DSAG-HANDLUNGSEMPFEHLUNG

BEST PRACTICE LEITFADEN DEVELOPMENT – PRAXISTIPPS RUND UM DAS THEMA ABAP DEVELOPMENT

Deutschsprachige
SAP® Anwendergruppe

DSAG

BEST PRACTICE LEITFADEN DEVELOPMENT

PRAXISTIPPS RUND UM DAS THEMA ABAP DEVELOPMENT

Version:
2.0

Stand:
20. September 2016

Autoren:
Dr. Christian Drumm, Leiter Anwendungsentwicklung & Beratung,
FACTUR Billing Solutions GmbH

Martin Fischer, Portfolio Unit Manager SAP Database & Technology, BridgingIT GmbH

Judith Forner, Senior Consultant Finance & Controlling,
Mundipharma Deutschland GmbH & Co. KG

Edo von Glan, SAP-Entwickler, Drägerwerk AG & Co. KGaA

Florian Henninger, Senior Consultant SAP Development, FIS GmbH

Martin Hoffmann, Head of Software Engineering, Miele & Cie. KG

Valentin Huber, Senior IT Consultant, msg systems ag

Jens Knappik, SAP System Architect, thyssenkrupp Materials Services GmbH

Dr. Christian Lechner, Principal IT Consultant, msg systems ag

Steffen Pietsch, Head of Backoffice, Haufe-Lexware GmbH & Co.KG

Daniel Rothmund, IT Business Analyst SAP, Geberit Verwaltungs GmbH

Holger Schäfer, Business Unit Manager, UNIORG Solutions GmbH

Denny Schreber, Senior Solution Architect, cbs Corporate Business Solutions Unternehmensberatung GmbH

Andreas Wiegenstein, Managing Director und Chief Technology Officer (CTO), Virtual Forge GmbH

Bärbel Winkler, Systemanalystin SAP-Basis/Programmierung,
Alfred Kärcher GmbH & Co. KG

Weitere Informationen zu den Autoren finden Sie in Kapitel „11 Die Autoren“.

UNSER DANK GILT ALLEN BETEILIGTEN

Darüber hinaus gilt besonderer Dank den SAP-Mitarbeitern, die uns durch Reviews, Diskussionen und konstruktive Vorschläge bei der Erstellung der 2. Auflage des Leitfadens aktiv unterstützt haben. Insbesondere danken wir Jürgen Adolf, Carine Tchoutouo Djomo, Olga Dolinskaja, Thomas Fiedler, André Fischer, Dr. Thomas Gauweiler, Oliver Graeff, Michael Gutfleisch, Martin Huvar, Karl Kessler, Michael Schneider, Harald Stevens, Christoph Stöck, Dr. Wolfgang Weiss, Wolfgang Wöhrle und Margot Wollny.

INHALT

ABBILDUNGEN

1 EINLEITUNG

Die Software der SAP zeichnet sich als Standardsoftware durch ein hohes Maß an Flexibilität und Erweiterbarkeit aus. In nahezu allen Unternehmen, die SAP-Software einsetzen, finden sich kundenspezifische Anpassungen und Erweiterungen. Die SAP-Software unterliegt damit sowohl auf Hersteller- als auch auf Kundenseite der kontinuierlichen Anpassung und Erweiterung durch sich ändernde Kundenbedürfnisse.

Das hohe Maß an Flexibilität und Erweiterbarkeit von SAP-Software bringt Vor- und Nachteile mit sich: Die Software kann optimal an kundenspezifische Anforderungen angepasst und damit die Wertschöpfung durch den Einsatz deutlich gesteigert werden. Zeitgleich birgt die Erweiterbarkeit das Risiko kundenspezifischer Entwicklungen, die komplex, aufwendig wartbar und fehleranfällig sind.

2012 erschien die erste Version des DSAG-Best-Practice-Leitfadens mit dem Ziel, Praxistipps und Denkanstöße für die wartbare und effiziente Gestaltung kundenspezifischer Entwicklungen zu liefern. In den Folgejahren wurde er in die englische Sprache übersetzt. Aus dem Leserkreis der ersten Version haben wir zahlreiche Rückmeldungen erhalten. Zudem hat sich in der SAP-Entwicklung in den letzten Jahren viel verändert und es wurden etliche Neuerungen eingeführt. Deshalb stellen wir Ihnen hiermit die zweite, vollständig überarbeitete und aktualisierte Auflage des DSAG-Best-Practice-Leitfadens zur Verfügung.

1.1 MOTIVATION & IHR MITWIRKEN

Die Arbeit der Deutschsprachigen SAP-Anwendergruppe e.V. (DSAG) fußt auf drei Säulen – Wissensvorsprung, Einflussnahme und Netzwerk. Das vorliegende Dokument wurde von Mitgliedern des DSAG-Arbeitskreises SAP NetWeaver Development initiiert und adressiert die erste Säule, Wissensvorsprung für Anwender und Partner.

Als Autorenteam ist es unser Anliegen, das in den Unternehmen verteilt und implizit vorliegende Wissen zum Thema Entwicklung in einem kompakten Dokument anderen DSAG-Mitgliedern zur Verfügung zu stellen. Unser Wunsch ist, dass dieses Dokument „lebt“ und mit Ihrem Erfahrungsschatz einer kontinuierlichen Verbesserung unterliegt.

Wir haben hierzu eine Community-Webseite eingerichtet, über die Sie weitere Informationen zum Leitfaden erhalten, mit dem Autorenteam und anderen DSAG-Mitgliedern in Kontakt treten und Ihre Rückmeldung mit anderen teilen können:

www.dsag.de/leitfaden-abap

Wir freuen uns auf Ihr Feedback!

1.2 POSITIONIERUNG

Von der SAP und einer ganzen Reihe von Fachverlagen existieren bereits sehr gute Publikationen zu Anwendungsentwicklung und Erweiterung der SAP-Plattform. Im Verlauf dieses Leitfadens weisen wir auf aus unserer Sicht lesenswerte Literatur hin.

Der Mehrwert dieses Dokuments liegt in der Zusammenfassung bewährter Vorgehensweisen, Praxistipps und erprobter Regelwerke aus den Anwenderunternehmen. Diese Guideline soll Ihnen als Anwender, Entwickler, Entwicklungs-, Projekt- oder IT-Leiter Anregungen und Hilfestellung geben, um „das Rad nicht immer wieder neu erfinden zu müssen“ und auf die Erfahrungen anderer aufbauen zu können. Dabei erheben die in dieser Guideline vorgestellten Empfehlungen nicht den Anspruch auf Vollständigkeit oder absolute Gültigkeit, sondern stellen eine Auswahl von Praxistipps dar.

Als Autorenteam haben wir uns darum bemüht, im Spannungsfeld zwischen Überblickswissen und Detailtiefe den richtigen Mix zu finden. Daher verweisen wir an entsprechenden Stellen auf weiterführende Quellen, um bereits ausführlich diskutierte Themen nicht redundant wiederzugeben.

1.3 ÄNDERUNGEN IN DER 2. AUFLAGE

Die zweite Auflage dieses Dokuments orientiert sich an der Struktur der ersten Auflage aus dem Jahr 2012. Inhaltlich wurde jedes Kapitel grundlegend geprüft und überarbeitet. Hierbei wurden einige Empfehlungen der ersten Auflage aufgrund von Rückmeldungen der DSAG-Mitglieder angepasst und andere umfassend durch das Autorenteam erweitert. Darüber hinaus sind die Kapitel „Entwicklungsumgebung“ und „User Interface“ neu hinzugekommen.

Auch wenn Sie bereits mit der ersten Auflage dieses Leitfadens vertraut sein sollten, empfehlen wir Ihnen die Lektüre und Anwendung der Empfehlung dieser vollständig überarbeiteten, zweiten Auflage.

2 PROGRAMMIERRICHTLINIEN

Dieses Kapitel beschreibt erprobte und empfohlene Programmierrichtlinien für Anwendungen, die mit Hilfe der Programmiersprache ABAP erstellt werden. Es wird beschrieben, wie mit Standard-SAP-Mitteln und Disziplin lesbarer und verständlicher ABAP-Quellcode entwickelt werden kann. Dies erleichtert die Wartung des Quellcodes und ermöglicht, dass verschiedene interne und externe Personen effizient an der (Weiter-) Entwicklung und Wartung eines Programms zusammenarbeiten.

Verwendung der offiziellen ABAP-Programmierrichtlinien

Seit NetWeaver 7.31 SP5 sind die offiziellen ABAP-Programmierrichtlinien der SAP fester Bestandteil der ABAP-Schlüsselwortdokumentation. Sie sind dem Entwickler über die lokale Systeminstallation (Transaktion ABAPDOCU / F1-Hilfe im ABAP-Editor / ADT) oder über das SAP Help Portal[1] zugänglich. Neben dem hochwertigen und umfangreichen Inhalt hat das Dokument gegenüber anderen Entwicklungsrichtlinien den Vorteil, dass es direkt in die Entwicklungsumgebung integriert werden kann und nicht in der Schreibtischschublade verstaubt. Sie finden eine Anleitung für die SAP-Easy-Access-Menü-Integration im SAP-Hinweis 1387086[2] bzw. für die SE80-Integration im verlinkten SAP-Community-Network (SCN)-Blog[3].

Die ABAP-Programmierrichtlinien der SAP zeichnen sich aus durch

- sehr fundierte und praxisnahe Empfehlungen,
- detaillierte Informationen zu jeder Regel mit Beispielen,
- ihre Verfügbarkeit in Deutsch und Englisch,
- die kontinuierliche Weiterentwicklung durch SAP,
- eine interaktive Navigation zu Schlüsselwortdokumentationen, Release-abhängigen Änderungen, Performance- und Sicherheitsthemen, Beispielprogrammen und Transaktionen,
- die Nutzung aller verfügbaren ABAPDOCU-Funktionen (Export als HTML oder PDF, Suche, etc.),
- ihren hohen Bekanntheitsgrad und Quasi-Standard,
- individuelle Erweiterungsoptionen.

1 Vgl. SAP Help Portal „ABAP Schlüsselwortdokumentation – NW7.50"
2 Vgl. SAP-Hinweis „1387086 – HTML Viewer Control auf Einstiegsbild von SAP Easy Access"
3 Vgl. SCN-Artikel „Enhancing the ABAP Workbench with a website containing dev guidelines!"

Erweiterung der ABAP-Programmierrichtlinien

Eine Erweiterung der ABAP-Programmierrichtlinien macht vor allem dann Sinn, wenn Sie Themenbereiche aufnehmen wollen, die die Standard-Regeln momentan nicht abdecken. Hierzu gehören Designprinzipien zur Objektkomposition (SOLID[4]/GRASP[5]) und architektonische Konzepte wie z.B. das SAP-Paketkonzept oder die Verwendung von Frameworks. Denkbar ist auch eine zusätzliche Auflistung der Regeln, die für Ihr Unternehmen als zentral erachtet werden. Weitere Details, wie Sie die ABAP-Schlüsselwortdokumentation erweitern können, entnehmen Sie bitte dem Anhang (Kapitel B.1).

BEST PRACTICE

- Stellen Sie Ihre Entwicklungsrichtlinien in der Entwicklungsumgebung zur Verfügung, damit schnell auf sie zugegriffen werden kann.
- Statten Sie die Entwicklungsrichtlinien mit vielen transparenten und nachvollziehbaren Beispielen aus, die ggf. als Code Snippets wiederverwendet werden können.
- Orientieren Sie sich an den offiziellen ABAP-Programmierrichtlinien der SAP. Das sehr umfangreiche Dokument bietet für ein sehr breites Feld von Aktivitäten detaillierte Handlungsempfehlungen an.

4 Vgl. https://de.wikipedia.org/wiki/Prinzipien_objektorientierten_Designs#SOLID-Prinzipien
5 Vgl. https://de.wikipedia.org/wiki/GRASP

- Fassen Sie das Dokument der SAP bei Bedarf in einer Kurzversion zusammen. Durch seinen Umfang und den hohen Detaillierungsgrad ist die vollständige Sichtung der offiziellen ABAP-Programmierrichtlinien relativ aufwendig und beinhaltet zum Teil Themenbereiche, die nicht für jede alltägliche Entwicklungsaktivität relevant sind. Falls das Dokument in Ihrer Organisationseinheit eingesetzt werden soll, empfehlen wir, ein Team mit der Zusammenfassung des Dokuments zu beauftragen und den Umgang mit dem Dokument anhand von Schulungsmaßnahmen in die Organisation auszurollen. Gerade in größeren Organisationseinheiten macht es Sinn, das Team mit Interessenvertretern aus dem Bereich Qualitätssicherung und Entwicklung zu besetzen und die einzelnen Punkte ggf. an die individuellen Bedürfnisse der Organisation anzupassen.
- Wird das Dokument Bestandteil von externen Werkverträgen sollte der Abnahmeprozess durch die externen Entwickler in die Aktivierung des Entwicklerschlüssels eingebunden und im Dialog für die Abnahme auf das Dokument verwiesen werden.

WEITERE QUELLEN:

1. Development Guidelines for Greenfield Implementation in sync with SAP Custom Code Management[6]

2.1 NAMENSKONVENTION

Namenskonventionen beschreiben die einheitliche und verbindliche Vorgabe zur Benennung von Softwareobjekten (z.B. Klassen, Funktionsbausteinen) bzw. zur Benennung von Objekten im Quellcode (z.B. Variablen).

Wir empfehlen ausdrücklich, eine Namenskonvention als Richtlinie für Entwicklungen im SAP-System vorzugeben. Das Ziel der Verwendung einer einheitlichen Namenskonvention ist die deutliche Steigerung der Wartbarkeit kundenspezifischer Anpassungen und Erweiterungen. In der Konsequenz führt dies zu geringeren Wartungsaufwänden bzw. -kosten und einer schnelleren Problemlösung im Fehlerfall.

Die explizit formulierte Namenskonvention sollte Bestandteil der internen Ausbildung sein, um neue Mitarbeiter mit den Grundregeln und Unternehmensspezifika vertraut zu machen. Zudem hat es sich bewährt, diese Namenskonvention zum Vertragsgegenstand für externe Entwickler und Partnerunternehmen zu machen. Automatisierte Überprüfungen stellen die Einhaltung sicher (vgl. Kapitel 2.9 und Anhang A).

BEST PRACTICE
Eine exemplarische Namenskonvention finden Sie im Anhang A.

6 *Vgl. SCN-Artikel http://scn.sap.com/docs/DOC-56285*

2.2 NAMENSRAUM

Die Trennung von Kundenobjekten und SAP-Objekten kann über die Präfixe Y oder Z sowie über einen eigenen Namensraum erfolgen. Die Syntax lautet wie folgt:

Z...

Y...

/<kundeneigener Namensraum>/...

Der kundeneigene Namensraum kann bei SAP von SAP-Bestandskunden kostenfrei registriert werden und ist nach Bestätigung weltweit eindeutig und für das jeweilige Unternehmen zur Verwendung registriert. Dieses Vorgehen unterstützt bei der konfliktfreien Vergabe von Namen für Softwareobjekte.

Der Vorteil des kundeneigenen Namensraums liegt in der garantierten Überschneidungsfreiheit beim Import fremder Objekte in das eigene SAP-System (z.B. beim Einsatz von Fremdanwendungen, die per Transportauftrag eingespielt werden) und bei Zusammenführungen von SAP-Systemen im Rahmen einer Post-Merger-Integration. Durch die Reservierung des Namensraums ist sichergestellt, dass auf keinem fremden, d.h. nicht für diesen Namensraum registrierten, System ein Softwareobjekt mit dem gleichen Präfix erstellt werden kann.

Der Nachteil bei der Verwendung des kundeneigenen Namensraums liegt darin, dass durch die durchgängige Verwendung des Präfixes bereits mehrere Zeichen bei der Benennung von Objekten „verbraucht" werden. Besonders bei Objekten, die nur wenige Zeichen zur Benennung bieten, kann die Vergabe eines sprechenden Namens schwierig werden. Darüber hinaus gibt es Objekttypen, wie z.B. Berechtigungsobjekte, die die Verwendung von Namensräumen nicht unterstützen. Gleiches gilt für Tools und Frameworks der SAP. Obwohl es sinnvoll und empfehlenswert ist, diese einzusetzen, kann es bei der Verwendung zu Problemen kommen, weil die Verwendung eines Namensraums in der Partner- oder Kundenentwicklung nicht durchgängig oder gar nicht vorgesehen ist. Wir empfehlen daher, dies vor Einsatz eines neuen Tools oder Frameworks zu überprüfen.

BEST PRACTICE
Wir empfehlen die Verwendung eines kundeneigenen Namensraums.

WEITERE QUELLEN

1. http://help.sap.com (Namensraum einrichten)
2. Hinweis 105132 – Reservierung von Namensräumen
3. Hinweis 84282 – Entwicklungs-Namensräume für Kunden und Partner
4. SAP Support Portal: http://support.sap.com/namespaces

2.3 LESBARKEIT UND MODULARISIERUNG

Es ist nicht einfach, einen gut lesbaren und einfach zu verstehenden Quellcode zu erzeugen und es erfordert Disziplin und Professionalität. Der Aufwand lohnt sich aber langfristig, vor allem bei langlebigen Applikationen mit kontinuierlichen Wartungs- und Erweiterungsaktivitäten. Doch was macht gut lesbaren, einfach zu verstehenden Quellcode aus? Wirft man einen Blick über den ABAP-Tellerrand[7], geht es immer um die Einfachheit des Quellcodes. Um dies zu erreichen, schlagen wir folgende Vorgehensweisen vor:

- Einsatz von natürlicher Sprache („sprechende" Benennung von Variablen, Prozeduren usw.)
- Durchgängige Verwendung domänenspezifischer Bezeichnungen
- Vermeidung abstrakter, nicht direkt zu verstehender Begriffe und Abkürzungen

7 *Einschlägige Autoren / Bücher zu dem Thema sind:*
- *Robert C. Martin: Clean Code – Refactoring, Patterns, Testen und Techniken für sauberen Code: Deutsche Ausgabe, Heidelberg. MITP, 2009.*
- *Steve McConnell: Code Complete, 2. Auflage. Unterschleißheim. Microsoft Press, 2005.*
- *Martin Fowler: Refactoring – Improving the Design of existing Code. Addison-Wesley, 1999*

- Einheitliche Quellcodestrukturierung (Einrückung, Schreibstil)
- Aufgabe eines individuellen Programmierstils zugunsten eines gemeinsamen Standards
- Modularisierung
- Nicht zu lange / komplexe Prozeduren
- Vermeidung von globalen Variablen

Welche Besonderheiten gibt es im ABAP-Umfeld?

