählt. Die Oberfläche ändert sich dadurch zunächst nur leicht. Um nun die impliziten Enhancement Points sehen zu können, wird der Menüpfad **Bearbeiten -> Erweiterungsoperationen -> Implizite Erw.-Optionen einblenden** gewählt. Daraufhin erscheinen im Quelltext des Programms an allen impliziten Enhancement Points Zeilen wie die folgende:



Abbildung 83: Anzeige eines impliziten Enhancement Points: SAP-System-Screenshot

Diese Zeilen werden verwendet, um die Implementierung anlegen zu können. Dies geschieht durch Klicken mit der rechten Maustaste auf die Zeile und die Auswahl von **Erweiterungimplementierung -> Anlegen** aus dem Kontextmenü. Das System fragt Art und Namen der Erweiterung und diese kann implementiert werden. Durch die Schaltfläche werden die Erweiterungen aktiv und sind daraufhin wirksam. Sie werden gleich anhand eines Beispiels die Anwendung üben.

Um die Schnittstelle einer Methode einer globalen Klasse zu erweitern sowie zur Definition zusätzlicher Attribute bzw. Methoden globaler Klassen, wählen Sie aus dem Class Builder den Menüpfad **Klasse -> Erweitern**. Sie werden dann dazu aufgefordert, eine Erweiterungimplementierung anzulegen. Anschließend können Sie im Erweiterungsmodus wie gewohnt Methoden und Attribute anlegen sowie die Parameter von Methoden ergänzen. Die bereits vorhandenen Attribute und Methoden sind hingegen gesperrt.

Die Erweiterung der Schnittstelle von Funktionsbausteinen geschieht analog zur Schnittstellenerweiterung bei Methoden globaler Klassen. Verwenden Sie hier den Menüpfad **Funktionsbaustein -> Erweitern** im Function Builder.

Eine Pre- Post- oder Override-Methode zu einer Methode einer globalen Klasse wird ebenfalls im Erweiterungsmodus angelegt. Hierfür ist die betreffende Methode auszuwählen und im Menü unter **Bearbeiten -> Erweiterungsoperationen** der entsprechende Unterpunkt anzuklicken. In den Spalten Pre-Exit, Post-Exit bzw. Overwrite-Exit erscheinen dadurch Schaltflächen:

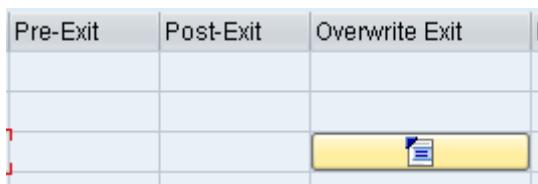


Abbildung 84: Schaltfläche zur Erweiterungsimplementierung: SAP-System-Screenshot

Über die jeweilige Schaltfläche kann dann zur Implementierung navigiert werden. Diese besteht aus einer automatisch generierten, lokalen Klasse.

Beachten Sie, dass zwar parallel Pre- und Post-Methoden, nicht jedoch eine Pre- und eine Overwrite- oder eine Post- und eine Overwrite-Methode zu einer Methode existieren dürfen.

## 6.5.2 Praxis: Beispiel zur Verwendung von Enhancement Points in Programmen

Öffnen Sie den Object Navigator und öffnen Sie darin das vordefinierte Programm **ZZ\_ABAP2\_ENH\_####**, indem Sie in den Feldern oberhalb des Navigationsbaums **Programm** aus der Drop-Down-Box auswählen, den Namen des Programms in das Feld darunter eingeben und mit Enter bestätigen.

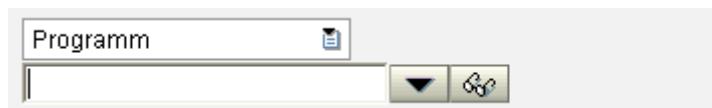


Abbildung 85: Öffnen des Programms: SAP-System-Screenshot

Testen Sie das Programm. Es handelt sich hier um einen Report, mit dem die im System bekannten Fluggesellschaften ausgegeben werden. Die Darstellung umfasst den Namen und die Internet-Adresse der Gesellschaft.

Airline	Website
American Airlines	<a href="http://www.aa.com">http://www.aa.com</a>
Air Berlin	<a href="http://www.airberlin.de">http://www.airberlin.de</a>
Air Canada	<a href="http://www.aircanada.ca">http://www.aircanada.ca</a>
Air France	<a href="http://www.airfrance.fr">http://www.airfrance.fr</a>
Alitalia	<a href="http://www.alitalia.it">http://www.alitalia.it</a>

Abbildung 86: Ausgabe des Programms ohne Erweiterung: SAP-System-Screenshot

Machen Sie sich mit dem Quellcode des Programms vertraut. Die Anwender bitten darum, vor dem Namen der Fluggesellschaft auch das Kürzel der Fluggesellschaft sehen zu können. Darüber hinaus soll zu jeder Fluggesellschaft in einer zweiten Zeile ausgegeben werden, wie viele Verbindungen diese anbietet. Auch bei diesem Beispiel sei davon ausgegangen, dass es sich nicht um ein Programm im Kundennamensraum handle, so dass eine direkte Bearbeitung nicht infrage kommt.

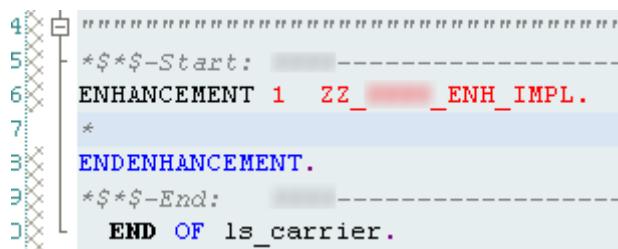
Wechseln Sie vom Anzeigemodus mit dem Menüpfad **Programm -> Erweitern** in den Erweiterungsmodus. Um die impliziten Enhancement Points sichtbar zu machen, wählen Sie **Bearbeiten -> Erweiterungsoperationen -> Implizite Erw.-Optionen einblenden**. Es werden daraufhin mit Zeilen, die sehr viele Anführungsstriche enthalten, die entsprechenden Stellen angezeigt. Es handelt sich um mehrere Stellen, wovon eine die Definition des Strukturtyps betrifft, während zwei andere das Unterprogramm einrahmen.

Klicken Sie mit der rechten Maustaste auf die erste der Stellen im Code und wählen Sie aus dem Kontextmenü **Erweiterungsoperationen -> Implementierung Anlegen**. Es erscheint das folgende Fenster:



Abbildung 87: Erweiterungsimplementierung anlegen: SAP-System-Screenshot

Geben Sie als Namen in das erste Feld **ZZ\_####\_ENH\_IMPL** ein, pflegen Sie einen Kurztext und bestätigen Sie dieses Fenster sowie die gewohnten Nachfragen nach Paket und Transportauftrag. Es entsteht daraufhin ein ENHANCEMENT-Block:



```
4
5 *$*$-Start:
6 ENHANCEMENT 1 ZZ_____ENH_IMPL.
7 *
8 ENDENHANCEMENT.
9 *$*$-End:
10 END OF ls_carrier.
```

Abbildung 88: Enhancement im Code: SAP-System-Screenshot

Die Stelle zwischen ENHANCEMENT und ENDENHANCEMENT ist hier der Bereich, in dem Sie ihren Code schreiben. Beachten Sie, dass diese Bereiche in sich syntaktisch korrekt sein müssen. Sie können daher hier nicht den umgebenden Kettensatz nutzen, sondern müssen die Zeile so schreiben, wie es ohne Kettensatz der Fall wäre.

Legen Sie analog auch an den anderen Enhancement Spots Entsprechende Codebereiche an und implementieren Sie so die gewünschte Funktionalität.

Tipp: Fügen Sie in der Strukturdefinition ein Feld für das Kürzel der Fluggesellschaft so ein, dass es von dem vorhandenen SELECT in die Struktur gelesen wird. So können Sie es am Beginn der Ausgabe ausgeben und anschließend zur Bestimmung der Verbindungsanzahl nutzen.

Speichern und prüfen Sie ihre Erweiterungimplemertierung und aktivieren Sie mit  die implementierten Erweiterungen. Das Ergebnis sollte etwa wie folgt aussehen:

Liste der Fluggesellschaften		
Fluggesellschaft	Internet-Adresse	
AA American Airlines	<a href="http://www.aa.com">http://www.aa.com</a>	2 Verbindungen im Angebot
AB Air Berlin	<a href="http://www.airberlin.de">http://www.airberlin.de</a>	0 Verbindungen im Angebot
AC Air Canada	<a href="http://www.aircanada.ca">http://www.aircanada.ca</a>	0 Verbindungen im Angebot

Abbildung 89: Ausgabe des Erweiterten Programms: SAP-System-Screenshot

### 6.5.3 Praxis: Beispiel für implizite Erweiterungsmöglichkeiten bei Klassen

Öffnen Sie den Object Navigator und öffnen Sie darin die vordefinierte Klasse **ZCL\_STOREROOM\_####**, indem Sie in den Feldern oberhalb des Navigationsbaums **Klasse** aus der Drop-Down-Box auswählen, den Namen der Klasse in das Feld darunter eingeben und mit Enter bestätigen.

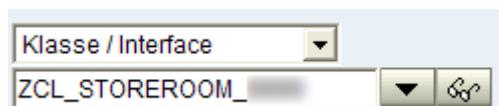


Abbildung 90: Öffnen der Klasse: SAP-System-Screenshot

Die Klasse repräsentiert Lagerräume in einem Gebäude. Testen Sie die Klasse und machen Sie sich mit den Attributen und Methoden vertraut.

### 6.5.3.1 Hinzufügen eines Attributs

Die Klasse soll in diesem Beispiel um ein Attribut **name** erweitert werden, um einen Lagerraum mit einer genaueren Bezeichnung versehen zu können. Gehen Sie wieder davon aus, dass es sich um eine SAP-Klasse handelt, die nicht direkt bearbeitet werden kann. Führen Sie also keine Änderungen außerhalb des Erweiterungskonzeptes durch.

Wählen Sie den Menüpfad **Klasse -> Erweitern** und geben Sie als Name für die Erweiterungsimplementierung **ZZ\_####\_ENH\_IMPL2** an. Pflegen Sie eine passende Kurzbeschreibung und Bestätigen Sie das Fenster wie auch die Nachfragen nach Paket und Transportauftrag.

Wechseln Sie zur Registerkarte **Attribute** und legen Sie das zusätzliche Attribut an. Dieses soll wie die anderen Attribute privat sein. Wechseln Sie anschließend zur Registerkarte **Methoden** und legen Sie dort eine entsprechende get- und set-Methode an. Sichern Sie und implementieren Sie die Methoden, sichern Sie diese auch jeweils und kehren Sie zur Klasse zurück. Öffnen Sie als nächstes die Signatur des Konstruktors. Fügen Sie dort den Namen als Parameter hinzu. Sichern Sie und öffnen Sie die Implementierung des Konstruktors. Wählen Sie den Menüpfad **Methode -> Erweitern**, um in den Erweiterungsmodus zu gelangen. Nutzen Sie hier nun ihre Kenntnisse über **implizite Enhancement Spots**, um das Attribut **name** auf den übergebenen Wert zu setzen. Dabei werden Sie nach einem Namen für diese Implementierung gefragt, wählen Sie diesmal **ZZ\_####\_ENH\_IMPL3**. Sichern Sie das Enhancement und kehren Sie zur Klasse zurück. Speichern, prüfen und aktivieren Sie Ihre Erweiterungsimplementierung und testen Sie die Klasse nun erneut. Das neue Attribut sollte nun vom Konstruktor gesetzt werden und funktionierende get- und set-Methoden haben.

### 6.5.3.2 Definition eines post-exits

Als neue Anforderung wird von Ihnen eine Anpassung der Methode **get\_max\_shelves** verlangt. Durch Brandschutzrichtlinien muss ein Stellplatz freigehalten werden, so dass sich die Maximalkapazität jeweils um einen Schrank reduziert.

Positionieren Sie den Cursor auf der betroffenen Methode und wählen Sie den Menüpfad **Bearbeiten -> Erweiterungsoperationen -> Post-Methode hinzufügen**. (Sollte dieser ausgegraut sein, haben Sie den Erweiterungsmodus verlassen, wählen Sie in diesem Fall erneut den Menüpfad **Klasse -> Erweitern**). Sichern Sie und öffnen Sie über die Schaltfläche in der Spalte **Post-Exit**. Sie gelangen hierdurch zum Quelltext der Post-Methode, mitsamt der Klasse die bei der Implementierung verwendet wird. Den leeren Methodenblock zur Implementierung finden Sie am unteren Ende:

```
 METHOD IPO_ZZ_■■■■■_ERW_IMPL2~GET_MAX_SHELVES.  
 *"-  
 ** Declaration of POST-method, do not insert any comments here please!  
 **  
 ** methods GET_MAX_SHELVES  
 **   changing  
 **     value(RE_MAX_SHELVES) type I .  
 **-  
- ENDMETHOD.
```

Abbildung 91: Leere Post-Methode: SAP-System-Screenshot

Sie sehen dort in den Kommentaren den Namen des Parameters, der von der eigentlichen Methode zurückgegeben wird. Verringern Sie diesen im Code der Post-Methode um 1. Speichern, prüfen und aktivieren Sie die Post-Methode und kehren Sie zur Klasse zurück. Aktivieren Sie auch diese und testen Sie, ob Ihre Methode nun durch die Post-Methode verringerte Rückgabewerte liefert.

Sie haben nun praktisch kennengelernt, wie mit impliziten Erweiterungsmöglichkeiten Anpassungen vorgenommen werden können, ohne dass diese von SAP vorgedacht werden mussten. Als nächstes werden Sie die expliziten Erweiterungsmöglichkeiten des neuen Erweiterungskonzepts kennen lernen.

#### 6.5.4 Explizite Enhancement Points und Enhancement Sections

Im Gegensatz zu den impliziten Enhancement Points werden **explizite Enhancement Points** vom SAP-Entwickler zum Einfügen von Kunden-Code in ein SAP-Programm vorgesehen. Um Code nicht nur ergänzen, sondern auch ersetzen zu können, gibt es zudem **Enhancement Sections**. Diese geben einen Bereich an, der durch Kundencode ersetzt werden kann.