- Die Vermeidung abstrakter Begriffe ist teilweise schwierig. Vor allem bei älteren Entwicklungen (klassische SAP-Module) trifft man auf eine Vielzahl von deutschen Abkürzungen und Bezeichnern, die in internationalen Projektteams oder bei Junioren mit wenig SAP-Hintergrundwissen den Einstieg in die Materie zusätzlich erschweren.
- Die Entwicklungsumgebung schränkt den Einsatz von natürlicher Sprache zur Benennung von Repository-Objekten und anderen Bezeichnern zum Teil sehr stark ein. Hinzu kommt, dass ABAP im Gegensatz zu anderen Programmiersprachen einen globalen und keinen paketgebundenen Namensraum verwendet. In Kombination mit Präfixnamensräumen (/*/) fördern diese Umstände den Einsatz von schwer lesbaren Abkürzungen und können sich negativ auf die Produktivität der Entwickler auswirken.

Sieht man von diesen Besonderheiten ab, lassen sich die oben genannten Prinzipien gut umsetzen. Hilfreich sind die Refactoring-Werkzeuge der ABAP-Development-Tools (ADT) für Eclipse (vgl. Kapitel 9). Durch den Einsatz von ADT lassen sich z.B. einzelne Variablen oder ganze Klassen auf einen Schlag in allen gefundenen Verwendern umbenennen, wodurch sich der Aufwand (und das Fehlerrisiko) für die Korrektur eines unpassend gewählten Bezeichners auf ein Minimum reduziert.

BEST PRACTICE

- Berücksichtigen Sie die oben genannten Vorgehensweisen bei der Auswahl und Ausbildung der Entwickler, bei der Zeitplanung von Projekten (Zeitdruck bei der ursprünglichen Entwicklung kann zu schlechter wartbarem Quellcode und damit zu höheren Folgekosten führen) und bei der Planung von Qualitätsmaßnahmen (automatische und manuelle Prüfungen).
- Nutzen Sie eine einheitliche, natürliche Sprache, deren Begriffe im Quellcode, in den Dokumenten und in Gesprächen mit Kollegen und Kunden möglichst identisch sind. Orientieren Sie sich an der SAP-Terminologiedatenbank (Transaktion SAPTERM oder http://www.sapterm.com/).
- Definieren Sie Vorgaben für den einheitlichen Umgang mit Bezeichnern. Berücksichtigen Sie dafür Ihren Bezug zur SAP-Software (Kunde / Produktlieferant), die Menge des kundeneigenen Quellcodes, sowie die Aufbau- und Ablauforganisation Ihrer SAP-Entwicklungsabteilung (Generalisten / Spezialisten in nationalen / internationalen Teams). Je nach Konstellation der vorangegangenen Eigenschaften kann die Regulierung für den Umgang mit Bezeichnern (z.B. Variablenbezeichnung sales_organization statt vkorg[8], oder index statt i) die Wartungs- und Weiterentwicklungsaktivitäten beeinflussen. Nachfolgende Tabelle soll Sie bei der Entscheidungsfindung unterstützen.

8 *Die fünfstelligen Feldbezeichner aus den klassischen Modulen sind ein Sonderfall. Wenn sie in Strukturen verwendet werden, fällt bei Umstellung auf sprechende Bezeichner einiger Aufwand (und Fehlerrisiko) für das Mapping in beide Richtungen an. Daher kann die Beibehaltung (trotz schlechterer Lesbarkeit) sinnvoll sein.*

Kleines zentrales Team	Große / Dezentrale Teams	Einheitliche Muttersprache	Multilingual	Modulspezialisten	Generalisten	Viel Custom Code	Wenig Custom Code	Add-on Entwicklung
•	• •	•	• • •	•	• •	• • •	•	• •

• • • Starke Beeinträchtigung der Wartungs- / Weiterentwicklung durch schlechte Lesbarkeit / Bezeichnerwahl

• Geringe Beeinträchtigung der Wartungs- / Weiterentwicklung durch schlechte Lesbarkeit / Bezeichnerwahl

Code-Formatierung

Übersichtlicher, leserlicher Quellcode erleichtert jedem Entwickler die (erneute) Einarbeitung in den Quellcode. Als einfachste und schnellste Möglichkeit, Quellcode gut lesbar zu machen und zu halten, kann der Pretty Printer bzw. der Formatierer aus der ABAP-Entwicklungsumgebung bzw. den ABAP Development Tools genutzt werden. Mit einem einzigen Knopfdruck/Shortcut wird der ausgewählte Quellcode einheitlich formatiert. Er bietet verschiedene Konfigurationsmöglichkeiten. In der klassischen Workbench des SAP GUIs sind diese in den Workbencheinstellungen zu finden, in den ABAP Development Tools müssen die Einstellungen in den projektspezifischen Formatiereigenschaften hinterlegt werden. Wir empfehlen, den Quellcode einzurücken, Schlüsselwörter in Großbuchstaben sowie Bezeichner in Kleinbuchstaben darzustellen. Dadurch kann man Quellcode auch in ausgedruckter Form und ohne Syntaxeinfärbung leicht verstehen. Der Pretty Printer / Formatierer ermöglicht somit auf einfachem Weg ein einheitliches Quellcodelayout. Wir empfehlen, die Option „Standardkommentare einfügen“ zu deaktivieren, da die erzeugten Kommentare bei Änderungen nicht automatisch angepasst werden und redundante Informationen enthalten.

BEST PRACTICE

Wir empfehlen, den Pretty Printer / Formatierer zu verwenden und die Einstellungen als einheitliche Vorgabe zu definieren. Neben den Einstellungen im Pretty Printer empfehlen wir außerdem, die funktionale Schreibweise von Methodenaufrufen zu verwenden und auf den Ausdruck CALL METHOD zu verzichten.

Softwarestrukturierung mit dem SAP-Paketkonzept

Mit der Einführung des SAP NetWeaver Application Servers 6.20 wurde das Konzept der Entwicklungsklasse vollständig durch das SAP-Paketkonzept ersetzt und um viele Eigenschaften zur besseren Softwarestrukturierung erweitert. Neben der groben Dokumentation des Systemaufbaus liegt die wesentliche Aufgabe des Paketkonzeptes darin, zusammengehörige Objekte in einen gemeinsamen Rahmen einzubetten, den Zugriff auf Objekte zu regulieren und die Abhängigkeit zu anderen Paketen zu kontrollieren. Die Erläuterung der Einzelheiten zum Thema Paketkonzept würde den Umfang dieses Leitfadens sprengen, weshalb wir auf die vorhandene SAP-Help-Dokumentation[9] und auf die sehr ausführlichen SCN-Artikel[10] verweisen. Objektiv betrachtet, lassen sich Paketstrukturen auf Basis von fachlichen, technischen und organisatorischen Kriterien bilden, wobei nachfolgende Liste die gängigsten Kriterien aufführt:

- Abhängigkeit zur Softwarekomponente
- Zuordnung zur SAP-Standard-Anwendungshierarchie
- Wiederverwendbarkeit der Entwicklungsobjekte
- Gruppierung einzelner Anwendungen
- Stabilität der Entwicklungsobjekte
- Layerspezifische / technische Zugehörigkeit
- Organisatorische Zuordnung der enthaltenen Objekte
- Übersetzungsrelevanz der Entwicklungsobjekte

9 Vgl. SAP Help Portal – Suche nach Schlagwort Package Builder
10 Vgl. SCN-Artikel Tobias Trapp „ABAP Package Concept – Pert 1 – 4“

BEST PRACTICE

- Nutzen Sie Strukturpakete, um die Abhängigkeit von verschiedenen Softwarekomponenten hervorzuheben, umfangreiche Eigenentwicklungen sinnvoll zu gruppieren oder wenn Sie die Paketprüfung verwenden möchten.
- Prüfen Sie im Rahmen Ihrer Entwicklung, ob im Kundennamensraum bereits ein Hauptpaket für die passende Top-Level-Anwendungskomponente (SE81) existiert und legen Sie es anderenfalls an.
- Ordnen Sie Pakete immer der semantisch am besten passenden Anwendungskomponente zu.
- Verwenden Sie Pakethierarchien, um Ihre Softwaresysteme zu organisieren. Jedes Softwaresystem sollte mindestens über ein Hauptpaket verfügen, das seine Teilsysteme zusammenhält und deren wesentliche Absicht verdeutlicht.
- Setzen Sie Entwicklungspakete ein, um die semantisch und technisch zusammenhängenden Repository-Objekte an einer zentralen Stelle zu bündeln.
- Vermeiden Sie gegenseitige Abhängigkeiten zwischen zwei oder mehreren Paketen. Lagern Sie die abhängigen Komponenten in separate Pakete aus, so dass nur noch eine einseitige Abhängigkeit besteht.
- Ordnen Sie Pakete mit einseitiger Abhängigkeit möglichst weit oben in der Pakethierarchie Ihres Softwaresystems an. Entsteht Bedarf, die Funktionalität aus mehreren Softwaresystemen zu verwenden, kann das Paket in ein allgemeines Basispaket ausgelagert werden.
- Gruppieren Sie Repository-Objekte über ein paketspezifisches Namensraumpräfix. Orientieren Sie sich dabei an der Anwendungshierarchie oder an einem Produktkürzel (Beispiel: Alle Objekte in der Pakethierarche zu dem Hauptpaket Z_SALES_SUPPORT erhalten das Präfix SAS: Klasse ZCL_SAS_FIELD_WORKER, Stammdatentabelle ZSASFIELDWORKERS, etc.).

Paketschnittstellen und Verwendungserklärungen

Der Paketzugriff lässt sich auf allen Paketebenen über Paketschnittstellen und Verwendungserklärung definieren. Während die Paketschnittstelle der Bekanntmachung wiederverwendbarer Komponenten des Pakets dient, lässt sich über die Verwendungserklärung der Zugriff auf die Komponenten kontrollieren. Programmiert ein Entwickler an der Paketschnittstelle vorbei und greift z.B. auf SAP-Standard-Komponenten zu, die nicht über eine Paketschnittstelle freigegeben sind, kann das einen nicht zu unterschätzenden Aufwand auf Kundenseite im Rahmen von Upgrade-Aktivitäten bedeuten. Was bei Nutzung von nicht freigegebenen Komponenten passieren kann, ist exemplarisch im SCN-Blog[11] beschrieben.

Sie können den Zugriff auf Komponenten anderer Pakete optional über das Konzept der Paketprüfungen steuern. Weitere Details für die Aktivierung der Paketprüfung sind in dem SAP-Hinweis 648898[12] enthalten. Nach der Aktivierung können Sie die Werkzeuge des Paketes SPAK_API nutzen und die Paketzugriffe kontrollieren.

11 Vgl. http://scn.sap.com/community/abap/blog/2013/01/28/is-sap-nw-ehp-3-really-non-disruptive
12 Vgl. https://launchpad.support.sap.com/#/notes/648898

BEST PRACTICE

- Prüfen Sie vor der Wiederverwendung eines paketfremden Repository-Objekts erst, ob das Objekt in einer Paketschnittstelle freigegeben wurde. Nur Objekte die über eine Paketschnittstelle freigegeben sind, gewährleisten eine langfristige Stabilität. Im SAP-Standard reduziert die Verwendung solcher Objekte die Gefahr vor Veränderung bei einem Releasewechsel.
- Stellen Sie den Zugriff auf Objekte eines Paketes über die Definition von Paketschnittstellen zur Verfügung. Nur durch eine stringente Einhaltung der Paketschnittstellen lässt sich gewährleisten, dass die von Ihnen definierte Paketstruktur eingehalten wird und kein wilder Zugriff auf die Objekte Ihres Paketes stattfindet.
- Bieten Sie bei Bedarf mehrere Paketschnittstellen an und gruppieren Sie die Schnittstellen auf Basis semantischer Informationen, wie z.B. Datenmanipulation oder Reporting-Werkzeuge, oder nach Verwendungskriterien.
- Reichen Sie Ihre Paketschnittstellen bei Bedarf in der Pakethierarchie weiter nach oben. Dort haben Sie die Möglichkeit, die Zugriffe weiter einzuschränken oder die freigegebenen Elemente auch außerhalb Ihres Systems zugänglich zu machen.
- Schränken Sie die Verwendung von kritischen Funktionen ein, indem Sie für diese Objekte separate Paketschnittstellen anbieten. Die Nutzungsrechte der Schnittstellen können Sie über die Definition einer Einschränkung von Verwenderpaketen ausprägen.

Paketnamensräume

Neben den allgemein bekannten Kundennamensräumen Y*, Z* und /namespace/ bietet SAP über das Paketkonzept die Bereiche $* und T* an. $TMP dürfte jedem Entwickler geläufig sein. Aber wissen Sie auch, dass Sie die beiden Namensräume auch für eigene Pakethierarchien im Sinne von lokalen Entwicklungspaketen nutzen können? Der Unterschied zwischen den beiden Namensräumen besteht darin, dass $* nicht transportiert werden kann und sich der Entwickler manuell um die Versionierung kümmern muss. Im Gegensatz dazu lässt sich der Inhalt von T* Paketen manuell via Transport von Kopien in alle Systeme, die nicht als Produktivsystem gekennzeichnet sind, transportieren. Aus Sicht der Softwarestrukturierung ergeben sich daraus folgende Vorteile:

- Die Einarbeitung in neue Pakete ist einfacher, da weniger Overhead für die Identifikation von halb fertig entwickelten Test- und produktiv genutzten Komponenten entsteht.
- Produktivsysteme werden nicht mit unnötiger und evtl. sicherheitskritischer Funktionalität überfrachtet.
- Wartungs- und Weiterentwicklungskosten lassen sich reduzieren, da notwendige Änderungen zwingend nur in produktiv genutzten Paketen durchgeführt werden müssen.

BEST PRACTICE

- Nutzen Sie einen passenden Namensraum für die Anlage Ihrer Pakete. Orientieren Sie sich an der Transportrelevanz und dem Verwendungszweck der Entwicklung (Testentwicklung lokal / Testentwicklung mit abweichendem Transportziel).
- Achten Sie darauf, dass lokale Entwicklungen nicht in das Produktivsystem gelangen.

Einsatz des Paketkonzepts

Die vollständige Nutzung aller zur Verfügung stehenden Mittel des Paketkonzeptes ist nicht für jedes Unternehmen gleich geeignet und kann bei falscher Ausprägung nicht zu rechtfertigende Mehraufwände bedeuten. Um Ihnen bei der Entscheidungsfindung zu helfen, stellen wir Ihnen nachfolgende Tabelle zur Verfügung.
Die Bewertungskriterien orientieren sich an den Werten (+) = weniger wichtig bis (+++) = sehr wichtig.

	Kleines zentrales Team	Große / Dezentrale Teams	Regionale Spezialentwicklung	Übersetzungs-Aktivitäten	Viel Custom Code	Wenig Custom Code	Add-on-Entwicklung
Allg. Paketkonzept	++	+++	++	+++	+++	++	++
Paket schnittstellen	++	+++	+++	+	+++	+	+++
Strukturpakete	+	++	+	++	++	+	+++
Paketprüfung	+	++	++	+	+++	+	+++

Modularisierung

Programme, in denen logische Verarbeitungseinheiten zu lang sind und nicht aufgeteilt werden, sind in weiterer Folge schwer lesbar und damit schwer wart- und erweiterbar.

Eine Modularisierungseinheit (Form-Routine, Methode, Funktionsbaustein) soll logisch zusammengehörende Anweisungen zusammenfassen. Dabei ist jedoch zu beachten, dass die einzelnen Einheiten nicht triviale Funktionalitäten abdecken: Modularisierungseinheiten mit sehr wenigen Anweisungen sind zu vermeiden, es sei denn, sie dienen der besseren Lesbarkeit des Quellcodes und sind damit Teil der Dokumentation des Quellcodes.

Die Modularisierung dient dazu, trotz Komplexität in der Aufgabenstellung, den Quellcode übersichtlich zu gestalten. Siehe hierzu auch Abschnitt „2.12 Programmiermodell: Objektorientiert vs. Prozedural“.

Die ABAP Development Tools (siehe Kapitel 9) bieten umfangreiche Möglichkeiten für das Refactoring an, wie zum Beispiel die automatische Methodenextraktion, mit der man den Quellcode nachträglich besser modularisieren kann (natürlich gibt es auch hier Grenzen, sodass manuelles Editieren notwendig sein kann).

Vermeidung von globalen Variablen

Neben der Aufteilung des Programmtextes betrifft die Modularisierung auch die Sichtbarkeit der Variablen. Diese sollte so gering wie möglich gehalten werden.

Globale Variablen zerstören die Modularisierung eines Programms. Sie erschweren das Verständnis und die Wartbarkeit massiv. Fehlerbehebungen und Erweiterungen sind oft nur noch in der Art „Versuch und Irrtum“ möglich.

BEST PRACTICE
Vermeiden Sie sämtliche globale Variablen.

Grenzen der Regel

Bei kleinen Programmen mit nur wenigen, überschaubaren Modularisierungseinheiten sind globale Variablen meist unschädlich (durch die geringe Größe ist das Programm normalerweise übersichtlich und „versteckte“ bzw. wenig offensichtliche Nebenwirkungen durch Änderungen an globalen Variablen sind unwahrscheinlich). Globale Variablen lassen sich leider nicht in allen Fällen vermeiden. Insbesondere für die Elemente klassischer Dynpros und Selektionsparameter sind sie weiterhin erforderlich.

Eine weitere Ausnahme sind Druckprogramme unter der Technik SAPSCRIPT, welche explizit mit globalen Variablen arbeitet. Hier sollte man aufgrund der Lesbarkeit immer mit Rückgabeparametern arbeiten, welche dann entsprechend in globalen Variablen verfügbar gemacht werden.

Vermeidung von Code-Kopien

Doppelter oder mehrfach existierender Quellcode (Klone) erschwert die Fehlerbehebung, Funktionsänderung oder -erweiterung, da sichergestellt werden muss, dass alle Kopien gefunden und angepasst werden. Hinzu kommt, dass die Kopien üblicherweise nicht mehr 100%ig identisch sind und mit viel Mühe herauszufinden ist, ob und wo sie sich unterscheiden.

Ein Sonderfall sind Kopien des SAP-Standards. Siehe hierzu Kapitel 8.4 Anpassung der SAP-Funktionalität.

BEST PRACTICE

Befolgen Sie das DRY-Prinzip: „Don't repeat yourself"[13] und lagern Sie vor der erneuten Verwendung den Quellcode in eine separate Methode aus.

Anmerkungen

Eine etwas großzügigere Formulierung des DRY-Prinzips ist die Dreierregel[14], nach der die erste Kopie noch erlaubt ist.

Bei der Suche von Code-Kopien[15] sollte generierter Quellcode ausgenommen werden, da es sich hier um bewusste Kopien handelt.

Mehrere Anweisungen in einer Zeile, Method Chaining

Um die Lesbarkeit des Quellcodes zu erhöhen, empfehlen wir, auf mehrere Anweisungen in einer Codezeile zu verzichten.

Eine Ausnahme stellt das Method Chaining dar. Durch die Verkettung von Methodenaufrufen können Variablen, die nur zum Halten von Zwischenergebnissen eingeführt würden, vermieden und der Quellcode so lesbarer gestaltet werden. Eine sehr tiefe Verkettung von Methoden kann jedoch insbesondere in Kombination mit wenig sprechenden Methodennamen die Lesbarkeit reduzieren. Eine pauschale Richtlinie kann daher nicht gegeben werden.

2.4 TRENNUNG VON PRÄSENTATIONS- UND ANWENDUNGSLOGIK

In allen Programmen sollte stets eine Trennung von Präsentations- und Anwendungslogik erfolgen. Dies erlaubt es, Ergebnisse und Funktionen der Anwendungslogik durch verschiedene User Interfaces (UIs) dem Benutzer anzuzeigen sowie über eine einheitliche Schnittstelle anderen Systemen bereitzustellen. Diese Aussage ist für alle gängigen UI-Technologien gültig, wobei der Grad der Unterstützung bzw. Einhaltung dieser logischen Trennung unterschiedlich ist. In einer Floorplan-Manager/Web-Dynpro-ABAP-Realisierung ist schon vom Framework eine Trennung zwischen Modell- und UI-Logik vorgesehen. Bei klassischen Dynpros und BSPs wird die Trennung nicht in gleicher Weise forciert, aber grundsätzlich kann und sollte die Trennung auch in diesen Umgebungen umgesetzt werden. Allerdings gibt es hierfür keine technische Prüfung wie bei Floorplan-Manager/Web Dynpro, für den entsprechende Prüfungen im Code Inspector realisiert sind. Die gleichen Grundsätze gelten natürlich auch bei der UI5-Entwicklung (siehe Kapitel 10), bei der die Oberflächentechnologie bereits das Model-View-Controller-Muster vorsieht und die Applikationslogik vom jeweiligen Backend zur Verfügung gestellt wird.[16]

Ein typisches Beispiel für eine klare Trennung von Anwendungslogik und UI sind Plausibilisierungsregeln. Wenn die Plausibilisierung von Eingaben in einer bestimmten UI-Technologie (in der Präsentationsschicht) entwickelt wird, müssen diese Prüfungen bei einem Wechsel auf eine andere UI-Technologie neu entwickelt werden. Um dies zu vermeiden, sollten die Funktionen zur Prüfung von Eingaben oder Parametern unabhängig von der verwendeten UI erstellt und gepflegt werden.

Meist ist es darüber hinaus auch sinnvoll, den Quellcode für das Datenmodell bzw. die Datenbankzugriffe getrennt zu halten, sei es (bei kleinen, nicht auf Wiederverwendung zielenden Entwicklungen) über eine separate lokale Klasse, oder über Frameworks wie BOPF (Business Object Processing Framework[17]). Dies entspricht dem klassischen Model-View-Controller-Muster.

13 Vgl. https://de.wikipedia.org/wiki/Don%E2%80%99t_repeat_yourself
14 Vgl. http://en.wikipedia.org/wiki/Rule_of_three_(programming)
15 Zum Beispiel mit dem SAP Clone Finder, Transaktion CCAPPS

16 SAP-Roadmap für die Oberflächenstrategie
17 http://scn.sap.com/community/abap/bopf

2.5 INTERNATIONALISIERUNG

Sprachabhängige Texte in Programmen dürfen nicht „hart codiert" werden, sondern müssen in Textelementen (Programmtexte, Klassentexte, Online-Text-Repository [OTR]), Standardtexten oder Nachrichtenklassen hinterlegt werden. Da alle Eigenentwicklungen den Anspruch haben sollten, weltweit eingesetzt zu werden, sollten alle Texte in die jeweils wichtigsten Sprachen übersetzt werden.

Konfigurierbare sprachabhängige Texte werden in eigenen Texttabellen abgelegt. Eine solche Texttabelle besitzt dieselben Schlüsselattribute wie die zugehörige Customizing-Tabelle und verweist auf diese über eine Fremdschlüsselbeziehung. Zusätzlich muss das erste Schlüsselattribut nach dem Mandantenfeld das Sprachenattribut sein (Datenelement SPRSL oder SPRAS).

BEST PRACTICE

- Wir empfehlen, den Code Inspector bzw. ATC für die Suche nach nicht übersetzbaren Texten zu verwenden.
- Um spätere Übersetzungen einfach zu gestalten, sollte die Länge der Feldbezeichner und Textelemente möglichst lang gewählt werden. Als Faustregel für die Länge von Textelementen hat sich bewährt, die 1,5-fache Länge der nativen Beschreibung vorzusehen.

Um den Aufwand für Übersetzungen zu reduzieren, sollten bei der Festlegung einer Übersetzungsstrategie die folgenden Aspekte berücksichtigt werden:

- Nutzung der Sprache Englisch als zentrale Originalsprache
- Definition einer Übersetzungstiefe (z.B. nur Oberflächentexte, Oberflächentexte und F1-Hilfen, Komplettübersetzung) mit Orientierung am SAP Standard Translation Level[18]
- Zuordnung der Pakete zur SAP-Anwendungshierarchie, um für die Verteilung von Toptexten domänenspezifische Textvorschläge nutzen zu können
- Kennzeichnung technischer Texttabellen, die nicht übersetzungsrelevant sind
- Nutzung von Langtextobjekten (z.B. via SO10) statt der Aufteilung von Texten in mehrere Zeilen
- Verwendung einer einheitliche Terminologie und Schreibweise von Bezeichnern mit Hilfe der Terminologiedatenbank
- Vermeidung anonymer Platzhalter (&) in Nachrichtentexten (stattdessen Nutzung von &1 &2 &3 &4, da deren Positionierung im Nachrichtentext in verschiedenen Sprachen unterschiedlich sein kann)
- Arbeiten mit Auffüllsprachen in Systemen ungleich dem Entwicklungssystem für die Auffüllung von Übersetzungslücken in den Zielsprachen Einsatz der Pseudosprache 2Q für technische Oberflächentests[19]
- Verwendung des SAP-Paketkonzeptes für die Trennung von übersetzungsrelevanten von nicht übersetzungsrelevanten Elementen

WEITERE QUELLEN

1. Vortrag „Best Translation Practices in SAP Custom Development Projects" von Lindsay Russel (SAP SE)

18 Vgl. https://websmp102.sap-ag.de/~form/handler?_APP=00200682500000002672&_EVENT=DISPLAY&_SCENARIO=011000358700000001 22&_HIER_KEY=501100035870000008578&_HIER_KEY=601100035870000248115&

19 http://www.se63.info/pseudo-localization-sap-applications/

2.6 DYNAMISCHE PROGRAMMIERUNG UND AUDITIERBARKEIT

Dynamische Programmierung

In der „klassischen", statischen Entwicklung werden Entwicklungsobjekte und Quellcode zur Designzeit definiert und im SAP-System statisch hinterlegt. Zur Laufzeit wird der vorgegebene Quellcode ausgeführt. Im Gegensatz dazu ermöglicht die dynamische Programmierung die Flexibilisierung des Quellcodes. Das nachfolgende Beispiel verdeutlicht die dynamische Programmierung:

Im Quellcode wird der Name einer aufzurufenden ABAP-Klasse nicht statisch hinterlegt, sondern es wird zur Laufzeit die Klasseninstanz aufgerufen, deren Name durch den Inhalt einer Variablen oder durch eine Customizingeinstellung definiert ist. Der Name und damit die konkret ausgeführte Implementierung kann z.B. aufgrund von Benutzereingaben variieren.

Der Vorteil dieser Methodik liegt in der gesteigerten Flexibilität und ihr Nachteil in der erhöhten Komplexität. Je nach Art der dynamischen Programmierung können auch Sicherheitsrisiken damit verbunden sein. Tools zur automatischen Erkennung von Sicherheitsrisiken funktionieren für dynamischen Quellcode schlechter.

Vorteile:

- Steigerung der Flexibilität
- Steigerung der Wiederverwendbarkeit
- Vermeidung von Kopien und damit Verkleinerung der zu wartenden Quellcode-Menge

Beispiele zu den Vorteilen:

Beispiel 1 – Eigener Aufbau von User Exits

Durch die statische Definition einer abstrakten Klasse inkl. Methodensignatur wird das Grundgerüst für einen „User Exit" vorgegeben. Anschließend können mehrere konkrete Implementierungen der abstrakten Klasse angelegt werden. Zur Laufzeit wird z.B. aus einer Customizing-Tabelle der Name der zu verwendenden konkreten Klassenimplementierung gelesen und diese aufgerufen. Somit können unterschiedliche Implementierungsvarianten per Customizing aktiviert/deaktiviert werden.

Beispiel 2 – Dynamische WHERE-Bedingung

Zur Laufzeit wird die WHERE-Bedingung für eine Datenbankoperation, z.B. SELECT, in einer String-Variablen erstellt. Dadurch können komplizierte CASE-Abfragen vermieden werden, die abhängig von den Eingaben verschiedene Open-SQL-Befehle ausführen.

Beispiel 3 – Ersetzen von sogenanntem Boilerplate Code [20]

Dynamische Programmierung kann auch dazu verwendet werden, ähnliche Logik nicht immer wieder in nahezu identischen Varianten neu zu implementieren. Ein Beispiel wäre eine Datenbankzugriffsschicht: die Tabellen unterscheiden sich zwar, der Algorithmus ist aber immer der gleiche. In diesem Fall lässt sich das Problem mittels dynamischer Programmierung oder Generierung des Quellcodes auflösen. Dies ist mit einem Mehraufwand verbunden, der sich je nach Szenario aufgrund der besseren Wartbarkeit wieder amortisiert.

Nachteile:

- Durch die Nutzung von dynamischen Aufrufen geht der Verwendungsnachweis innerhalb der ABAP-Entwicklungsumgebung verloren. Problematisch sind dann Änderungen an den Aufrufzielen.
- Bei der dynamischen Programmierung ist in der Regel zur Designzeit keine syntaktische Prüfung möglich. Dies kann bei fehlerhafter Belegung der variablen Inhalte (z.B. fehlerhafte Klammerung innerhalb einer dynamischen WHERE-Klausel, fehlerhafter Name einer Klasse) einen Abbruch bei der Programmausführung verursachen.
- Die dynamische Programmierung birgt hohe Sicherheitsrisiken, wenn die dynamischen Inhalte durch ungeschützten Zugriff beeinflussbar sind (z.B. wenn eine WHERE-Bedingung durch Benutzereingaben beeinflusst werden kann; Stichwort: SQL-Injection).
- Die Komplexität des Quellcodes erhöht sich.

20 https://de.wikipedia.org/wiki/Boilerplate#Programmierung

BEST PRACTICE

- Dynamische Programmierung sollte nur sehr dosiert und kontrolliert zum Einsatz kommen. Quellcode, der dynamische Anteile enthält, sollte nach dem Vier-Augen-Prinzip kontrolliert und dokumentiert werden, denn er stellt ein potenzielles Sicherheitsrisiko dar.[21]
- Für generierten Quellcode sollte nach der Generierung eine Syntaxprüfung mit dem Befehl SYNTAX-CHECK durchgeführt werden.
- Für dynamischen oder generierten Quellcode sollten die Prüfmechanismen der Klasse CL_ABAP_DYN_PRG genutzt werden, mit denen z.B. Whitelists für dynamische Tabellenzugriffe verwendet werden können.

Im Kapitel 5.2 wird das Thema dynamische Programmierung auch im Kontext Sicherheit behandelt.

Auditierbarkeit von ABAP-Quellcode

Es muss jederzeit möglich sein, durch manuelle Untersuchungen oder statische Codeanalyse-Tools den selbst geschriebenen ABAP-Quellcode auf Mängel zu untersuchen. Alle Methoden, ABAP-Quellcode unsichtbar zu machen, sind unzulässig. Sie behindern solche Untersuchungen und können sogar gezielt dafür verwendet werden, Hintertüren in ein System zu schleusen. Verschleierter Quellcode (z.B. Makros) kann mit dem Debugger nicht (mehr) untersucht werden, was neben der Auditierbarkeit auch die Fehleranalyse erschwert. Es wird an dieser Stelle explizit darauf verzichtet, die Techniken zu erläutern, mit denen Quellcode versteckt werden kann.

BEST PRACTICE

- Verwenden Sie keine Techniken, um Ihren Quellcode zu verstecken, und vereinbaren Sie dies auch in Verträgen mit externen Entwicklern.
- Aufgrund von möglichen Problemen bei Wartungsaktivitäten durch unterschiedliche Entwickler raten wir davon ab, das Sperrkennzeichen für die Änderbarkeit von Quellcode zu nutzen. Alternativ lässt sich der Schutz des Quellcodes durch dessen Auslagerung in separate Pakete erreichen, wobei die Berechtigung auf bestimmte Personen über das Berechtigungsobjekt S_DEVELOP und einer expliziten Ausprägung von DEVCLASS eingeschränkt werden kann.

2.7 NEUE SPRACHELEMENTE

Mit NetWeaver 7.40 und 7.50 wurde die ABAP-Syntax stark erweitert.[22] Die Neuerungen lassen sich grob in zwei Themen aufteilen:

1. Ermöglichung der in vielen anderen Programmiersprachen üblichen Ausdrucksorientierung

Diese ersetzt die bisherige Anweisungsorientierung, die noch aus den ABAP-Ursprüngen als Makro-Sprache herrührt. Die ABAP-Entwickler sollen sich bei der Programmierung auf das, was gemacht werden soll, konzentrieren können, anstelle auf das wie (siehe die vielen Hilfskonstrukte im alten Quellcode im unten nachfolgenden Beispiel).

2. Unterstützung des „Code2Data"-Paradigmas

Gemeint ist die Verlagerung von datenintensiven Operationen vom Anwendungsserver in die Datenbank, was auch als „code pushdown" bezeichnet wird. Die zugehörigen Spracherweiterungen sind teilweise datenbankübergreifend

21 *Zum Auffinden dynamischen Quellcodes (Vorauswahl für die Vier-Augen-Prüfung) können die Sicherheitsprüfungen des Code Inspectors / ATC verwendet werden.*

22 http://scn.sap.com/community/abap/blog/2013/07/22/abap-news-for-release-740
http://scn.sap.com/community/abap/blog/2015/11/27/abap-language-news-for-release-750

(Erweiterungen OpenSQL, Core Data Service -CDS Views[23]), teilweise HANA-spezifisch (AMDP – ABAP Managed Database Procedures). Für den Einsatz der HANA-spezifischen Sprachelemente muss zwischen universeller Lauffähigkeit auf allen Datenbanken gegenüber Performanceoptimierung für SAP HANA abgewogen werden.

Um einen Eindruck von den neuen Möglichkeiten der Ausdrucksorientierung aus Punkt 1 zu geben, hier ein Beispiel aus dem Blog von Horst Keller (siehe unten):

Quellcode in NetWeaver 7.0

```
DATA itab TYPE TABLE OF scarr.
SELECT * FROM scarr INTO TABLE itab.

DATA wa LIKE LINE OF itab.
READ TABLE itab WITH KEY carrid = 'LH' INTO wa.

DATA output TYPE string.
CONCATENATE 'Carrier:' wa-carrname INTO output SEPARATED BY space.

cl_demo_output=>display( output ).
```

Entsprechender Quellcode in 7.40 (SP08)

```
SELECT * FROM scarr INTO TABLE @DATA(itab).
cl_demo_output=>display( |Carrier: { itab[ carrid = 'LH']-carrname }| ).
```

Folgende Spracherweiterungen scheinen uns besonders elementar:

Mit SAP NetWeaver 7.02 eingeführt:

- Inline-Berechnungen und Inline-Methodenaufrufe
- Zeichenketten-Templates (NW 7.40 Beispiel Zeile 2: Parameter der Display-Methode)
- Zeichenkettenfunktionen und Reguläre Ausdrücke

Mit SAP NetWeaver 7.40:

- Inline-Deklarationen (in Zeile 1: DATA(itab))
- Konstruktor-Operatoren VALUE und NEW
- Tabellenausdrücke (in Zeile 2: itab[...])
- Neuer Parser für OpenSQL (in Zeile 1: @ als Kennzeichnung für Host-Variablen)
- SQL-Ausdrücke

BEST PRACTICE

Wir empfehlen, sich mit den neuen Sprachelementen vertraut zu machen und sie einzusetzen, da die Effektivität bei der Entwicklung erhöht und die Lesbarkeit des Quellcodes verbessert werden kann. Hierbei kann die Liste oben als Einstieg dienen.

23 *In NetWeaver 7.40 noch mit Einschränkungen (parametrisierte Views nur auf SAP HANA), ab NetWeaver 7.50 sind die CDS-View-Funktionalitäten datenbank-agnostisch*

In diesem Zusammenhang sind auch die Skills des Teams und die eingesetzten SAP-Basisversionen in der Systemlandschaft zu berücksichtigen. Ältere SAP-Versionen unterstützen nicht alle neuen Sprachkonstrukte und auch nicht das Code2Data-Paradigma. Arbeiten die Entwickler oft in unterschiedlichen Systemen mit unterschiedlichen SAP-Basisversionen, sollte ggf. deren kleinster gemeinsamer Nenner verwendet werden, um nachträglichen Mehraufwand bei der Übertragung in ältere Systeme zu vermeiden.