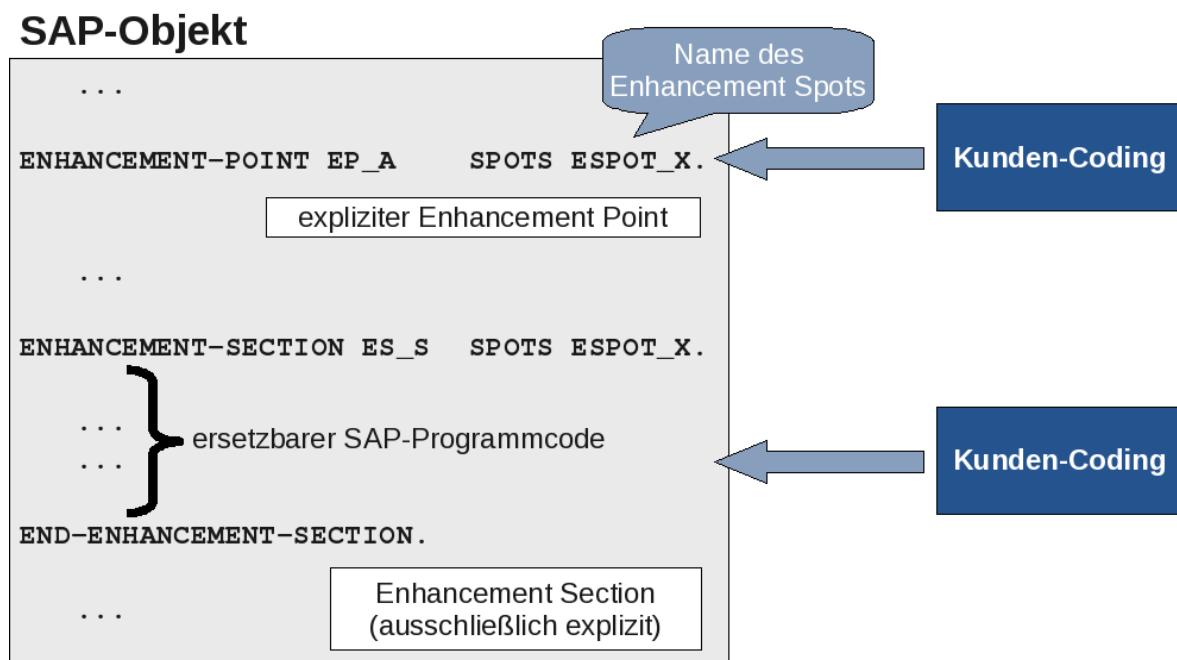


Abbildung 92: Explizite Enhancement Points und Enhancement Sections

Im Code erscheinen explizite Enhancement-Points durch die Anweisung ENHANCEMENT-POINT, während Enhancement Sections Blöcke sind, die durch ENHANCEMENT-SECTION und END-ENHANCEMENT-SECTION eingerahmt werden. Sowohl die Anweisung des Enhancement Point als auch die einleitende Anweisung der Enhancement Section besitzen einen Zusatz SPOTS, gefolgt von einem Bezeichner. Hier wird der sogenannte **Enhancement Spot** angegeben. Sowohl explizite Enhancement Points als auch Enhancement Sections und die im weiteren Verlauf dieses Unterkapitels behandelten neuen BAdIs werden über Enhancement Spots organisiert.

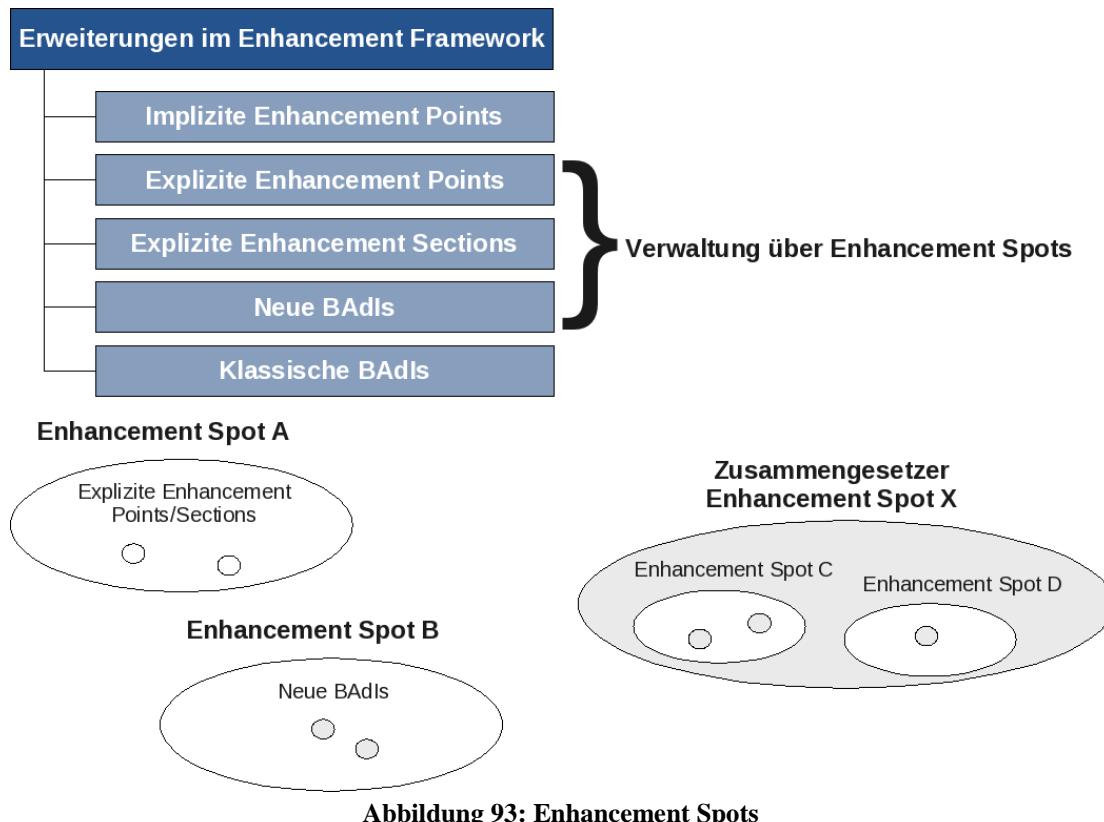


Abbildung 93: Enhancement Spots

Jeder explizite Enhancement Point, jede Enhancement Section und jedes neue BAdI ist einem **Enhancement Spot** zugeordnet, der als Verwaltungseinheit dient. Aus mehreren Enhancement Spots können **zusammengesetzte Enhancement Spots** gebildet werden. Diese können einfache und/oder weitere zusammengesetzte Enhancement Spots enthalten. So können inhaltlich zusammengehörende Enhancement Spots gebündelt werden.

Explizite Enhancement Points und Enhancement Sections, die die Erweiterung oder Ersetzung von Quelltext ermöglichen, werden **dynamisch** genannt, während solche die Deklarationen ersetzen oder erweitern, **statisch** genannt werden.

### 6.5.5 Kernelbasierte BAdIs

Im Rahmen der Einführung neuer Erweiterungstechniken wurde auch das BAdI-Konzept überarbeitet. Die neue BAdI-Technik, die **kernelbasierten BAdIs**, existiert dabei parallel zum klassischen BAdI-Konzept.

Die Motivation für die Einführung der neuen BAdI-Technik war vielfältig. Durch die neue Technik wird eine Performancesteigerung erreicht, da keine Adapterklasse mehr benötigt wird, die instanziert werden müsste. Stattdessen wird die Funktion über ein im Kernel des Systems verankertes BAdI-Handle wahrgenommen (daher auch die Bezeichnung dieser Technik). Weiterhin ist die neue BAdI-Technik Teil des Enhancement Frameworks. So ist es möglich, neue BAdIs zusammen mit Enhancement Points und Enhancement Sections in Enhancement Spots zu gliedern. Hinzu kommen die Integration in das Switch Framework, ein erweitertes

Filterkonzept zur bedingten Ausführung oder die Möglichkeit die Implementierung von einem Beispiel erben zu lassen und so nur Teile des Beispiels zu redefinieren.