Weitere Informationen zu den neuen Sprachelementen finden Sie in der ABAP-Schlüsselwort-Dokumentation und den SCN-Blogs von Horst Keller.[22]

2.8 OBSOLETE ANWEISUNGEN

SAP vertritt eine strikte Abwärtskompatibilität. Trotzdem muss bei der Verwendung von obsoleten Anweisungen, wie z.B. Kopfzeilen in internen Tabellen, berücksichtigt werden, dass bei der Übernahme von Code in Klassen Probleme auftreten. Für obsolete Sprachelemente gibt es immer modernere Alternativen. Außer Gewohnheit existieren wenige Gründe für deren Verwendung, deshalb sollten sie vermieden werden.

BEST PRACTICE

Wir empfehlen den regelmäßigen Einsatz eines statischen Codeanalyse-Tools, um obsolete Anweisungen zu entdecken. Aus den SAP-Bordmitteln eignet sich der Code Inspector/ATC bzw. die Durchführung des Syntax-Checks. Um den Zusatzaufwand zu minimieren, sollten die obsoleten Anweisungen immer dann ersetzt werden, wenn Entwicklungsobjekte aufgrund anderer Erweiterungen sowieso bearbeitet werden. Dadurch ist auch der zusätzliche Testaufwand in der Regel gering.
Darüber hinaus existieren sehr gute Analysewerkzeuge von Drittanbietern.

2.9 AUTOMATISCHE PRÜFUNGEN DER ENTWICKLUNGSOBJEKTE

Für die automatische Überprüfung von Entwicklungsobjekten zur Designzeit bietet SAP verschiedene Werkzeuge an:

- Die einfache Syntaxprüfung wird automatisch bei der Aktivierung ausgeführt und verhindert die Aktivierung fehlerhafter Entwicklungsobjekte.
- Die erweiterte Programmprüfung kann auf individuellem, bereits aktiviertem Quellcode (Programmen, globalen Klassen ...) ausgeführt werden und berichtet in drei Prioritäten potenzielle Probleme.
- Der Code Inspector bietet einen umfangreichen Katalog von teilweise konfigurierbaren Prüfungen, aus denen eine Prüfvariante zusammengestellt werden kann. Die erweiterte Programmprüfung ist hier enthalten. Es gibt unter anderem auch etliche Prüfungen zu den Bereichen Performance und Robustheit. Zusätzlich können eigene Prüfungen programmiert und eingebunden werden.
- Das neueste Werkzeug ist das ABAP Test Cockpit (ATC)[24], welches die Prüfvarianten des Code Inspectors verwendet und einige Zusatzfunktionen bietet.

Diese Werkzeuge können von den Entwicklern bereits während der Entwicklung genutzt werden. Alle sind gut in die ABAP Workbench und die ADT integriert (Ausnahme: Die erweiterte Programmprüfung ist von ADT nicht direkt erreichbar).

Zusätzlich kann der Code Inspector oder das ATC über die Transaktion SE03 global angeschaltet werden, um bei der Freigabe von Transportaufträgen frühzeitig Probleme oder Schwächen zu erkennen und zu beheben. So kann unnötiger Aufwand für Transporte, Test und Fehlersuche eingespart und die Programme performanter, robuster, sicherer und besser wartbar gemacht werden.

Der Code Inspector oder das ATC werden im SAP-Standard nur bei Freigabe des Transportauftrags ausgeführt. Empfehlenswert (und bei Transportmanagement mit Transporten von Kopien – siehe Abschnitt 8.1.4 – die einzige sinnvolle Möglichkeit) ist die Prüfung bereits bei der Freigabe der jeweiligen Transportaufgabe bzw. bereits bei der Fertigstellung der Entwicklungsobjekte durch den Entwickler. Hierzu muss das BAdI CTS_REQUEST_CHECK implementiert werden.[25]

24 Verfügbar ab NetWeaver 7.4, sowie ab bestimmten SPs von 7.0 und 7.3.
25 Wie dies für den ATC geschehen kann, ist hier beschrieben.

BEST PRACTICE

Viele der in diesem Leitfaden genannten Regeln und Empfehlungen für Entwicklungsobjekte lassen sich durch die Werkzeuge automatisch prüfen. Die Werkzeuge sollten daher den Entwicklern vertraut sein und frühzeitig im Entwicklungsprozess verwendet werden. Die generelle Prüfung mit Code Inspector oder ATC bei Transportfreigabe sollte konfiguriert werden. Als Voraussetzung hierfür muss eine Prüfvariante ausgewählt oder selber konfiguriert werden. Zusätzlich ist festzulegen, wer bei fehlgeschlagenen Prüfungen Ausnahmen genehmigen darf (z.B. Qualitätssicherungs-Beauftragter oder Entwicklungsleiter).

Zusätzlich existieren diverse Tools von Drittanbietern, die bereits zur Designzeit den Quellcode auf Verstöße gegen Entwicklungsrichtlinien überprüfen und dem Entwickler ein direktes Feedback über bestimmte Eigenschaften der Codequalität liefern. Neben der Prüfung zur Designzeit bieten die Tools zum Teil auch die Möglichkeit der automatischen Korrektur von Fundstellen an.

Kapitel 7, insbesondere Abschnitt 7.2.2 Automatische Prüfungen enthält übergreifende Empfehlungen zu diesem Themenbereich.

WEITERE QUELLEN:

Die Vorgehensweise der Implementierung des BAdIs CTS_REQUEST_CHECK für den Code Inspector ist im Buch „Praxishandbuch SAP Code Inspector“ (SAP Press) beschrieben, oder auch in diesem Blog.
Das Buch beschreibt auch die Integration eigener Prüfungen in den Code Inspector / ATC. Auch hierzu gibt es einen Blog.

2.10 HARTE CODIERUNG, MAGIC NUMBERS

Unter problematischer Harter Codierung versteht man die Codierung von Daten wie Texten, Zahlen, Benutzernamen usw. direkt im Programm, die sich möglicherweise ändern, z.B. weil sie eigentlich Konfigurationsinformationen (Pfade für Schnittstellen, ...) sind. Harte Codierung spart zunächst Entwicklungszeit, führt aber auf den Gesamtlebenszyklus der Anwendung bezogen zu höheren Kosten, weil z.B. Konfigurationsänderungen eine Anpassung des Programms erfordern. Wenn der gleiche Wert in mehreren Programmen oder Programmteilen hart kodiert wird, ergibt sich zusätzlich noch das Problem, dass bei Änderungen alle Stellen angepasst werden müssen (siehe auch „Vermeidung von Code-Kopien“ in Abschnitt 2.3).

Ein verwandtes Thema sind Magic Numbers, also Zahlen, die ohne Erläuterung im Programm verwendet werden. Das Problem hierbei ist die Verständlichkeit und die Wartbarkeit des Programms. Auch hier wird also zunächst Aufwand gespart, der aber bei späteren Anpassungen doch noch aufgewendet werden muss und ggf. sogar höher ist, falls die Bedeutung der Magic Number erst ermittelt werden muss.

Ein weiterer Aspekt bei der Verwendung von Harter Codierung oder Magic Numbers ist, dass dies auch ein Hinweis auf eine Implementierung ohne Berücksichtigung objektorientierter Konzepte sein kann. Ein Beispiel wären CASE Statements über Konstanten, die entweder über die Refactoring Methodik „Replace Code with Subclasses“[26] oder via Filter-BAdIs vermieden werden können[27].

BEST PRACTICE

Problematische Harte Codierung sollte vermieden werden. Magic Numbers sollten durch sprechend benannte Konstanten ersetzt oder mit einem entsprechenden Kommentar versehen werden.

Die Benutzung von Konstantennamen, die identisch mit dem Wert sind, sollte unbedingt vermieden werden. Sie führen bei einer Änderung des Wertes zu Fehlinformationen. (Paradoxerweise ist genau die Änderbarkeit der Hauptgrund für die Verwendung von Konstanten.)

26 *https://sourcemaking.com/refactoring/replace-type-code-with-subclasses*
27 *siehe dazu auch http://scn.sap.com/docs/DOC-10286*

Die Definition von Konstanten sollte als Attribute an einem dediziert dafür vorgesehenen Interface oder einer abstrakten, finalen Klasse erfolgen. Von der Verwendung von Konstantenincludes raten wir ab. Im Rahmen einer größeren Entwicklung kann es sinnvoll sein, nicht alle Konstanten an einem Interface zu hinterlegen, sondern eine fachliche Strukturierung der Interfaces und der enthaltenen Konstanten vorzunehmen. Dafür bietet es sich an, die Interfaces mit einem Tag-Interface zu versehen (analog zum BAdI Interface IF_BADI_INTERFACE). So ist die Auffindbarkeit der Interfaces sichergestellt, die Konstanten beinhalten.

Bei der Einführung von Konstanten ist sicherzustellen, dass diese auch von anderen Entwicklern wiedergefunden werden, da ansonsten identische Konstanten mehrmals im System hinterlegt werden. Hierfür gibt es leider kein offizielles SAP-Tool, das dies automatisch bewerkstelligt. Alternativ kann auf das SAP-Ecosystem zurückgegriffen und das frei verfügbare Tool ConSea[28] verwendet werden. Eine Option wäre die Dokumentation der Konstanten in Interfaces außerhalb des Systems, wobei die Wahrscheinlichkeit hoch ist, dass die Pflege der externen Dokumentation vernachlässigt wird. Ab NetWeaver 7.40 empfiehlt sich die Inline-Dokumentation mittels ABAP-Doc. Ab NetWeaver 7.50 lässt sich die Dokumentation dann analog zu JavaDoc exportieren und so zugänglich machen. Falls die TREX-Suche bereits eingesetzt wird, könnte auch diese zur performanten Suche genutzt werden.

2.11 BERECHTIGUNGSPRÜFUNG IM QUELLCODE

Beim Zugriff auf Daten sowie bei ihrer Präsentation sind die dafür notwendigen Berechtigungsobjekte zu prüfen. Bei der Verwendung von Standardobjekten soll die Prüfung auf die entsprechenden SAP-Standard-Berechtigungsobjekte erfolgen (vereinfacht die Wartung der notwendigen Rollen). Zur Prüfung kundeneigener Datenobjekte können in der Regel keine SAP-Standard-Berechtigungsobjekte verwendet werden. Zu diesem Zweck können kundeneigene Berechtigungsobjekte implementiert und geprüft werden. Weitere Informationen finden Sie im Kapitel 5.1.1.

2.12 PROGRAMMIERMODELL: OBJEKTORIENTIERT VS. PROZEDURAL

Es ist ratsam, vom prozeduralen Programmiermodell auf objektorientierte Programmierung überzugehen, um zukunftssicher zu entwickeln und Objekte zu kapseln. Insbesondere bei neuen Projekten sollte nur noch objektorientiert entwickelt werden.

Objektorientierte Entwicklung wurde so konzipiert, dass logisch zusammenhängende Aufgaben einheitlich in Objekten zusammengefasst werden können. Dadurch wird unter anderem auch die Wiederverwendbarkeit von Quellcode erhöht. Insbesondere können solche Objekte auch von anderen Entwicklern für ihre Zwecke leicht erweitert oder verändert werden, ohne dass die grundlegenden Funktionen beeinträchtigt werden (Open-Closed-Prinzip). Andererseits können zentrale Funktionalitäten oder einzelne Variablen auch gezielt vor unerwünschtem Lese- oder Schreibzugriff durch aufrufende Programme geschützt werden. Einen ersten Überblick über die Grundprinzipien des objektorientierten Designs finden Sie z.B. in Wikipedia[29].

Im Rahmen der Einführung der Objektorientierung in ABAP mit ABAP Objects fand eine Bereinigung der Sprache und Vereinheitlichung der Konstrukte statt. Die Verwendung von ABAP Objects führt damit zu einer Steigerung der Wartbarkeit. Die Steigerung der Wartbarkeit kann für kleinere Programme[30] bereits durch den Ersatz von FORM-Routinen durch Klassen-Methoden einer lokalen Klasse lcl_main erreicht werden, ohne ein striktes objektorientiertes (Re-)Design vorzunehmen.

Die prozedurale Entwicklung in ABAP z.B. mit FORM-Routinen wurde von SAP inzwischen als obsolet eingestuft. Eine prozedurale Entwicklung hat sich im Laufe der Jahre auch im Hinblick auf globale Variablen und Includes als sehr unübersichtlich, komplex und fehleranfällig erwiesen. Das ist ein weiterer Grund, zur objektorientierten Programmierung zu wechseln.

BEST PRACTICE

Wir empfehlen Neuentwicklungen möglichst nur noch nach den Prinzipien der Objektorientierung mit ABAP Objects umzusetzen. Obsolete Sprachkonstrukte des prozeduralen ABAP, wie z.B. FORM-Routinen, sollten vermieden und durch ABAP Objects-Konstrukte ersetzt werden. Diese Empfehlung deckt sich mit der Vorgabe „ABAP Objects als Programmiermodell“ der Programmierrichtlinien aus der ABAP-Schlüsselwortdokumentation.

28 *Vgl. GitHub*

29 *https://de.wikipedia.org/wiki/Prinzipien_objektorientierten_Designs*

30 *Die Obergrenze für „kleine Programme“ in diesem Sinne sehen wir zwischen 200 und 500 Zeilen, je nach Programmierstil und Komplexität der Aufgabe.*

Mögliche Gegengründe:

- Verfügbare ABAP Objects-Kompetenz im Team sowie der geplante und realistisch erwartete Kompetenzaufbau
- Systeme mit älteren Basisreleases, in denen ABAP Objects noch weniger ausgereift war (vor SAP Web Application Server 6.20) bzw. in Systemen/Modulen, in denen auch im SAP-Standard prozedurale Bestandteile dominieren.

WEITERE QUELLEN

1. Horst Keller und Gerd Kluger, Not Yet Using ABAP Objects? Eight Reasons Why Every ABAP Developer Should Give It a Second Look, Sap Professional Journal
2. Horst Keller and Gerd Kluger, NetWeaver Development Tools ABAP, SAP AG
3. Bertrand Meyer, Objektorientierte Softwareentwicklung, Hanser 1990, ISBN 3-446-15773-5
4. Wikipedia: Übersicht über Entwurfsmuster

2.13 ENTWICKLUNGSSPRACHE

Alle Entwickler sollten sich mit der gleichen Sprache anmelden und sprachabhängige Texte in dieser Sprache pflegen. Grundsätzlich sollten sprachabhängige Texte in übersetzbarer Form angelegt werden.

3 PERFORMANCE

In den folgenden Abschnitten empfehlen wir einige Best Practices, die bei der täglichen Arbeit in der ABAP-Entwicklung beachtet werden sollten, um für die zu entwickelnde Anwendung eine gute Performance sicherzustellen. Sofern sich beim Einsatz der SAP-HANA-Datenbank gegenüber anderen Datenbanken Besonderheiten ergeben, wird hierauf in den jeweiligen Unterkapiteln hingewiesen.

3.1 VERMEIDUNGSPRINZIP

„Die sichersten, schnellsten, präzisesten, billigsten, wartbarsten, zuverlässigsten und am leichtesten zu dokumentierenden Teile eines Computersystems sind die, die man weggelassen hat." (Gorden Bell)

BEST PRACTICE

Vermeiden Sie jeglichen unnötigen Quellcode. Dies umfasst auch unnötig durchlaufenen Quellcode.

3.2 PERFORMANCE-OPTIMIERUNGEN NUR AN RELEVANTEN STELLEN

Performance-Optimierungen sollten (insoweit sie Aufwand oder Komplexität erhöhen) nur an relevanten Stellen erfolgen. Um an diese „Hot Spots" zu gelangen, muss zuvor die Performance gemessen werden.

BEST PRACTICE

- Konzentrieren Sie sich zuerst auf die klare und einfache Implementierung der Sachlogik. Führen Sie dann auf einem System mit genügendem Datenvolumen (meist das Testsystem) Performance-Tests aus, um – falls das Programm überhaupt langsamer als vom Nutzer gewünscht ist – anschließend an den relevanten Stellen die Performance zu optimieren. Eine Ausnahme stellen Programme dar, bei denen von vornherein klar ist, dass sie Performance-kritisch sein werden. Für diese sind schon zur Designzeit Entscheidungen zu treffen, z.B. Einsatz von Parallelisierung.
- Wir empfehlen, die Suche nach Performance-Engpässen mit dem Einsatz der Laufzeitanalyse (Transaktion SE30/SAT) mit voller Aggregation zu beginnen. Dadurch sollte deutlich werden, ob die Laufzeit aus der Interaktion mit der Datenbank oder der Verarbeitung der geladenen Daten im Hauptspeicher resultiert.[31] Wichtig ist dabei, dass ein repräsentativer, praxisnaher Datenbestand verarbeitet wird, um nicht durch seltene Verarbeitungsmuster einer falschen Spur zu folgen. Wird mehr als die Hälfte der Laufzeit im DB-Teil verbraucht, sollte eine genauere Analyse der SQL-Kommandos mit der Transaktion ST05 erfolgen. Wird mehr Laufzeit im ABAP-Teil verbraucht, erfolgen tiefergehende Analysen mit Hilfe der Transaktionen SE30/SAT, wobei die Aggregationsstufen schrittweise verringert werden, um genauere Aussagen über die kritischen Programmstellen zu bekommen. Nach jedem Optimierungsschritt sollten die Ergebnisse verglichen und dokumentiert werden.

3.3 VORHANDENE WERKZEUGE NUTZEN

Die im SAP-System bereits vorhandenen Werkzeuge bieten eine gute Unterstützung bei der Erstellung von performanten Anwendungen bzw. bei der Analyse von Performance-Engpässen.

Die wichtigsten Transaktionen sind

- ATC/SCI – ABAP-Test-Cockpit / Code Inspector
  Statische Analyse von (unter anderem) Performance-Aspekten. Integriert in Workbench und ADT, Administration mit Transaktionen ATC und SCI
- SAT – Laufzeitanalyse
  Gesamtanalyse zur Laufzeit
- ST05 – Performance-Trace
  Analyse der in einem Programm ausgeführten SQL-Befehle (und anderer Ereignisse)

Darüber hinaus gibt es einige allgemeine, auch für Performance-Analysen nutzbare Werkzeuge

- SM50/SM66 – Übersicht Workprozesse
- Debugger – Schrittweise Ausführung von Quellcode
- ST03 – Systemlastmonitor
- ST22 – Analyse von Laufzeitfehlern (z.B. Speichermangel)
- STAD – Workload-Analyse (Post Runtime)

Die folgenden Werkzeuge eignen sich für spezielle Performance-bezogene Themen

- ST04 – DB-Performance-Übersicht
- DB05 – Wertverteilungs-Analyse für Index-Design

31 *Ein ersten Überblick über die Verteilung der Laufzeiten zwischen Datenbank und Applikationsserver bekommt man auch mittels der Transaktion STAD.*

- ST10 – Tabellenaufruf-Statistiken zur Prüfung von Tabellenpufferung
- ST12 – Single Transaction Analysis (End-to-End Trace)
- S_MEMORY_INSPECTOR – Memory Inspector (auch integriert im SAP GUI Debugger)
- SQLM/SQLMD – SQL Monitor
  Aufzeichnung und Analyse aller SQL-Abfragen im Produktivsystem.
  Neu seit NetWeaver 7.0/7.4 (Details siehe Hinweis 1885926).
- SWLT – Performance Tuning Worklist
  Arbeitsvorrat basierend auf ATC- und SQL-Monitor-Ergebnissen.
  Neu seit NetWeaver 7.0/7.4 (bestimmte SPs, siehe SAP Dokumentation).

Im Kontext der SAP-HANA-Einführung wurden von SAP weitere Tools oder Erweiterungen bestehender Tools angeboten, um Performance-Probleme frühzeitig zu identifizieren und ein Vorgehen zu etablieren, das die iterative Optimierung der Performance von Anwendungen auf SAP HANA ermöglicht.

Die Performance-Analyse in diesem Kontext umfasst eine statische Quellcode-Analyse durch das ATC mit den drei Prüfvarianten FUNCTIONAL_DB, FUNCTIONAL_DB_ADDITIONAL und PERFORMANCE_DB. Diese Varianten führen die statische Codeanalyse unter den Aspekten der Verwendung von SAP HANA als DB durch. Weitere Informationen sind im Hinweis 1912445 – ABAP Kunden Code Migration für SAP HANA – Empfehlungen und Code-Inspektor-Varianten für eine SAP HANA Migration hinterlegt.

Zusätzlich wurde der ABAP-SQL-Monitor eingeführt. Dieses Tool erlaubt das systemweite Tracing aller SQL-Abfragen über ein längeres Zeitintervall und mit zusätzlicher Aufzeichnung der „entry points" (Transaktionen, Reports, usw. in deren Rahmen ein SQL-Befehl ausgeführt wurde). Detailliertere Informationen zum SQL-Monitor finden Sie im SAP Community Network im Dokument „Optimizing Custom ABAP Code for SAP HANA – The New ABAP SQL Monitor".

Die Zusammenführung der Laufzeitsicht des SQL-Monitors sowie der statischen Analyse mittels ATC erfolgt über die SQL Performance Tuning Worklist. Diese ermöglicht den Aufbau eines Arbeitvorrats zur Performance-Optimierung, der nach erfolgter Analyse und Priorisierung abgearbeitet werden kann. Weitere Informationen zum Vorgehen finden Sie im SAP Community Network im Dokument „Best Practice Guide – Considerations for Custom ABAP Code During a Migration to SAP HANA".

3.4 DATENMODELL UND DATENZUGRIFF

3.4.1 DATENMODELL

Der Aufbau des Datenmodells bildet die Basis performanter Anwendungen. Ein mit Augenmaß normalisiertes Datenmodell kann effizienter mit Daten und Indizes arbeiten.[32] Hierzu finden – unabhängig von der Programmiersprache ABAP – die Normalisierungsregeln Anwendung. Darüber hinaus sollten sich bei der Datenmodellierung im Data Dictionary die Datenelemente jeweils auf Domänen beziehen. Dieses führt sowohl zu einer erhöhten Transparenz als auch zu einer verbesserten Wartbarkeit.

3.4.2 DATENBANKZUGRIFFE

Beim Zugriff auf die Datenbank existieren im Wesentlichen fünf Regeln (Quelle: SAP, siehe Referenz in Abschnitt 3.7), die verwendet werden sollten, um eine performante Ausführung von Programmen mit Datenbankzugriff sicherzustellen. Die fünf Regeln sind:

1. Die Treffermenge klein halten – Wenn Sie die Menge der selektierten Daten klein halten, vermeiden Sie Last sowohl auf dem Datenbanksystem als auch im Bereich des Netzwerks bei der Übertragung der Daten auf den Applikationsserver.
2. Die übertragene Datenmenge klein halten – Aufgrund der blockweisen Übertragung der Daten zwischen Datenbank und Applikationsserver empfiehlt es sich die zu übertragende Datenmenge klein zu halten und so die Last auf dem Netzwerk zu verringern.
3. Die Zahl der Datenbankzugriffe klein halten – Durch eine Reduktion der Zugriffe auf das Datenbanksystem können Sie Last vom Datenbanksystem und dem Netzwerk reduzieren, da jeder Zugriff Aufwand für die Verwaltung auf dem Datenbanksystem mit sich bringt.
4. Den Suchaufwand klein halten – Bei der empfohlenen Verwendung von WHERE und HAVING-Klauseln können Sie die Performance weiter optimieren, wenn Sie den Suchaufwand für das Datenbanksystem mittels Verwendung geeigneter Indizes verringern.
5. Die (unnötige) Datenbanklast klein halten – Generell sollte unnötige Datenbanklast vermieden werden also z.B. die Durchführung von Berechnung auf der Datenbankabfrage, deren Ergebnisse in der Applikation gar nicht benötigt werden. Zusätzlich bietet der Application Server ABAP viele Möglichkeiten die Last der Datenbankressource durch Pufferung klein zu halten.

32 *Im S/4HANA-Modell geht man noch einen Schritt weiter und denormalisiert das Datenmodell. Dies ist aufgrund der Art der Datenablage in der In-Memory-DB HANA möglich und führt zu einem vereinfachten Datenmodell sowie zu einem Performancegewinn bei der Abfrage der Daten.*

Die fünf Regeln gelten für jede Art von Datenbanksystem. Im Kontext des Code Push Downs bzw. des Code-to-Data-Paradigmas mit SAP HANA ist eine geänderte Gewichtung der Regeln im Vergleich zu non-HANA Datenbanken zu beachten. Dies ist in Abschnitt 3.6 erläutert.

Unabhängig vom Code Push Down muss bei der Verwendung von SAP HANA als Datenbank Regel 4 bzgl. der Einführung von Indizes deutlich relativiert werden. In SAP HANA sollten so wenige Indizes wie möglich angelegt werden. Durch die spaltenbasierte Ablage kann im Prinzip jede Spalte als Index verwendet werden und in der Theorie werden daher keine Indizes mehr benötigt. Es gibt allerdings speziell im OLTP-Bereich Szenarien bei denen es auch mit SAP HANA sinnvoll ist, Indizes einzuführen, da es sonst zu einer schlechten Performance und/oder einem hohem CPU-Verbrauch von SAP HANA kommen kann. Details dazu finden Sie im SAP-Hinweis 1794297 – Sekundäre Indizes für die Business Suite auf HANA.[33]

BEST PRACTICES

Spaltenreduzierung

- Vermeiden Sie an voraussichtlich Performance-kritischen Stellen (siehe Abschnitt 3.2), dass unnötige Spalten, vor allem Large Objects (zum Beispiel Strings), von der Datenbank übertragen werden.
- Achtung: Entgegen der verbreiteten Meinung stellen SELECT * und INTO CORRESPONDING FIELDS selbst in diesen Fällen oft kein Problem dar, da das System bereits zur Compile-Zeit die Quell- und Zielfelder intelligent auswertet. [34]

Optimale Suchabfrage

- Versuchen Sie für die Selektion von Daten einen der vorhandenen Indizes vollständig zu nutzen. Ist dies nicht möglich, achten Sie darauf, zumindest die ersten Elemente eines Index zu verwenden, damit die Datenbank den Index bei der Suche nutzen kann.

Zeilenreduzierung

- Nutzen Sie die WHERE-Bedingung, um die an das ABAP-System zu übertragenden Daten zu minimieren. Verwenden Sie SELECT SINGLE / UP TO n ROWS, wenn Sie nur einzelne Zeilen benötigen.

Aggregate

- Aggregate (MIN, MAX, ...) sind in vielen Fällen (bei HANA grundsätzlich) sinnvoll, da sie schon auf dem Datenbankserver ausgewertet werden, und somit weniger Daten übertragen werden müssen. Beachten Sie aber, dass eventuell vorhandene Tabellenpuffer in diesem Fall vom System nicht genutzt werden, was einen gegenteiligen Effekt haben kann.

Updates

- Analog zur Spaltenreduzierung beim Lesen kann mit der Anweisung UPDATE SET die Anzahl der zu schreibenden Spalten reduziert werden. Wenn möglich, sollte diese Anweisung verwendet werden.

33 SAP Hinweis 1794297 – Sekundäre Indizes für die Business Suite auf HANA
34 Vgl. SCN-Artikel „Why ‚INTO CORRESPONDING' is much better than its reputation"

Mengenoperationen

- Jede Ausführung eines Open-SQL-Statements ist mit einem Overhead (Parsen, Abgleich gegen Statement-Puffer im DBMS usw.) verbunden. Daher sollten pro Statement möglichst viele der benötigten Daten auf einmal übertragen werden. Benötigen Sie z.B. die Daten von 50 Aufträgen, sollten Sie diese nicht durch 50 Einzel-Selects ermitteln, sondern durch eine Select-Abfrage. Dies erfolgt durch die Zusätze INTO TABLE bei lesenden bzw. FROM TABLE bei schreibenden Zugriffen (Array-Operation).

Vermeidung von MODIFY

- Setzen Sie MODIFY <DB-Tabelle> nicht ein. Das Statement ist aus Performance-Gesichtspunkten sehr kritisch: Sogar bei der Verwendung von Array-Operationen werden pro Zeile des Arrays zuerst ein UPDATE und im Fehlerfall im Anschluss ein INSERT ausgeführt, was die Performance unnötig verschlechtern kann.

VIEWS/JOINS

- Geschachtelte SELECT-Anweisungen und SELECT-Anweisungen in Schleifen sollten vermieden werden. Stattdessen bieten sich JOINS, Views oder der Zusatz FOR ALL ENTRIES an.
- Sehr umfangreiche JOINS (mehr als 5 Tabellen) sollten vermieden werden, da der DB Optimizer dadurch behindert werden kann.

Bei FOR ALL ENTRIES sollten Sie Folgendes beachten:

- Ist die interne Tabelle der FOR ALL ENTRIES Anweisung leer, wird die WHERE-Bedingung ignoriert und alle Einträge der DB-Tabelle selektiert. Das ist meistens nicht gewünscht und kann zu Performanceproblemen führen. Sie können das Problem durch eine Prüfung der Größe der Tabelle vor dem Ausführen der Abfrage beheben[35].
- Enthält die interne Tabelle der FOR-ALL-ENTRIES-Anweisung doppelte Einträge, kann dies dazu führen, dass Datensätze doppelt von der Datenbank geladen werden bzw. das SELECT-Statement in der DB Schnittstelle unnötig aufgebläht wird. Es empfiehlt sich, ein DELETE ADJACENT DUPLICATES auf der internen Tabelle vor dem Ausführen der Abfrage auszuführen.
- Da bei FOR ALL ENTRIES ein implizites DISTINCT-Select ausgeführt wird, sollten Sie immer alle Schlüsselfelder mit selektieren, da es sonst vorkommen kann, dass nicht alle relevanten Datensätze gelesen werden.

3.4.3 ABAP CORE DATA SERVICE (CDS) VIEWS[35]

ABAP CDS Views sind ein mit NetWeaver 7.40 SP05 eingeführtes DDIC-Artefakt, das im Gegensatz zu klassischen SE11-Views erweiterte Möglichkeiten für die Definition von Views bietet. Dies umfasst neben klassischen SQL-Funktionalitäten wie Outer Joins oder Case-Anweisung auch SQL-Funktionen wie zum Beispiel zur Währungsumrechnung. Des Weiteren bieten CDS Views die Möglichkeit, Assoziationen zwischen Tabellen zu definieren und so Geschäftsobjekte wiederverwendbar in Form von Views zu definieren.

35 *Mit Netweaver 7.40 liefert SAP den Runtime-Check-Monitor, der eine Prüfung zur Laufzeit auf dieses Problem ermöglicht. Siehe dazu auch Hinweis 1931870 – Downport of transaction SRTCM to SP02 / 03 / 04*

Weitere Vorteile sind die Optionen die Views zu parametrisieren (in NetWeaver 7.40 nicht für alle Datenbanken unterstützt) und Annotationen im View bzw. zu View-Feldern zu hinterlegen. Letzteres erlaubt ab NetWeaver 7.50 die einfache Exponierung der Views mittels OData Service und die Nutzung der Views bzw. der Annotationen zum automatischen Aufbau von UI5-Oberflächen (z.B. via Smart Templates). Auf SAP HANA stehen zusätzlich die CDS Table Functions zur Verfügung, die den Aufruf von SQL-Script ermöglichen. So stehen alle SAP HANA Features (z.B. Calculation Views) über CDS Views zur Verfügung. Weitere Informationen zu ABAP CDS Views finden Sie in der ABAP-Schlüsselwertdokumentation zum Netweaver 7.40[36] bzw. NetWeaver 7.50[37] sowie auf der zugehörigen Landingpage im SCN.[38] Beachten Sie, dass ABAP CDS Views nur über die ABAP-Development-Tools in Eclipse erstellt werden können.

3.5 INTERNE TABELLEN UND REFERENZEN

Interne Tabellen stellen ein zentrales Konstrukt bei der Entwicklung von Anwendungen mit ABAP dar. Neben Datenbankzugriffen sind sie die zweite Quelle von Performance-Problemen. Bei kleinen Datenmengen stellen Dinge wie die Wahl der passenden Tabellenart und eines passenden Schlüssels noch kein Problem dar. Werden größere Datenmengen verarbeitet, können vorher aus Performance-Sicht unkritische Stellen zu erheblichen Laufzeitverlängerungen führen.

BEST PRACTICE

Wir empfehlen die Beachtung der nachfolgenden Hinweise zur Steigerung der Performance von Anwendungen:

- Wählen Sie die Tabellenart passend zur späteren Verwendung. Details hierzu finden Sie in der Schlüsselwortdokumentation unter „Auswahl der Tabellenart“. [39]
- Wenn für eine Tabelle vom Typ Sorted oder Hashed Zugriffe erfolgen, sollten diese immer mit dem passenden (Teil-)Schlüssel stattfinden.

Ab AS ABAP 7.02 können Sie bei internen Tabellen, die selten geändert aber mit mehr als einem Zugriffmuster gelesen werden, neben dem primären Schlüssel auch weitere sekundäre Schlüssel definieren. Der primäre Schlüssel wird dabei wie gehabt definiert und verwendet. Die Sekundärschlüssel können einen vom Primärschlüssel abweichenden Typ (Sorted, Hashed) haben.

Als Beispiel können Sie damit für eine als Hashed definierte Tabelle mit eindeutigem Primärschlüssel einen weiteren Schlüssel vom Typ Sorted definieren. Dieser erlaubt einen performanten Zugriff auf die Daten der Tabelle aus einem anderen Blickwinkel (nicht eindeutig, Teilschlüssel möglich), ohne die eigentlichen Daten zweimal im Hauptspeicher anlegen und bei Änderungen manuell für die Konsistenz zwischen den beiden Tabellen sorgen zu müssen.

36 http://help.sap.com/abapdocu_740/de/index.htm?file=abencds.htm
37 http://help.sap.com/abapdocu_750/de/index.htm?file=abencds.htm
38 http://scn.sap.com/docs/DOC-70385

39 *Zugriff wie vorne in Kapitel 2 beschrieben. Pfad: ABAP – Referenz → Interne Daten verarbeiten → Interne Tabellen → Interne Tabellen-Übersicht*

BEST PRACTICE

- Ähnlich wie bei den DB-Zugriffen existieren für interne Tabellen Einzelsatzzugriffe und Massenzugriffe. Wenn möglich, sollten immer die Varianten mit Massenzugriffen gewählt werden, die performanter arbeiten als mehrere Einzelzugriffe.
- Bei der Verwendung der Anweisung SORT sollten Sie eine implizite Sortierung vermeiden und aus Gründen der besseren Nachvollziehbarkeit immer die gewünschten Sortierfelder angeben.
- Vor dem Einsatz der Anweisung DELETE ADJACENT DUPLICATES sollte immer sichergestellt sein, dass die Tabelle nach den gleichen Feldern sortiert ist, damit doppelte Einträge auch wirklich eliminiert werden.
- Reine Existenzprüfungen auf internen Tabellen sollten immer mit READ TABLE TRANSPORTING NO FIELDS durchgeführt werden.[40]

3.5.1 FELDSYMBOLE

Feldsymbole bieten die Möglichkeit, auf existierende Daten, z.B. Zeilen von internen Tabellen, zu referenzieren. Das Arbeiten mit Referenzen ist deutlich performanter als das Kopieren der Daten. Daher sollten, wo möglich, Feldsymbole verwendet werden. Bedenken Sie beim Einsatz von Feldsymbolen jedoch, dass eine Änderung des Wertes eines Feldsymbols auch den Wert im referenzierten Datenelement überschreibt.

BEST PRACTICE

Verwenden Sie standardmäßig Feldsymbole für die Zugriffe auf interne Tabellen.