Die folgende Abbildung zeigt zunächst die Architektur klassischer BAdIs, wie Sie sie bereits im entsprechenden Abschnitt dieses Kapitels kennen gelernt haben.

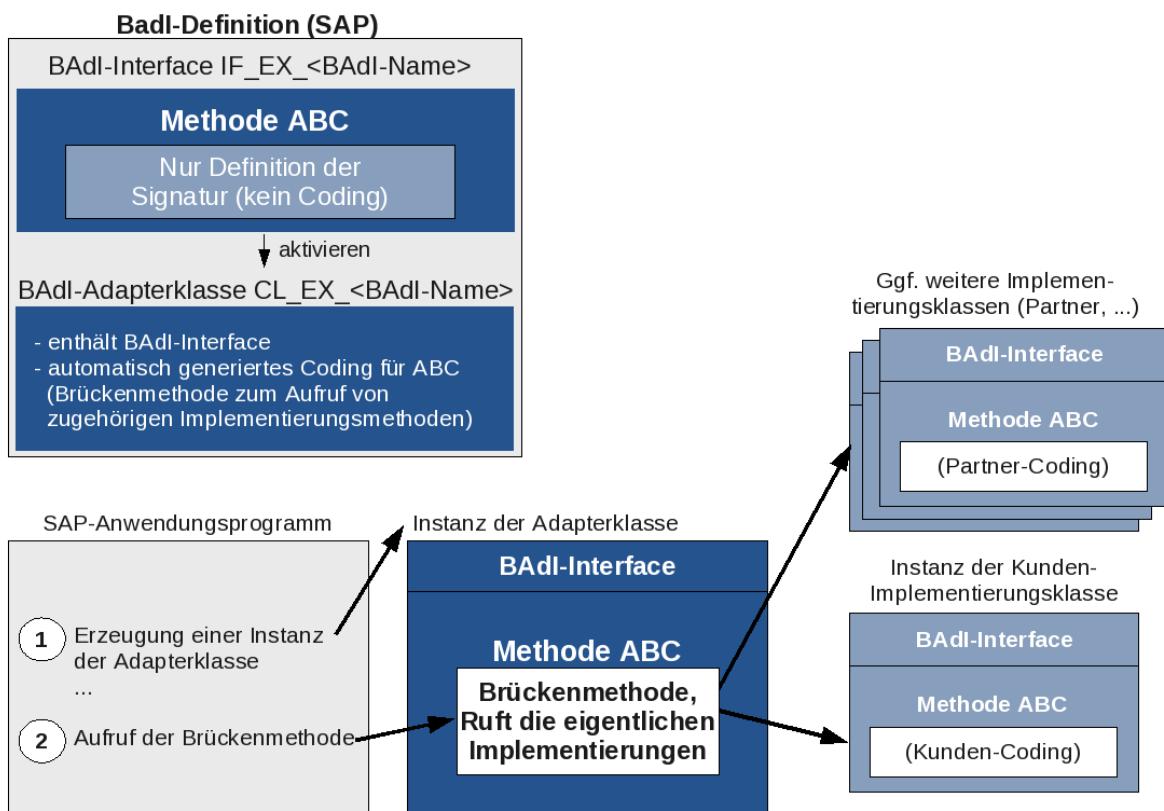


Abbildung 94: Architektur klassischer BAdIs

In dieser klassischen Technik wird automatisch eine Adapterklasse generiert, die das BAdI-Interface implementiert. Dabei werden die Methoden als Brückenmethoden der Adapterklasse erzeugt, die ihrerseits die vorhandenen (aktiven) Implementierungen aufrufen.

Im Code wird über die Klasse `cl_exithandler` eine Instanz der Adapterklasse erzeugt, und über eine Referenz auf diese Instanz die Methode aufgerufen:

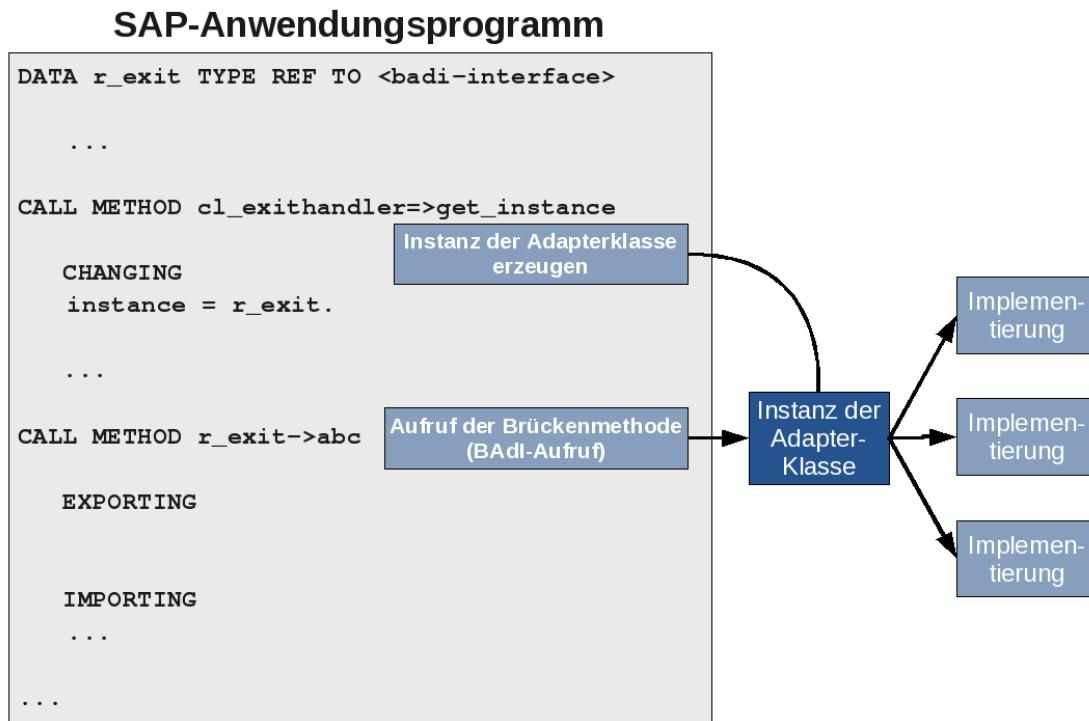


Abbildung 95: Aufruf klassischer BAdIs

Bei neuen BAdIs wird die Instanz der Adapterklasse nicht mehr benötigt. Ihre Rolle nimmt ein BAdI-Handle ein, das im Kernel definiert ist (daher auch die Bezeichnung kernelbasierte BAdIs). Der Verzicht auf die Adapterklasse und ihre Instanziierung durch Verwendung des kernelimplementierten Handles stellt einen erheblichen Performancegewinn dar, der für die Verwendung der neuen Technik spricht. Die folgende Abbildung stellt die Architektur der neuen BAdIs dar.

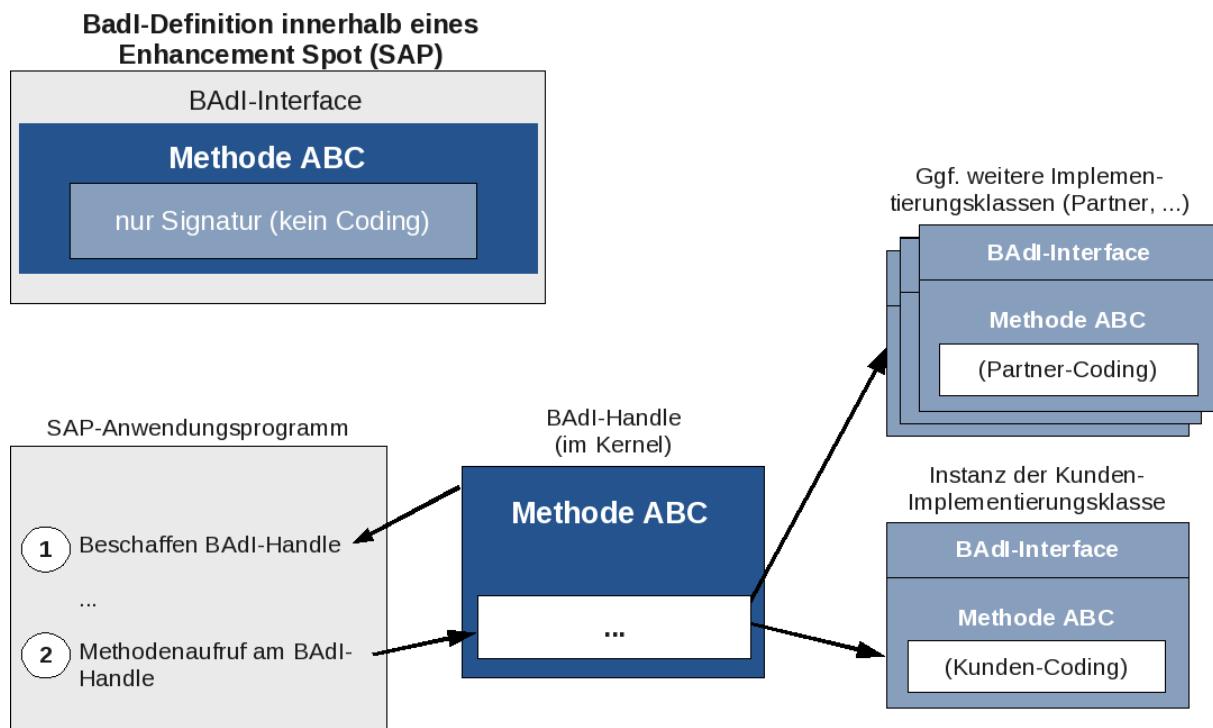


Abbildung 96: Architektur neuer, kernelbasierter BAdIs

Für die Beschaffung des Handles wird in dieser Architektur im Programm eine auf das BADI typisierte Referenzvariable benötigt. Die Beschaffung findet dann im SAP-Programm wie in der folgenden Abbildung dargestellt über den Befehl GET BADI statt. Der Aufruf der Methode am BADI-Handle geschieht über den speziellen Befehl CALL BADI.

Die folgende Abbildung zeigt beispielhaft beide Aufrufe. Dabei wurde jeweils eine Ausnahmebehandlung realisiert. `cx_badi_not_implemented` ist eine Ausnahme die eintritt, wenn zu einem BADI keine aktive Implementierung vorliegt. Die Ausnahme `cx_badi_initial_reference` wird hingegen beim CALL BADI-Befehl ausgelöst, wenn die angegebene Handle-Referenz (hier also `gb_demo`) initial ist (was etwa eine Folge des Fehlens einer Implementierung sein könnte). Beide Ausnahmen treten nicht bei BAdIs ein, die für die Mehrfachverwendung vorgesehen sind, zu denen also beliebig viele Implementierungen aktiv sein dürfen.

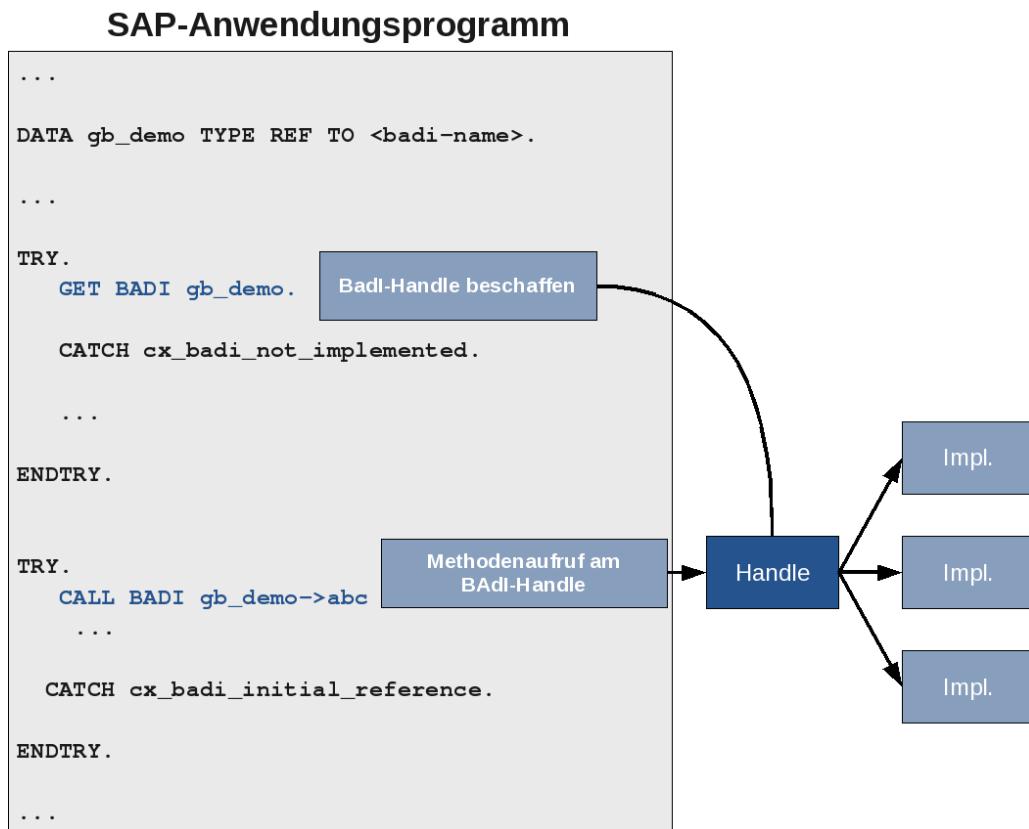


Abbildung 97: Aufruf neuer (kernelbasierter) BAdIs

Die Anweisungen GET BADI bzw. CALL BADI können hier zur programmlokalen Suche nach BAdIs genutzt werden. Für eine programmiübergreifende Suche können Sie das Repository Infosystem benutzen. Verwenden Sie dort wie bei den klassischen BAdIs im Navigationsbaum den Pfad **Repository Infosystem -> Erweiterungen -> Business Add Ins -> Definitionen**, und wählen Sie auf der Suchmaske **kernelbasierte BAdIs** aus. Um die Suche auf bestimmte Anwendungen zu beschränken, verwenden Sie die Anwendungshierarchie (**SE81**) und verzweigen von dort in das Infosystem.