3.5.2 PARAMETERÜBERGABE

Die Wertübergabe von Parametern sollte nur dort eingesetzt werden, wo es aus technischen Gründen vorgeschrieben ist (z.B. RFC-Funktionsbausteine, Returning-Parameter bei funktionalen Methoden). So werden unnötige Kopierkosten bei der Parameterübergabe gespart. Dies gilt in besonderem Maße bei Parametern mit tiefen Datentypen wie internen Tabellen oder Strings. Weiterhin sollten so wenig Parameter wie möglich definiert werden. Dies erhöht auch die Verständlichkeit des Quellcodes. Die erweiterte Programmprüfung weist auf überflüssige Variablen und Parameter hin, und unterstützt damit das Aufräumen.

BEST PRACTICE

Verwenden Sie so wenig Parameter wie möglich. Nutzen Sie in der Regel die Referenzübergabe und nur in den technisch notwendigen Ausnahmefällen die Wertübergabe.

40 *Ab NetWeaver Release 7.40 SP02 ist für diese Art der Überprüfung die Funktion LINE_EXISTS( ) verfügbar. Sie sollte aufgrund der besseren Lesbarkeit verwendet werden.*

3.6 CODE PUSH DOWN

Beim Einsatz von SAP HANA als Datenbanksystem kann die Performance insbesondere von datenintensiven Applikationen massiv gesteigert werden, wenn die datenintensiven Operationen auf die Datenbank verlagert werden. Die Verlagerung der Logik aus dem Applikationsserver in SAP HANA wird als Code Push Down bezeichnet. Das verfolgte Ziel des Vorgehens ist es, die Daten nicht mehr zur Applikationschicht zu transportieren und dort zu verarbeiten (data-to-code), sondern die Verarbeitungslogik dort zur Ausführung zu bringen, wo die Daten „leben" (code-to-data).

Dieses Ziel in Kombination mit der Architektur von SAP HANA hat zur Folge, dass die in Abschnitt 3.4.1 definierten Regeln im Vergleich zu einer Non-HANA-Datenbank eine andere Gewichtung erfahren: Die ersten drei Regeln (Minimierung der Treffermenge, der übertragenen Daten und der Anzahl der Datenbankzugriffe) gewinnen im Zuge des Code Push Downs an Bedeutung, während die vierte Regel (Optimierung des Suchaufwands) aufgrund der spaltenbasierten Ablage der Daten etwas weniger wichtig wird. Die fünfte Regel lässt sich darauf reduzieren, dass unnötige Last auf der Datenbank vermieden werden sollte, da durch das Vorgehen bewusst datenintensive Operationen auf die Datenbank verlagert und so die Datenbank belastet wird.

Der mit SAP NetWeaver 7.40 bzw. 7.50 erweiterte Sprachumfang von ABAP und Open SQL ermöglicht den Code Push Down im Wesentlichen auf zwei Wegen:

- Verwendung der Open-SQL-Erweiterungen und der ABAP CDS Views
- Verwendung von ABAP Managed Database Procedures (AMDP) und als letzte Option von Native SQL

Welche der beiden Optionen in welchem Umfang genutzt werden, hängt stark von der Zielsetzung ab. In beiden Fällen muss bestehender Code angepasst werden. Die Komplexität der Anpassung ist im Falle der Verwendung von Open SQL und ABAP CDS Views geringer als bei der Verwendung von AMDPs oder Native SQL. Ist das oberste Ziel eine Maximierung der Performance der Anwendung, so wird man mit Open SQL und ABAP CDS Views an Grenzen stoßen, die nur mittels AMDPs oder Native SQL überwunden werden können. Ein weiterer Aspekt ist, dass bei der Verwendung von AMPDs bzw. Native SQL tiefergehende Kenntnisse von SAP-HANA-Spezifika (z.B. SQLScript) notwendig sind, um optimale Ergebnisse erzielen zu können. Generell ist zu beachten, dass die Kompatibilität zu anderen Datenbanken nur bei der Verwendung von Open SQL und ABAP CDS Views (Ausnahme Table Functions) sichergestellt ist. Bei der Verwendung von AMDPs und Native SQL verliert man die Datenbankunabhängigkeit.

BEST PRACTICE

- Wir raten von der Verwendung von External Views und Database Procedure Proxies aufgrund der unterschiedlichen Lebenszyklus Problematik im AS ABAP und SAP HANA Stack ab, sofern dies möglich ist.
- Wenn Sie eine Optimierung Ihrer Anwendung mittels Code Push Down vornehmen wollen, empfehlen wir zur Selektion des zu optimierenden Codes die SQL Performance Tuning Worklist (Transaktion SWLT), die auf Basis der Ergebnisse von statischen Codechecks mittels ATC und Laufzeitinformationen aus dem SQL-Monitor eine priorisierte Liste von Stellen im Quellcode zur Verfügung stellt. Auch hier sind die in Abschnitt 3.3 beschriebenen Empfehlungen zu beachten.
- Wir raten von der Verwendung von Native SQL ab.

WEITERE QUELLEN

1. Einen Einstieg in die Performance-Optimierung bei ABAP bietet der SAP-Kurs BC490.
2. Sehr gute Einführungen in die Performanceanalyse-Tools SAT/SE30 und ATC bieten die Blogposts von Olga Dolinskaja im SAP Community Network
3. Das Profiling in Eclipse ist in dem Blogpost von Thomas Fiedler beschrieben
4. Die 5 Regeln für SQL-Zugriffe finden sich im Original jetzt (7.40) in gestraffter Form in der ABAPDOCU/F1-Hilfe, unter ABAP – Referenz / Externe Daten verarbeiten / ABAP-Datenbankzugriffe / Open SQL / Performance-Hinweise. Zusätzlich sind dort noch ausführliche Empfehlungen zu Sekundärindizes verlinkt

5. Für Release 7.31 lagen die 5 Regeln, in etwas ausführlicherer Form, noch an dieser Stelle: Performance-Hinweise 7.31

6. Siegfried Boes, „Performance-Optimierung von ABAP-Programmen“, dpunkt Verlag 2009, ISBN 3898646157

7. Hermann Gahm, „ABAP Performance Tuning“, SAP Press, 2009, ISBN 978-1-59229-555-5

8. Hermann Gahm, Thorsten Schneider, Eric Westenberger, Christiaan Swanepoel „ABAP-Entwicklung für SAP HANA“, 2015, ISBN 978-3-8362-3661-4

4 ROBUSTHEIT UND KORREKTHEIT

Unter Robustheit verstehen wir die Fähigkeit eines Programms, selbst unter ungünstigen Umständen nicht abzubrechen sondern trotzdem korrekte Ergebnisse zu liefern. Das bedeutet insbesondere, dass Programme Fehlersituationen erkennen und in einer Weise darauf reagieren, dass die gewünschte Funktionalität nicht gefährdet wird.

Andererseits können Fehlersituationen manchmal nicht sinnvoll behandelt werden, und in vielen Fällen ist die Implementierung einer echten Fehlerbehandlung mit mindestens einer Ausnahme oder Fehlermeldung unangemessen aufwändig, da die Eintrittswahrscheinlichkeit als sehr gering eingeschätzt wird. In diesen Fällen ist es wichtig, ein zügiges Auffinden des Fehlers in der Wartungsphase zu ermöglichen, anstatt die Fehlerursache durch eine ungeschickte Fehlerbehandlungsroutine zu verschleiern.

In diesem Kapitel gehen wir auf Best Practices ein, die die Robustheit eigenentwickelter Programme fördern.

Viele Aspekte können durch den Code Inspector/ATC automatisch getestet werden, siehe dazu auch Abschnitt 7.2.2 Automatische Prüfungen.

4.1 FEHLERBEHANDLUNG

Die ABAP-Laufzeitumgebung bietet verschiedene Möglichkeiten, eine Anwendung auf Fehlersituationen hinzuweisen. Diese Möglichkeiten sind hier aufgeführt und die Best Practices jeweils erläutert.

4.1.1 SY(ST)-SUBRC-PRÜFUNGEN

Bei einer ganzen Reihe von ABAP-Befehlen setzt der SAP-Kernel nach deren Ausführung die globale Variable SY(ST)-SUBRC. In der Regel wird eine erfolgreiche Ausführung durch den Wert 0 angezeigt.

Der sy-subrc sollte nach allen Befehlen, die ihn setzen, geprüft werden. Alternativ zu einer dedizierten Auswertung von sy-subrc kann die Anweisung ASSERT sy-subrc = 0 verwendet werden, wenn der Programmierer der Ansicht ist, dass die Fehlersituation nicht auftreten oder die Fehlersituation in der Anwendung nicht sinnvoll behandelt werden kann. Die Verwendung dieser Anweisung führt im Fehlerfall zu einem Programmabbruch mit dem Laufzeitfehler ASSERTION_FAILED.

Der kurzfristige Mehraufwand macht sich bezahlt, weil Fehlersituationen sofort bemerkt und exakt lokalisiert werden können. Dies ist einem „rätselhaften" Systemverhalten vorzuziehen, welches vielleicht lange Zeit nicht bemerkt wird und bei dem die Ursache („root cause") oft nur mit viel Detektivarbeit zu ermitteln ist. Die Behandlung des sy-subrc kann durch den Code Inspector/ATC automatisch geprüft werden.

Für Programme z.B. mit Massenverarbeitung, deren Fortsetzung auch bei Fehlern für einzelne Objekte sichergestellt werden muss, ist eine echte Fehlerbehandlung notwendig (Fehler protokollieren, diesen Schleifendurchlauf abbrechen, usw.).

BEST PRACTICE
Setzen Sie bei Funktionsbausteinaufrufen explizit den speziellen Wert OTHERS, da sich die Liste der Ausnahmen ändern könnte, nachdem das rufende Programm fertiggestellt wurde.

4.1.2 MESSAGE-ANWEISUNG

Mit der MESSAGE-Anweisung können Status- oder Fehlermeldungen ausgegeben werden. Dabei unterscheidet sich das Verhalten der MESSAGE-Anweisung je nach Meldungsart (Status, Information, Fehler, Abbruch) und Betriebsmodus (Dialog oder Hintergrund).

BEST PRACTICE
Vermeiden Sie MESSAGE-Anweisungen außerhalb von Modulen, die direkt mit dem Anwender kommunizieren bzw. die in einer definierten Dialogschicht liegen.

MESSAGE-Anweisungen können in bestimmten Konstellationen von Meldungsart und Betriebsmodus aufgrund eines Bildschirmwechsels in klassischen Dynpros implizite COMMITs auslösen und führen innerhalb eines RFC-Aufrufs zu Verbindungsabbrüchen. Verwenden Sie innerhalb des eigentlichen Anwendungskerns klassenbasierte Ausnahmen, um auf Fehlersituationen hinzuweisen. Eine Ausnahme von dieser Regel ist die Verwendung der Anweisung MESSAGE * INTO zum Setzen der SY(ST)-Felder.

4.1.3 KLASSENBASIERTE AUSNAHMEN

Klassenbasierte Ausnahmen sind ein objektorientierter Mechanismus zur Fehlerbehandlung. Ein Programm erkennt einen Fehler und löst eine Ausnahme auf. Fängt der Aufrufer diese Ausnahme nicht, wird sie so lange im Call Stack nach oben propagiert, bis sie an anderer Stelle gefangen wird. Wird die Ausnahme an keiner Stelle im Call Stack behandelt, führt dies zu einem Dump. Ausnahmen können auch in prozeduralem Quellcode verwendet werden.

Die klassenbasierte Ausnahmebehandlung hat im Vergleich zur klassischen Fehlerbehandlung zwei wesentliche Vorteile:

1. Der Quellcode zur Fehlerbehandlung kann an einigen wenigen Stellen gesammelt werden, anstatt nach jedem Methodenaufruf u.Ä. wiederholt werden zu müssen.

2. Ausnahmeobjekte können nicht nur Fehlermeldungen beinhalten, sondern weitere Attribute wie interne Tabellen etc. Dem Benutzer können so zur Fehleranalyse mehr Details angezeigt werden.

Es ist wichtig, alle Ausnahmen zu fangen, auf die an der jeweiligen Stelle angemessen reagiert werden kann. Ausnahmen, die lokal nicht angemessen verarbeitet werden können, müssen in der Prozedur (Methode, Funktionsbaustein, Unterprogramm) deklariert werden, damit für den Aufrufer klar ersichtlich ist, dass solche Ausnahmen auftreten können[41]. Wichtig ist die einheitliche Verwendung: Für identische Fehlersituationen sollten einheitliche Ausnahmeklassen verwendet werden.

Zum Fangen von Ausnahmen dienen die ABAP-Befehle TRY...CATCH...ENDTRY. Entsprechend müssen alle Ausnahmen aller gerufenen Prozeduren, die klassenbasierte Ausnahmen erzeugen können, an irgendeiner Stelle des Call Stacks von einem TRY...CATCH behandelt werden, um einen robusten Ablauf zu gewährleisten.

41 *Eine detaillierte Unterscheidung der Ausnahmekategorien und Benutzungsempfehlungen dazu sind in der ABAP-Schlüsselwortdokumentation zu finden, Pfad: ABAP – Programmierrichtlinien → Architektur → Fehlerbehandlung →Ausnahmekategorien*

BEST PRACTICE:

- Erlauben Sie keine leeren Ausnahmebehandlungen, da es dadurch aufrufenden Stellen unmöglich gemacht wird, angemessen auf Ausnahmesituationen zu reagieren. In diesen Fällen ist das Propagieren der Ausnahme die richtige Reaktion. Eine Ausnahme von dieser Regel kann z.B. das Abfangen von Ausnahmen in einer Schleife sein. Dann sollte aber in einem Kommentar erläutert werden, warum weder Behandeln noch Propagieren der Ausnahme notwendig ist.
- Definieren Sie für Neuentwicklungen eine Ausnahmen-Oberklasse, die von CX_STATIC_CHECK erbt. Definieren Sie pro Anwendungskomponente eine oder mehrere Unterklassen, die sich auf Nachrichtenklassen mit Fehlermeldungen beziehen (automatische Implementierung des Interfaces IF_T100_MESSAGE). Deklarieren Sie in allen neuen Methoden und Funktionsbausteinen die Ausnahmen-Oberklasse als mögliche Ausnahme. Dadurch entsteht zu einem späteren Zeitpunkt z.B. beim Hinzufügen von Prüfungen kein Mehraufwand.

Bei Erweiterungen an Nicht-Kundencode kann es unter Umständen unmöglich sein, Ausnahmedeklarationen modifikationsfrei zu Methoden oder Funktionsbausteinen hinzuzufügen. In diesem Fall bietet sich die Verwendung von Ausnahmeklassen an, die von CX_NO_CHECK erben. Dies ermöglicht die Fehlerbehandlung ohne explizite Deklaration der Ausnahmeklassen in Methodensignaturen und ohne Programmabbruch zur Laufzeit.

Unterklassen von CX_DYNAMIC_CHECK sollten lediglich in Signaturen von Methoden deklariert werden, bei denen der Aufrufer durch Übergabe geeigneter Parameter ein Auftreten der Ausnahme verhindern kann. Dadurch muss an der Aufrufstelle kein unnötiger TRY-CATCH-Block ergänzt bzw. die Signatur nicht um die Ausnahmeklasse erweitert werden.

BEST PRACTICE

Fangen Sie in CATCH-Blöcken grundsätzlich Objekte von Ausnahmen-Oberklassen ab, da Fehlerbehandlungsroutinen in der Regel unabhängig vom konkreten Ursprung der Ausnahme arbeiten. Nur in Ausnahmefällen kann es sinnvoll sein, in vorangestellten CATCH-Blöcken gezielt Objekte von Ausnahmen-Unterklassen abzufangen, um unterschiedliche Fehlerbehandlungsroutinen durchzuführen.

Für die Verkettung von Ausnahmen gibt es beim Erzeugen einer Ausnahme das Attribut PREVIOUS.

Zum Werfen von mehreren Ausnahmen bietet sich folgendes Quellcodemuster an:

Quellcode

```
DATA: lx_validaton_exception type zcx_validation_exception.
IF ….
  “ Prüfung 1 nicht erfolgreich
  lx_validation_exception = new ...
ENDIF.

IF ….
  “ Prüfung 2 nicht erfolgreich
  lx_validation_exception = new … EXPORTING
    PREVIOUS = lx_validation_exception. “ Wichtig: Verkettung
ENDIF.

… “ weitere Prüfungen

IF lx_validation_exception IS BOUND.
  RAISE EXCEPTION lx_validation_exception.
ENDIF.
```

BEST PRACTICE

Schreiben Sie Fehlerbehandlungsroutinen in CATCH-Blöcken grundsätzlich so, dass alle verketteten Ausnahmeobjekte verarbeitet werden, es sei denn, vorhergehende Fehlermeldungen sind für die Fehlerbehandlung nicht relevant.

4.1.4 NICHT BEHANDELBARE AUSNAHMEN

Es gibt Ausnahmen, die vom ABAP-Quellcode aus nicht behandelt werden können und dadurch zwangsläufig zu einem Abbruch der Anwendung und einem Dump führen. In einigen Fällen ist es möglich, proaktiv vor der Ausführung eines Statements, das eine nicht behandelbare Ausnahme auslöst, die Rahmenbedingungen zu prüfen.

Beispiel: Das Statement OPEN DATASET löst einen Dump aus, wenn der Benutzer keine ausreichenden Berechtigungen zum Öffnen einer Datei hat. Um dies zu vermeiden, sollte die Berechtigung proaktiv mit dem Funktionsbaustein AUTHORITY_CHECK_DATASET geprüft werden.

4.2 KORREKTE IMPLEMENTIERUNG VON DATENBANK-ÄNDERUNGEN

4.2.1 SPERROBJEKTE

Um Dateninkonsistenzen zu vermeiden, müssen in Anwendungen mit pessimistischem Sperrkonzept[42] die entsprechenden Objekte vor Veränderung auf der Datenbank gesperrt werden. Zusammenhängende Objekte dürfen erst verändert werden, wenn alle zugehörigen Entitäten erfolgreich gesperrt sind.

Die SAP-Sperre besteht über mehrere Datenbank-LUWs (Logical Unit of Work) hinweg, sodass für das gesamte zu ändernde Business-Objekt konsistent Änderungen auf der Datenbank geschrieben werden können.