Auch für neue BAdIs ist eine Filterung realisierbar, so dass abhängig vom Wert eines Filters bestimmte Implementierungen verwendet werden. Der Filterwert des Aufrufs wird als Zusatz FILTERS filtername = filterwert dem GET BADI-Befehl mitgegeben. Das Konzept ist bis dahin identisch mit dem klassischer BAdIs. Im Unterschied zu klassischen BAdIs können bei der Implementierung aber nicht nur einzelne Filterwerte, für die die Implementierung aufgerufen werden soll, angegeben werden, sondern ganze Bereiche können durch die Operatoren >, <, >=, <=, <>, CP und NP beschrieben werden. Weiterhin ist es möglich mehrere Filter zu einem BAdI zu definieren.

### 6.5.6 Praxis: Beispiel zur Verwendung eines filterabhängigen, kernelbasierten BAdI

In dieser Übung werden Sie ein neues BAdI verwenden und dabei auch die Filterung nutzen. Öffnen Sie zunächst im Object Navigator (**SE80**) den Report **ZZ\_ABAP2\_NBADI**, indem

Sie in den Feldern oberhalb des Navigationsbaums **Programm** aus der Drop-Down-Box auswählen, den Namen des Programms in das Feld darunter eingeben und mit Enter bestätigen.



Abbildung 98: Öffnen des Programms: SAP-System-Screenshot

Testen Sie das Programm. Es fragt Sie zunächst nach Ihrer vierstelligen Usernummer ####. Geben Sie diese ein und bestätigen Sie. Sie sehen nun eine Ausgabe, die zu jedem Land die Anzahl der Flugverbindungen enthält.

Beispielreport für neue BAdIs Diese Länder werden angeflogen:		
Ländercode	Anzahl	Details
DE	9	
ID	1	
IT	1	
JP	4	
SG	2	
US	9	

Abbildung 99: Ausgabe des Programms: SAP-System-Screenshot

Sollten Sie unterhalb von **Details** bereits Ausgaben sehen, hat ein anderer Kursteilnehmer seine Implementierung nicht richtig gefiltert. Bitte ignorieren Sie diese Ausgaben gegebenenfalls.

Das Programm soll nun so erweitert werden, dass unter Details die Information zu finden ist, wie viele Einzelflüge (gemeint ist die Tabelle SFLIGHT) in das jeweilige Land im System gespeichert sind. Hierfür werden Sie ein neues BAdI verwenden. Wie bei den anderen Beispielen auch, dürfen Sie das Programm **nicht** direkt bearbeiten, da angenommen sei, dass es sich um ein Programm außerhalb des Kundennamensraums handelt. Diese Vorgabe ist hier besonders wichtig, da das Programm kursweit nur einmal existiert, und Sie sonst die Arbeit anderer Nutzer beeinflussen würden.

Machen Sie sich nun mit dem Quellcode des Programms vertraut. Sie sehen, dass dort die charakteristischen Befehle GET BADI und CALL BADI auftauchen. Schauen Sie sich an, an welcher Stelle der BAdI-Aufruf zum Tragen kommt. Doppelklicken Sie auf die Variable nach GET BADI, um zu deren Definition zu navigieren, und dann auf den Typ der Variablen. Doppelklicken Sie auf den angegebenen Typ. Hierdurch öffnet sich der zum BAdI gehörige Erweiterungsspot.

Um ein neues BAdI zu implementieren, muss eine Implementierung des zugehörigen Erweiterungsspots erstellt werden. Dies erreichen Sie durch Klick auf (Erweiterungsspot Implementieren) oder Die F6-Taste. Benennen Sie die Implementierung **ZZ\_####\_ESPORTIMPL** und geben Sie eine passende Kurzbeschreibung ein. Bestätigen Sie

das Fenster sowie Paket und Transportauftrag. Es erscheint ein Fenster, in dem Sie die Implementierung des eigentlichen BAdI anlegen können. Geben Sie dort in der ersten Zeile in die erste Spalte den Namen **ZZ\_####\_NBADIIMPL** als Namen für die BAdI-Implementierung ein. Geben Sie analog in der zweiten Spalte **ZZ\_####\_NBADI\_CLASS** ein. Dies ist der Name der implementierenden Klasse. Wählen Sie in der Spalte BAdI-Definition den Wert **ZBADICOUNTRY** aus. Bestätigen Sie das Fenster sowie Paket und Transportauftrag.

Sie gelangen nun zur Erweiterungsimplementierung. Klappen Sie dort den Baum links unter der Registerkartenreihe auf und doppelklicken Sie auf **Implementierende Klasse** (siehe folgende Abbildung).

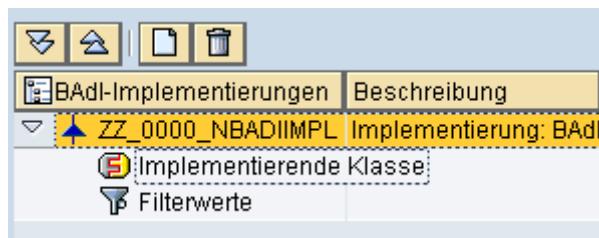


Abbildung 100: Navigation zur implementierenden Klasse: SAP-System-Screenshot

Doppelklicken Sie anschließend auf der rechten Seite auf den Methodennamen und bestätigen Sie die Nachfrage, ob Sie anlegen möchten, mit Ja. Implementieren Sie nun die Methode: Diese erhält einen Import-Parameter **IM\_COUNTRY**. Verwenden Sie diesen, um die Anzahl der Flüge zu bestimmen, die in der Datenbank (Tabelle **SFLIGHT**) gespeichert sind und in das angegebene Land führen. Sie werden hierfür auch die Tabelle **SPFLI** benötigen. Speichern, prüfen und aktivieren Sie. Kehren Sie dann zur Erweiterungsimplementierung zurück.

Durch die Angabe des Usernamens ist eine Filterung des BAdI vorgesehen. Doppelklicken Sie daher nun im oben abgebildeten Baum auf **Filterwerte**. Doppelklicken Sie auf **Kombination**. Es wird daraufhin eine neue Kombination von Werten für die Filterung angelegt. Klicken Sie auf die Fragezeichen unter **Wert 1**. Wählen Sie im erscheinenden Fenster als **Komparator 1** das Gleichzeichen und als **Wert 1** Ihre Usernummer ####.

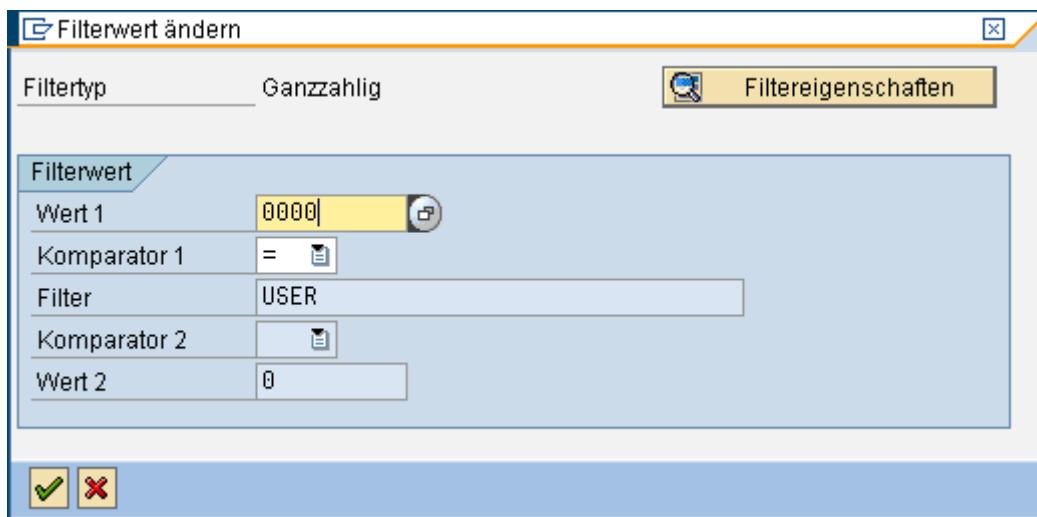


Abbildung 101: Einstellen der Filterung: SAP-System-Screenshot

Bestätigen Sie das Fenster, speichern und prüfen Sie Ihre Erweiterungsimplementierung und aktivieren Sie alle noch nicht aktiven Objekte. Starten Sie nun erneut das Programm. Bei Eingabe Ihrer Usernummer sehen Sie nun die zusätzlichen Ausgaben:

Ländercode	Anzahl	Details
DE	9	137 Fluege in der Datenbank
ID	1	14 Fluege in der Datenbank
IT	1	14 Fluege in der Datenbank
JP	4	57 Fluege in der Datenbank
SG	2	28 Fluege in der Datenbank
US	9	135 Fluege in der Datenbank

Abbildung 102: Ausgabe bei aktiver Implementierung: SAP-System-Screenshot

Rufen Sie das Programm erneut auf, und geben Sie als Test als Usernummer 7777 ein. Hierfür ist keine Implementierung vorhanden, daher sollte das Programm die gewohnten Ausgaben liefern, ohne dass Ihre BAdI-Implementierung zum Tragen kommt.

Sie haben nun praktisch kennen gelernt, wie kernelbasierte BAdIs verwendet werden und wie diese mit Filtern nur unter bestimmten Bedingungen ausgeführt werden.

### 6.5.7 Das Switch-Framework

Das Switch-Framework wurde von SAP eingeführt, um die Auslieferung von Industrielösungen zu vereinfachen. Alle Kunden erhalten ein Gesamtpaket der Industrielösungen und können die tatsächlich benötigte Lösung aktivieren. Die nicht benötigten Lösungen befinden sich dann zwar noch im System, sind aber nicht nutzbar. Analog dazu können bei Enhancement Packages, über die SAP Neuerungen des Systems ausliefert, einzelne neue Funktionen aktiviert werden.

Das Switch-Framework ermöglicht es, entsprechende Schalter zu definieren. Diesen werden dann Pakete, Dynpro-Elemente oder Menü-Einträge zugeordnet. Um einzelne Schalter

gruppieren zu können, gibt es **Business Functions**, denen die Schalter zugeordnet werden können. Eine SAP-Industrielösung entspricht dann wiederum einer Sammlung von Business Functions, die als **Business Function Set** bezeichnet wird.

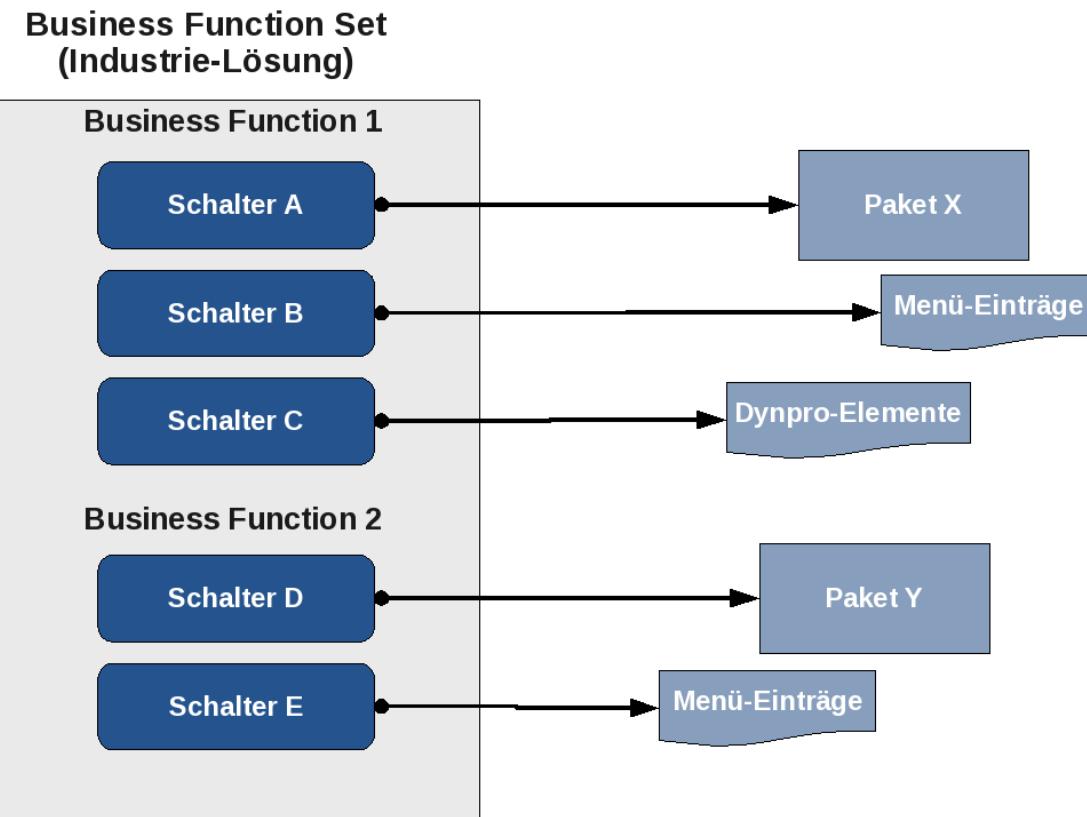


Abbildung 103: Elemente des Switch Frameworks

In der Transaktion **SFW5** kann das benötigte Business Function Set (genau eines) eingeschaltet, sowie einzelne Business Functions ein- und ausgeschaltet werden. Die folgende Abbildung zeigt die Transaktion, dabei steht die leuchtende Glühbirne für eine eingeschaltete, die dunkle Glühbirne für eine ausgeschaltete Business Function.

**S06 - Switch Framework: Business Function Status ändern**

Business Function Set		Vertragskontokorrent
Name	Beschreibung	
↳ FICAX	SAP Vertragskontokorrent	
💡 FICAX	branchenunabhängiges Vertragskor...	
💡 FICAX_CONV_INVOICING	Convergent Invoicing im branchenur...	

Abbildung 104: Business Function Sets und Business Functions (Transaktion SFW5): SAP-System-Screenshot

Neben dieser Verwendung des Switch Framework kann dieses auch vom Kunden zum Ein- und Ausschalten von Erweiterungsimplementierungen genutzt werden. Der Kunde definiert dafür einen Schalter (Transaktion **SFW1**) und ordnet diesen dem Paket zu, das die Erweiterungsimplementierungen enthält, die geschaltet werden sollen. Der Schalter wird

zudem einer Business Function zugeordnet. Diese definiert der Kunde in der Transaktion **SFW2** und nimmt auch dort die Zuordnung vor. Bei dieser Zuordnung kann zwischen den Zuordnungstypen **Aktivierung** und **Enabling/Standby** gewählt werden. Dies beeinflusst das Verhalten, wenn die Business Function eingeschaltet wird. Beim Typ Aktivierung werden alle mit dem Schalter verknüpften schaltbaren Objekte aktiviert. Zu diesen schaltbaren Objekten gehören auch die Erweiterungen des Kunden. Beim Typ Enabling/Standby werden hingegen nur deklarative Erweiterungen aktiviert, also Erweiterungen von Dictionary-Objekten. Im Normalfall ist der erstgenannte Typ daher sinnvoller.