Die Sperren sind so genau wie möglich abzusetzen, um nur die relevanten Objekte innerhalb einer LUW zu sperren. Zudem sind die Sperren nur so lange wie nötig und so kurz wie möglich zu halten.

Für die Änderung von SAP-Standard-Datenobjekten direkt auf der Datenbank sind die entsprechenden SAP-Standard-Sperrfunktionen zu verwenden, da diese auch von den Standard-Transaktionen verwendet werden und so ein konsistentes Verhalten sicherstellen.

Für kundeneigene Entwicklungen sind entsprechende Sperrobjekte mit den zugehörigen Sperrfunktionen zu implementieren.

Die Sperren sind nach Abschluss des Updates auf das Objekt aufzuheben. Dies erfolgt mittels Aufruf des Dequeue-Funktionsbausteins. Beachten Sie dabei den wichtigen Scope-Parameter, wenn in diesem Zusammenhang auch Verbuchungsbausteine verwendet werden.

Um sicherzustellen, dass gesperrte Objekte auch in Ausnahmesituationen entsperrt werden, bieten sich in der klassenbasierten Ausnahmebehandlung CLEANUP-Blöcke an.

Neben dem pessimistischen Sperrkonzept mit Exklusivsperren kann auch das optimistische Sperrkonzept basierend auf dem Vergleich von Zeitstempeln zum Änderungszeitpunkt verwendet werden. Welches Sperrkonzept bei Neuentwicklungen gewählt wird, ist von Fall zu Fall zu entscheiden.

WEITERE QUELLEN

1. ABAP-Schlüsselwortdokumentation Datenkonsistenz

42 *zu den Begriffen pessimistisches/optimistisches Sperren siehe zum Beispiel Introduction to Database Concurrency Control*

4.2.2 FRAMEWORKS FÜR DEN DATENBANKZUGRIFF

BEST PRACTICE
Evaluieren Sie vor der manuellen Entwicklung einer Datenbankzugriffsschicht existierende, von SAP bereitgestellte Frameworks.

Zur Implementierung von Datenbankzugriffen bietet SAP u.a. das Business Object Processing Framework (BOPF)[43] an. Eine Einarbeitung in dieses oder andere Frameworks kann den Aufwand mittelfristig deutlich reduzieren, auch weil Frameworks in der Regel für weitere Infrastrukturthemen wie externe Schnittstellen Werkzeuge zur Verfügung stellen.

4.2.3 VERBUCHUNGSKONZEPT

Bei Verwendung klassischer Dynpro-Oberflächen oder Remoteaufrufen von Funktionsbausteinen kann der Fall auftreten, dass auf einem ABAP-Anwendungsserver Programmlogik von mehreren Workprozessen hintereinander ausgeführt wird. Jeder Wechsel zwischen Workprozessen ist mit einem impliziten Datenbank-Commit verknüpft. Falls kein Framework zum Datenbankzugriff verwendet wird, kann es deshalb notwendig sein, in eigenentwickelten Programmen Maßnahmen zur Bündelung von Datenbankänderungen einzusetzen. Für mehr Informationen zu diesem Thema wird auf den Abschnitt Datenkonsistenz der ABAP-Schlüsselwortdokumentation[44] verwiesen.

4.3 PROTOKOLLIERUNG

Generell sollten Fehler, Ausnahmen und Meldungen ins Business Application Log gespeichert werden, um diese an zentraler Stelle (Transaktion SLG1) prüfen zu können.

Der Zugriff erfolgt über die Funktionsgruppe SBAL, die im System sehr gut dokumentiert ist.

Die Verwendung des Business Application Logs hat folgende Vorteile:

1. Zentrale Ablage: Im Business Application Log können Meldungen an zentraler Stelle verwaltet werden; dies erleichtert den Überblick und die Administration der Anwendungen.

2. Wiederverwendbarkeit: Das Business Application Log bzw. die zugehörigen Bausteine/Klassen bieten einen guten Funktionsumfang für das Thema Logging – dieser muss nicht „neu erfunden" werden. Zu diesen Möglichkeiten zählen u.a.:

   a. Persistenz: Speicherung von Meldungen im Log
   b. Hierarchiedarstellung von Meldungen: Zusammenfassung zu Problemklassen
   c. Integration zusätzlicher Felder in die Protokollierungsobjekte
   d. Interaktivität: Nachlesen von Informationen bei Aufruf einer Meldung und Anreichern von Informationen
   e. Integration der Protokollanzeige in eigene Anwendungen/UIs: Sub-Screen/Control/Pop-up

BEST PRACTICE
Es ist sinnvoll, die SBAL-Funktionen einmalig mit einer Klasse zu verschalen, damit eine einfache und knappe Verwendung in der Art

```
MESSAGE i002(zmessage) WITH lv_value_1 INTO
DATA(lv_dummy).
  lo_logger->add_msg_from_sy( ).
```

möglich ist, und gleichzeitig der Verwendungsnachweis für Nachrichten weiterhin funktioniert.

43 http://scn.sap.com/docs/DOC-45425
44 ABAP Schlüsselwortdokumentation Datenkonsistenz

Der etwas aufwändigere Aufruf der SBAL-Funktionen ist dann in der Klasse verborgen.

BEST PRACTICE

Geben Sie beim Anlegen von Business Application Logs ein Verfalldatum mit (BAL_S_LOG-ALDATE_DEL beim Aufruf von BAL_LOG_CREATE) und planen Sie eine regelmäßige Bereinigung mit der Transaktion SLG2 ein.

Eine eigene Protokolltabelle (DB-Tabelle) kann ein sinnvoller Ersatz bzw. eine sinnvolle Ergänzung zum Business Application Log sein, wenn man laufübergreifend Werte vergleichen möchte. Dies kann z.B. für tägliche Batchläufe in Frage kommen, die dann „n Datensätze von Typ x verarbeitet“ protokollieren.

Eine weitere Ergänzung sind die aktivierbaren Log-Points (Befehl LOG-POINT in Kombination mit Transaktion SAAB, siehe Referenz 3). Diese müssen jedoch zuvor aktiviert werden und es werden nur jeweils die Anzahl der Vorkommen sowie die Feldinformationen für das letzte Auftreten aufgezeichnet.

WEITERE QUELLEN

1. SAP Application Log – Leitfaden für Anwender
2. Beispiele zur Verwendung der BAL-Funktionen in: Thorsten Franz, Tobias Trapp: ABAP Objects: Application Development from Scratch. SAP Press, 2008. ISBN 9781592292110
3. ABAP-Schlüsselwortdokumentation zu Checkpoints und Checkpoint-Gruppen

5 ABAP-SICHERHEIT UND COMPLIANCE

Dieses Kapitel beschreibt, in welcher Weise sich Programmierfehler im ABAP negativ auf die Sicherheit eines Unternehmens auswirken können und welche Gegenmaßnahmen es gibt. Das Thema Applikationssicherheit betrifft prinzipiell alle Programmiersprachen, wurde aber bislang bei ABAP zumeist nicht hinreichend beachtet. Mögliche Risiken, die Unternehmen durch fehlerhafte ABAP-Programme entstehen können, sind in diesem Zusammenhang:

- Rechtsverstöße (z.B. könnte das Bundesdatenschutzgesetz verletzt werden, wenn die Programmierfehler zum Diebstahl von Personaldaten führen)
- Verletzung von Compliance-Anforderungen (z.B. SOX oder PCI/DSS)
- Cyber-Angriffe mit dem Ziel der Industriespionage, Sabotage oder Erpressung
- Hintertüren, die Innentätern erweiterten Zugriff auf Daten ermöglichen
- Pressemeldungen, wenn Schwachstellen oder Datendiebstähle bekannt werden

Insbesondere darf selbst geschriebener ABAP-Code die Vertraulichkeit, Integrität und Verfügbarkeit der Geschäftsdaten nicht gefährden.

Da dieses Thema sehr komplex ist, können wir im Rahmen dieses Dokuments nur ein paar ausgewählte, zentrale Themen behandeln. Für weitergehende Fragen verweisen wir auf die Literaturhinweise am Ende dieses Kapitels.

5.1 PRÜFUNGSRELEVANTE SICHERHEITSTHEMEN IM SAP STANDARD

Wir orientieren uns bei unseren Empfehlungen primär an einem Dokument, das beim Bundesamt für Sicherheit in der Informationstechnik (BSI) als Hilfsmittel veröffentlicht wurde, den „Top 20 Sicherheitsrisiken in ABAP-Anwendungen“[45].

Dieses Dokument basiert auf einer umfassenden statistischen Analyse von Sicherheitsfehlern in ABAP-Eigenentwicklungen (Details hierzu siehe beim BSI veröffentlichtes Dokument). Auf Basis dieser Statistik konnten die häufigsten Sicherheitsdefekte identifiziert werden. Diese wurden ergänzt durch einen Satz weniger häufiger aber dafür besonders kritischer Sicherheitsrisiken. In Kombination ist dies eine wertvolle

45 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Hilfsmittel/Extern/TOP-20_Sicherheitsrisiken-in-ABAP-Anwendungen.pdf

Orientierungshilfe für alle Unternehmen, die in diesen Bereich einsteigen und zunächst nur eine begrenzte Menge von Regeln vorgeben wollen.

Im Folgenden werden die Risiken zunächst in Bereiche unterteilt und dann Empfehlungen ausgesprochen.

5.1.1 BERECHTIGUNGSPRÜFUNG

Rollen und Berechtigungen sind ein zentrales Sicherheitsthema im SAP-Umfeld. Es ist daher wichtig zu verstehen, dass ABAP ein so genanntes "explizites Berechtigungsmodell" verwendet[46]. Berechtigungsprüfungen sollten grundsätzlich explizit im Quellcode programmiert werden, damit die Prüfung für alle Fälle sichergestellt ist.

Sicherheitsprobleme ergeben sich häufig in folgenden Fällen:

- Der Entwickler vergisst, die Berechtigungsprüfung zu programmieren.
- Der Entwickler prüft beim Aufruf einer Transaktion nicht die Transaktions-Startberechtigung.
- Der Entwickler prüft keine fachlichen Berechtigungen in RFC-fähigen Funktionsbausteinen.
- Der Entwickler sichert kritische PAI-Ereignisse nicht ab, die durch direkte Eingabe von Funktionscodes ausgelöst werden können, obwohl das entsprechende UI-Element beim PBO abgeschaltet wurde.
- Der Entwickler verwendet das falsche Berechtigungsobjekt.
- Der Entwickler verwendet eine proprietäre Berechtigungsprüfung.
- Der Entwickler behandelt den Rückgabewert der Prüfung nicht.

5.1.2 TESTBARKEIT

Immer mehr Unternehmen lassen ihre Eigenentwicklungen nicht nur durch interne Teams testen, sondern auch durch unabhängige Dritte. Dabei wird entweder getestet, wie sich ein Programm zur Laufzeit verhält (sogenannte dynamische Tests) oder der Test basiert auf einer Analyse des Quellcodes (sogenannte statische Tests). Damit ein Programm testbar ist, muss sichergestellt sein, dass der ABAP-Code sowohl vollständig ausführbar als auch vollständig einsehbar ist.

Sicherheitsprobleme ergeben sich in folgenden Fällen:

- Der Entwickler schreibt Code, der sich auf verschiedenen Systemen unterschiedlich verhält, was einen dynamischen Test potentiell gefährlicher Funktionalität auf einem Qualitätssicherungssystem beeinträchtigen kann.
- Der Entwickler schreibt Code, der sich auf verschiedenen Mandanten unterschiedlich verhält, was ebenfalls einen dynamischen Test auf einem Qualitätssicherungssystem beeinträchtigen kann.
- Der Entwickler versteckt seinen Code und verhindert damit eine Quellcode-Analyse.

Während versteckter Code selten anzutreffen ist, kommen hart-codierte Prüfungen auf Mandant oder System-ID mit einer Wahrscheinlichkeit von über 70 % pro System vor.

5.1.3 DATENSCHUTZ

Ein primärer Zweck von SAP-Systemen ist die Verwaltung von Geschäftsdaten. Darin enthalten sind unter anderem Geschäftsgeheimnisse, Personaldaten und Finanzdaten. Der Schutz dieser Daten ist für Unternehmen elementar wichtig, nicht zuletzt auf Grund gesetzlicher Anforderungen. Entsprechend muss auch selbst geschriebener Code größte Sorge tragen, dass diese Daten nicht unerlaubt aus dem System abgezogen werden können.

Sicherheitsprobleme ergeben sich in folgenden Fällen:

- Eigenentwickelte Programme zeigen sensible Daten ohne hinreichende Berechtigungsprüfung an, z.B. im SAP GUI oder in HTML-Seiten.
- Eigenentwickelte Programme exportieren sensible Daten ohne hinreichende Berechtigungsprüfung.
- Eigenentwickelte Programme übertragen sensible Daten ohne hinreichende Berechtigungsprüfung.

46 *Man kann zwar über S_TCODE viele Fälle implizit berechtigen, aber es verbleiben etliche Lücken (Externe Zugriffe über RFC oder Services, Verzweigung in andere Programme, unterschiedliche Modi für Anzeige und Änderung innerhalb eines Programms)*

5.1.4 INJECTION-SCHWACHSTELLEN

Im ABAP-Quellcode werden häufig dynamische Befehle, Strukturen und Bezeichner verwendet, die aus Benutzereingaben zusammengefügt werden. Wenn hier nicht durch Input-Validierung bzw. Output-Encoding verhindert wird, dass Steuerzeichen in den Eingaben die Semantik des dynamischen Befehls verändern können, dann kann der dynamische Befehl zur Laufzeit schadhaft manipuliert werden.

Sicherheitsprobleme ergeben sich in folgenden Fällen:

- Der Entwickler vermischt Input mit dynamischem ABAP oder gefährlichen Befehlen, wie z.B. die Ausführung von Betriebssystemkommandos oder von nativem SQL.
- Der Entwickler führt eine unzureichende Input-Validierung durch.
- Der Entwickler vergisst Output-Encoding, um schadhafte Effekte durch Steuerzeichen im Input zu vermeiden.

Manche Injection-Schwachstellen treten häufig im eigenen Code auf. So befinden sich z.B. statistisch betrachtet auf jedem Kunden-System 294 Directory Traversal Schwachstellen, welche jedoch nur begrenzten Schaden verursachen. Es gibt jedoch auch Injection-Schwachstellen, deren Ausnutzung einem Angreifer vollständigen Zugriff aus das gesamte SAP-System ermöglicht. Von diesen besonders schwerwiegenden Schwachstellen gibt es statistisch gesehen 16 pro SAP-System.

5.1.5 STANDARD-SCHUTZ

Der SAP-Standard liefert verschiedene Sicherheitsmechanismen, die SAP-Systeme vor Angriffen schützen. Zu diesen Sicherheitsmechanismen gehören unter anderem die Mandantentrennung, Logging und das Berechtigungswesen. Entsprechend verlassen sich Unternehmen auf diese Mechanismen, wenn sie ihre Systeme sicher konfiguriert haben.

Eigenentwicklungen dürfen daher die Funktionalität dieser Mechanismen nicht unterlaufen.

Sicherheitsprobleme ergeben sich beispielsweise in folgenden Fällen:

- Der Entwickler umgeht die Mandantentrennung.
- Der Entwickler deaktiviert das Datenbank-Logging.
- Eigenentwickelte Programme ändern Geschäftsdaten, ohne Änderungsbelege anzulegen.

Fehler in diesem Bereich sind insbesondere daher kritisch, weil sie ggf. umfangreiche Sicherheitsmaßnahmen des Unternehmens zunichtemachen können.

5.2 SICHERHEITSEMPFEHLUNGEN

Um Software schreiben zu können, die wirklich sicher ist, bedarf es jahrelanger Erfahrung. Und selbst dann übersehen auch Experten gelegentlich etwas. Wir beschränken uns daher in diesem Leitfaden auf eine ausgewählte Liste von Empfehlungen, die in vielen Fällen das Risiko eines erfolgreichen Angriffs zumindest deutlich senken können.

5.2.1 SIEBEN ALLGEMEINE REGELN FÜR DIE SICHERE ABAP-PROGRAMMIERUNG

BEST PRACTICE

Wir empfehlen bei der ABAP-Programmierung auf folgende Regeln zu achten:

- Schränken Sie die Ausführung jeglicher Geschäftslogik mittels Berechtigungsprüfungen ein, damit durch die Berechtigungsadministration der Benutzerkreis maximal reduziert werden kann.
  - Selbst wenn sich in Ihren Programmen Sicherheitsfehler befinden sollten, können diese dann zumindest nur von wenigen Angreifern ausgenutzt werden.
- Verlassen Sie sich bei relevanten Prüfungen nie darauf, dass diese bereits an anderer Stelle erfolgt sind.
  - Wenn für die Sicherheit Ihrer Programmlogik bestimmte Prüfungen erforderlich sind, dann führen Sie diese durch.
- Vermeiden Sie generische Programmierung.
  - Je vielseitiger eine bestimmte Programmlogik ist, desto höher ist die Wahrscheinlichkeit, dass ein Angreifer einen Weg findet, sie zu missbrauchen.
- Umgehen Sie keine Sicherheitsmechanismen des SAP-Standards.
  - Wenn SAP für eine bestimmte sicherheitsrelevante Funktionalität eine Methodik anbietet oder vorschreibt, verwenden Sie diese und schreiben Sie keine „einfachere“ Alternative.
- Prüfen Sie die Gültigkeit der Werte sämtlicher Eingabeparameter.
  - Insbesondere Injection-Angriffe werden erschwert, wenn der Wertebereich von Eingaben weitgehend eingeschränkt ist. Wenn Sie beispielsweise eine Telefonnummer verarbeiten, sollten nur Ziffern und wenige Sonderzeichen wie ( ) + - erlaubt sein.
- Verwenden Sie bei Eingabeparametern möglichst kurze Variablentypen.
  - Selbst wenn Ihre Gültigkeitsprüfung unzureichend sein sollte, werden Angriffe deutlich erschwert, wenn dem Angreifer nur wenige Zeichen zur Verfügung stehen.
- Verwenden Sie möglichst nur Funktionen aus zentralen Bibliotheken für Ihre Sicherheitsprüfungen. Schreiben Sie niemals eigene.
  - Funktionen, die Sicherheitsprüfungen ausführen, sind komplex in der Erstellung und daher fehleranfällig. Die Erstellung und Wartung solcher Funktionen sollte von einem zentralen Team im Unternehmen durchgeführt werden. Dadurch können potentielle Fehler vermieden bzw. im Nachgang effizient verbessert werden.