Das Ein- und Ausschalten geschieht dann wiederum in der Transaktion **SFW5**. Das Ausschalten ist nur bei reversiblen Business Functions möglich. Diese Eigenschaft wird in der Transaktion **SFW2** festgelegt. Es hat zur Folge, dass die schaltbaren Objekte des Pakets (und somit die Erweiterungen) unwirksam werden, ohne dass diese aus dem System entfernt werden. Pakete mit Dictionary-Objekten können nicht ausgeschaltet werden.

## 6.6 Kontrollfragen

1. Welche dieser Aussagen sind wahr?
  - a. Ein Customizing-Include kann in mehreren Tabellen verwendet werden.
  - b. Eine Append-Struktur muss von SAP explizit vorgedacht sein.
  - c. Wird ein von SAP vorbereitetes Customizing-Include vom Kunden nicht implementiert, führt dies zu einem Fehler.
  - d. Wenn ein Feld zu einer Append-Struktur hinzugefügt wird, ist dies stets mit einer Datenbankumsetzung verbunden.
  - e. Texterweiterungen sind globale Erweiterungen.
  - f. Ein Append gehört immer zu genau einer Tabelle.
  - g. Der Name eines Customizing-Includes muss mit YY oder ZZ beginnen.
2. Wo kann ein Kunde eigene Globale Daten in einer Exit-Funktionsgruppe definieren?
3. Wie läuft der Datentransport zwischen einem SAP-Programm und einem per Customer-Exit eingebundenen Subscreen ab?
4. Unter welcher Transaktion werden Erweiterungsprojekte gepflegt?
5. Wo wird ein Subscreen definiert, der durch ein Dynpro-Exit mit der Customer-Exit-Technik eingebunden wird?
6. Welche Vorteile bietet die Verwendung des Modifikationsassistenten?
7. Welche grundlegenden Funktionen stellt der Modifikationsassistent beim Bearbeiten von ABAP-Code im Modifikationsmodus bereit?
8. Warum ist bei User-Exits kein Modifikationsabgleich erforderlich?
9. Welche der folgenden Aussagen sind wahr?
  - a. Eine Anwendungserweiterung besteht aus Erweiterungsprojekten.
  - b. Eine Komponente darf nur einmal in der Gesamtheit der SAP-Erweiterungen vorkommen.
  - c. Eine SAP-Erweiterung kann in verschiedenen Erweiterungsprojekten wiederverwendet werden.
  - d. Mit dem Modification Browser können Gruppen von Modifikationen zurückgenommen werden.
  - e. Zu jedem klassischen BADI kann es mehrere Implementierungen geben.
  - f. Der Modifikationsassistent bietet eine feinere Granularität bei der Beschreibung von Änderungen, als diese zuvor zur Verfügung stand.
  - g. Um ein Programm-Exit zu verwenden, muss der Kunde einen Funktionsbaustein erstellen.
  - h. User-Exits sind eine modifikationsfreie Erweiterungstechnik.
  - i. Über ein Dynpro-Exit können nur Subscreen-Dynpros eingebunden werden.
10. Welcher Befehl kann den Zusatz FILTERS besitzen?
  - a. Der GET BADI-Befehl
  - b. Der CALL BADI-Befehl

## 6.7 Antworten

1. Ja=Aussage ist wahr, Nein=Aussage ist falsch.
  - a. Ja
  - b. Nein
  - c. Nein
  - d. Nein
  - e. Ja
  - f. Ja
  - g. Nein
2. In einem Include ZX...TOP.
3. Siehe Erläuterungen zu den Funktionsbausteinen im entsprechenden Kursteil
4. In der Transaktion CMOD
5. In einer X-Funktionsgruppe
6. Siehe Erläuterungen im entsprechenden Kursteil
7. Einfügen, Ersetzen, Löschen, Modifikationen zurücknehmen, Modifikationsübersicht
8. Includes, die User-Exits enthalten, werden von SAP nur genau einmal ausgeliefert. Hierdurch kommt es nicht zum Konflikt zwischen Änderungen seitens SAP und Änderungen des Kunden auf seinem System.
9. Ja=Aussage ist wahr, Nein=Aussage ist falsch.
  - a. Nein, aus Programm-, Menü- und Dynpro-Exits
  - b. Ja
  - c. Nein
  - d. Ja
  - e. Nein, siehe Menü-Exits
  - f. Ja
  - g. Nein, der Funktionsbaustein wird von SAP erstellt. Der Kunde legt ein Include an.
  - h. Nein
  - i. Ja
10. a

## 6.8 Kapitelabschluss

Sie befinden sich am Ende dieses Abschnitts. Bevor sie die im folgenden Absatz beschriebene E-Mail verfassen, beachten Sie bitte die folgenden Hinweise:

1. Prüfen Sie, ob sie wirklich **alle** Aufgaben seit dem vorangegangenen Abschluss bearbeitet haben. Diese sind mit „Praxis:“ in der Überschrift gekennzeichnet (6.2.3, 6.3.2, 6.5.2, 6.5.3, 6.5.6).
2. Prüfen Sie bitte noch einmal genau ob alle ihre Repository-Objekte **korrekt funktionieren**.
3. Stellen Sie sicher, dass alle Repository-Objekte **aktiviert** sind. Um Objekte zu finden, die noch nicht aktiviert sind, wählen Sie aus dem Drop-Down-Menü oberhalb des Navigationsbaums im Object Navigator **Inaktive Objekte** aus. Geben Sie anschließend im darunter befindlichen Feld ihren Benutzernamen **USER#-###** ein und bestätigen Sie. Anschließend werden im Navigationsbaum die inaktiven Objekte dargestellt, die noch aktiviert werden müssen. Aktivieren Sie diese nun. Beachten Sie, dass sie die Zweige des Baums ggf. noch aufklappen müssen. Um zu ihrem Paket zurückzukehren, wählen Sie im Drop-Down-Menü wieder **Paket** aus und bestätigen Sie ihren Paketnamen.
4. Stellen Sie weiterhin sicher, dass die **Namen** ihrer Entwicklungsobjekte genau den Vorgaben im Skript entsprechen. Sollten Sie sich vertippt haben, können Sie Programme umbenennen, indem Sie diese mit der rechten Maustaste im Navigationsbaum des Object Navigators anklicken und **Umbenennen...** auswählen.

**Wenn Sie den Kurs bis zu dieser Stelle bearbeitet haben, senden Sie bitte eine formlose E-Mail an die vom Kursbetreuer für diesen Kurs genannte Adresse mit dem Betreff „ABAP2: Abschluss Kapitel 6 User ####“ (die Anführungszeichen gehören nicht mit zum Betreff). Sie erhalten dann in Kürze Feedback (je nach Ergebnis entweder über den Fortschrittsbericht, wenn alles in Ordnung ist, oder per E-Mail, wenn noch Korrekturen nötig sind) und können Mängel ggf. noch nachbessern. Bitte achten Sie darauf, den Betreff genau wie angegeben zu formulieren, um eine effiziente Verarbeitung der Mail zu ermöglichen.**

Sollten Sie Fragen haben, formulieren Sie diese bitte in einer **separaten E-Mail** mit aussagekräftigem Betreff, da die Kapitelabschluss-mails meist nur über den Betreff verarbeitet werden!