5.2.2 DIE KRITISCHSTEN UND HÄUFIGSTEN RISIKEN IN ABAP

BEST PRACTICE

Konzentrieren Sie sich bei der Programmierung auf die Vermeidung der folgenden besonders häufigen und kritischen Sicherheitsrisiken. Damit stellen Sie in Ihren Eigenentwicklungen eine solide Grundabsicherung sicher.

Die kritischsten ABAP-Risiken und wichtigsten Empfehlungen entnehmen Sie bitte dem BSI Hilfsmittel „Top 20 Sicherheitsrisiken in ABAP-Anwendungen".

Eine etwas kompaktere Liste kritischer Risiken liefert der BIZEC APP/11-Standard. Hier sind allerdings die Beschreibungen nicht so ausführlich wie beim BSI-Hilfsmittel.

Die folgende Liste enthält eine Übersicht über SAP-Meldungen, die Gegenmaßnahmen zu einigen der oben beschriebenen Probleme enthalten.

SAP Note	Schwachstelle
1520356	SQL Injection
887168, 944279, 822881	Cross-Site Scripting
1497003	Directory Traversal

Tabelle 1: SAP-Hinweise, die Gegenmaßnahmen beschreiben

Selbstverständlich empfiehlt es sich, SAP-Sicherheitshinweise zeitnah zu prüfen und einzuspielen. Diese lösen allerdings nur Sicherheitsdefekte im SAP-Standard-Quellcode. Eine regelmäßige Prüfung der Eigenentwicklungen ist daher zwingend erforderlich.

5.3 COMPLIANCE-PROBLEME DURCH ABAP

Das Thema Applikationssicherheit wurde in der Vergangenheit selten mit Compliance in Verbindung gebracht, ist aber für Wirtschaftsprüfungen durchaus relevant[47].

Die meisten Unternehmen verfügen über ein Internes Kontrollsystem (IKS), das Compliance-Risiken entgegenwirkt. Dies ist beispielsweise in den international anerkannten Referenzmodellen COSO & COBIT beschrieben. In einer typischen IKS-Struktur sind die „Generellen IT-Kontrollen" (ITGC-IT General Controls) Voraussetzung für das Erreichen aller IKS-Ziele im IT-dominierten Umfeld.

Ein elementarer Baustein der ITGC ist das Änderungswesen (Change Management), zu dem wiederum Eigenentwicklungen zählen. Sicherheitsdefekte im selbstgeschriebenen ABAP-Quellcode stellen also eine Verletzung der generellen IT-Kontrollen dar. Wenn also in einem IKS der Bereich Change Management bzw. selbstgeschriebener ABAP-Code unzureichend abgedeckt ist, dann helfen auch alle anderen Maßnahmen nicht, da diese auf ein funktionierendes Change Management aufbauen. Dies führt zu einem Compliance-Verstoß.

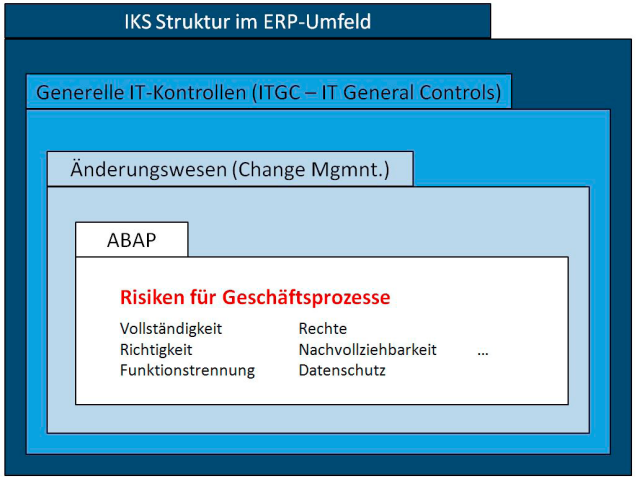


Abbildung 1: IKS-Risiken durch unsicheren ABAP-Quellcode

47 *Weitere Informationen: Maxim Chuprunov , Handbuch SAP-Revision, SAP Press 2011*

Dies bedeutet insbesondere, dass Sicherheitsdefekte im ABAP-Quellcode potenziell nicht nur Auswirkungen auf Compliance-Standards haben, sondern auch gesetzliche Anforderungen verletzen können.

Alle im BSI-Hilfsmittel „Top 20 Sicherheitsrisiken in ABAP-Anwendungen" dargestellten Sicherheitsrisiken sind daher auch Compliance-relevant.

5.4 TESTWERKZEUGE

Für ABAP-Sicherheitstests eignen sich insbesondere sogenannte statische Codeanalyse-Tools. SAP erweitert mit dem lizenzpflichtigen „SAP NetWeaver Application Server, Add-on for code vulnerability analysis" den Code Inspector/ATC um komplexere Sicherheitsprüfungen.

Daneben gibt es einige andere kommerzielle Anbieter. Interessant sind folgende Eigenschaften eines solchen Tools:

- Analyse auch des SAP-Standard-Quellcodes und von Third-Party-Code, entweder komplett oder Beschränkung auf die von Kundencode aufgerufenen Funktionen.
- Continuous Monitoring, ermöglicht durch sehr hohe Scan-Geschwindigkeiten.
- Globale Daten- und Kontrollflussanalysen, da diese für die meisten Sicherheitstests elementar sind.
- Umfangreiche Beschreibungen des jeweiligen Problems mit Lösungsvorschlägen.
- Hinreichende Testabdeckung:
  - „APP/11"-Standard von BIZEC
  - „Top 20 Sicherheitsrisiken in ABAP-Anwendungen", veröffentlicht beim BSI

Eine gute Integration externer Tools in die Entwicklungsumgebung (SE80, Eclipse) und den Entwicklungsprozess (ChaRM, TMS) fördert deren Akzeptanz und Verwendung.

WEITERE QUELLEN

1. Andreas Wiegenstein, Markus Schumacher, Sebastian Schinzel, Frederik Weidemann, Sichere ABAP-Programmierung, SAP Press 2009
2. Maxim Chuprunov, Handbuch SAP-Revision, SAP Press 2011
3. BIZEC – The Business Application Security Initiative
4. BSI-Hilfsmittel – Top 20 Sicherheitsrisiken in ABAP-Anwendungen vom 16.10.2014
5. The Business Code Quality Benchmark 2016
6. PCI / DSS (Payment Card Industry Data Security Standard)

6 DOKUMENTATION

Die Dokumentation von Software ist in vielen Fällen genauso wichtig wie die Entwicklung selbst. Fehlt die Dokumentation oder ist sie nicht in ausreichendem Maß vorhanden, führt dies spätestens bei der Weiterentwicklung oder dem Wechsel von Entwicklern zu erhöhten Aufwänden. In diesem Kapitel werden unterschiedliche Möglichkeiten zur Dokumentation von Entwicklungen in einem SAP-System dargestellt.

Allgemein ist es bei jeder Dokumentation wichtig, das richtige Maß zu finden und umzusetzen. In der Praxis findet man leider immer wieder die beiden Extreme: entweder eine sehr umfangreiche oder gar keine Dokumentation. Die umfangreiche Dokumentation ist allerdings inhaltlich oft mangelhaft, weil sie viele redundante/kopierte Informationen enthält und nicht aktuell gehalten wird. Sind die offiziellen Anforderungen an die Dokumentation zu hoch oder zu abstrakt, führt dies manchmal dazu, dass sie gar nicht erst erstellt wird.

Die Dokumentation sollte während der Entwicklung, unbedingt aber vor der Produktivsetzung bzw. Bereitstellung erstellt werden und es sollte keine Produktivsetzung ohne fertige Dokumentation geben. Ansonsten ergibt sich in der Regel ein Mehraufwand oder letztlich eine fehlende Dokumentation.

6.1 DOKUMENTATION UNABHÄNGIG VON ENTWICKLUNGSOBJEKTEN

Neben der Beschreibung der vielen Entwicklungsobjekte, die einzelne, sehr spezielle Funktionen im ABAP-System übernehmen, gibt es auch den Bedarf, die größeren Zusammenhänge innerhalb eines Moduls und modulübergreifend darzustellen. Dazu gehören z.B. Antworten auf Fragen wie:

- Welche Abhängigkeiten gibt es zwischen den Modulen?
- Welche Anwendungen werden bei welchen Geschäftsprozessen verwendet?
- Welche Hintergrundjobs laufen wann am Tag / im Monat / im Jahr und welche Entwicklungsobjekte sind davon betroffen?

Für die Beantwortung dieser Fragen findet sich unserer Meinung nach kein geeignetes Ablagemedium innerhalb der SAP-Entwicklungsumgebung, das insbesondere Grafiken gut integriert. Wir empfehlen daher, für die Dokumentation dieser übergreifenden Zusammenhänge auf andere Medien zurückzugreifen. Beispiele dafür sind:

- SAP Solution Manager
- Interne (Produkt-)Wikis
- Dokumente in gepflegten öffentlichen Verzeichnissen (Portalablage, SharePoint, Fileshare ...)

Die Erfahrung zeigt, dass die Herausforderung in diesem Bereich primär in der Frage der Disziplin liegt. Diese Herausforderung kann kein Tool lösen, sondern nur das Entwicklungsteam und die zugehörige Entwicklungsleitung.

Zur Dokumentation der Systemarchitektur inklusive Entwurfsentscheidungen bietet sich die Verwendung einer Vorlage an, wie z.B. das arc42-Template. Dies kann verhindern, dass wesentliche Aspekte in der Dokumentation vergessen werden und beschleunigt – bei Verwendung einer Vorlage über mehrere Projekte hinweg – die Suche nach spezifischen Informationen. Zudem erleichtert die Festlegung von Vorlagen das Erstellen der Dokumentation parallel zur Entwicklung und die Einhaltung eines angemessenen Abstraktionsniveaus.

Dokumentenvorlagen wie das arc42-Template müssen nicht immer vollständig „ausgefüllt" werden, sondern relevante Teile sollen je nach Art und Umfang des Entwicklungsprojekts identifiziert und der Rest gelöscht werden.

Darüber hinaus kann eine veraltete Dokumentation irreführend sein. Deshalb sollte in allen Dokumenten der Stand und eine Versionierung enthalten sein, um die Aktualität bewerten zu können.

Innerhalb einer SAP-Systemlandschaft bietet der SAP Solution Manager Möglichkeiten zur Projektdokumentation. Die nachfolgenden Links bieten weitere Informationen dazu.

WEITERE QUELLEN

1. Dokumentation SAP Solution Manager 7.1
2. SCN-Blog „Business process documentation with SAP Solution Manager 7.1“
3. Das arc42-Template zur Architekturdokumentation
4. Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. Carl Hanser Verlag GmbH Co KG, 2015. ISBN 9783446429246

6.2 DOKUMENTATION VON ENTWICKLUNGSOBJEKTEN

Neben Methoden, Funktionsbausteinen und Reports, die Dokumentation im Quellcode enthalten können, existieren weitere Entwicklungsobjekte im ABAP-System, die keinen Quellcode besitzen und daher auf anderem Weg dokumentiert werden müssen. Beispiele dafür sind:

- DDIC-Objekte
- Transaktionen

BEST PRACTICE

Wir empfehlen, für alle Entwicklungsobjekte und unabhängig vom Quellcode die Dokumentationsfunktion der ABAP-Workbench zu nutzen und die Aufgaben und Bedeutungen dieser Objekte im SAP-System zu dokumentieren. Hierbei sollte ausschließlich der Ist-Stand dokumentiert werden, gegebenenfalls angereichert um kurze Verweise auf die Änderungsdokumentation (Transportdokumentation, Defekt-Nummern).

Da die Workbench-Dokumentation auch an das Transportwesen angeschlossen ist, steht sie in allen Einzelsystemen einer Systemlandschaft zur Verfügung. Weiterhin kann diese Dokumentation von allen Benutzern eingesehen werden und wird für Reports vom ABAP-System automatisch in die Benutzeroberfläche eingebunden. Ein weiterer Vorteil kann darin bestehen, dass die Dokumentation mehrsprachig geführt werden kann.

Auf SAP-Systemen mit SAP_BASIS >= 7.40 können im Quellcode ABAP-Doc-Kommentare verwendet werden. Dies kann als Alternative zur Dokumentation in der ABAP-Workbench verwendet werden. Der volle Funktionsumfang von ABAP-Doc-Kommentaren lässt sich derzeit allerdings nur mit den ABAP-Development-Tools für Eclipse ausschöpfen. Bei Verwendung von Core Data Services zur Definition von DDIC-Objekten können wesentlich mehr Entwicklungsobjekte im Quellcode dokumentiert werden und die Notwendigkeit externer Dokumentation entfällt.

Beginnend mit SAP NetWeaver 7.50 lassen sich die ABAP-Doc-Kommentare von Klassen und Schnittstellen als HTML-Dateien exportieren[48].

48 http://scn.sap.com/community/abap/eclipse/blog/2015/10/21/new-abap-doc-features-with-netweaver-75

6.3 DOKUMENTATION IM QUELLCODE

6.3.1 DOKUMENTATIONSSPRACHE

BEST PRACTICE
Als Kommentierungssprache sollte Englisch verwendet werden.

Entwicklungsteams arbeiten heutzutage überwiegend international zusammen. Auch wenn Sie derzeit rein deutschsprachig entwickeln, kann Ihr Projekt im Laufe der Zeit internationalisiert werden. Der Aufwand, der dann durch Koordinationsprobleme oder sogar nachträgliches Übersetzen entsteht, steht in keinem Verhältnis zu dem vielleicht größeren Aufwand durch englische Dokumentation.

Es hat sich außerdem gezeigt, dass die Lesbarkeit von Quellcode und Kommentaren durch englischsprachige Kommentare erhöht wird. Denn die ABAP-Befehle selbst sind englisch und im Stil von Sätzen aufgebaut. Der Leser des Quellcodes muss bei englischer Dokumentation also nicht ständig die Sprache wechseln.

6.3.2 DOKUMENTATION VON ÄNDERUNGEN

Ab dem Zeitpunkt der Produktivsetzung eines Programms sollte darauf geachtet werden, dass Änderungen in Programmen angemessen dokumentiert werden. Hier ist das richtige Maß wesentlich: Eine vollständige Versionshistorie aller Änderungen und auskommentierter Quellcode reduzieren die Lesbarkeit des Quellcodes. Trotz dieses Nachteils dokumentieren einige Entwicklungsteams bewusst alle Änderungen im Quellcode, um die Fehlersuche auf Produktiv- oder Testsystemen zu vereinfachen, in denen die Versionshistorie nicht zur Verfügung steht.

BEST PRACTICE
Nachträgliche Änderungen am Quellcode sollten – von Kopfkommentaren abgesehen – nur in Ausnahmefällen direkt im Quellcode dokumentiert werden.

6.3.3 PROGRAMMKOPF

Änderungen in Programmen können zum Beispiel mit Namen oder Namenskürzel des Entwicklers, dem Tagesdatum oder Monat und der Änderungsbelegnummer (Change Document, Incident Ticket, ...) sowie einer stichwortartigen Beschreibung der Änderung im Programmkopf dokumentiert werden.

Beispiel:

```
*/ Change Log
*/ VN/Date                  ChangeDoc Description
*/ MZ/2012-08-06 CD4712     Add MMSTA in Output
*/ MZ/2012-02-01 CD4711     Import Material number
*/ MM/2009-01-01 CD0815     Added Field ABC in method signature and source
                            code in order to support quick search
```

Durch diese Informationen kann später nachvollzogen werden, welche Erweiterung oder Fehlerbehebung Auslöser einer Änderung war. Diese Kopfkommentare helfen auch dabei zu erkennen, wie oft ein Programm geändert wurde, wer sich wahrscheinlich gut damit auskennt und wie lange die letzte Änderung zurückliegt. Diese Informationen sind hilfreich, schon bevor die nächsten Änderungen in einem Programm geplant oder eingebaut werden.

6.3.4 KOMMENTARE IM QUELLCODE

Kommentare im Quellcode sollen dazu dienen, Entwicklern das Verstehen des Quellcodes zu erleichtern, sofern dies nicht durch geschickte Gestaltung des Quellcodes allein (Modularisierung, Namenswahl von Methoden und Variablen) erreichbar ist.

Kommentare sind für andere Entwickler und mit zunehmendem zeitlichen Abstand auch für den ursprünglichen Entwickler gedacht. Sie sollten die Frage beantworten, „Warum" etwas programmiert wurde und nicht „Was". Letzteres ergibt sich aus dem Quellcode ohnehin, während die Beweggründe oft nicht klar erkennbar sind. Gerade sie helfen beim Verständnis aber wesentlich weiter. Dabei gilt der Grundsatz: So wenig Kommentar wie möglich, so viel Kommentar wie nötig.

Stern-Kommentare sollten nur im Programmkopf oder für das temporäre Auskommentieren von altem Quellcode verwendet werden.

Für alle anderen Kommentare empfiehlt SAP, Inline-Kommentare zu verwenden. Diese sollten jeweils vor dem Quellcode stehen, den sie dokumentieren, und genauso eingerückt sein wie dieser Quellcode. Letzteres wird (nur) für Inline-Kommentare auch vom Pretty Printer korrekt durchgeführt.

WEITERE QUELLEN

1. Horst Keller, Wolf Hagen Thümmel: ABAP-Programmierrichtlinien. SAP Press, 2009. ISBN 9783836212861

7 UMSETZBARKEIT UND DURCHSETZBARKEIT

Dieses Kapitel beschreibt, wie sich die Best Practices aus dieser Guideline in der Praxis verwirklichen lassen. Wir unterscheiden hierbei Umsetzbarkeit und Durchsetzbarkeit der Richtlinien.

Im Abschnitt „Umsetzbarkeit" erklären wir, worauf ein Unternehmen achten sollte, das Programmierrichtlinien einführen möchte. Wir erklären, wie ein Prozess dafür aussehen könnte, wie man diesen ins Leben ruft und vor allem, wie man ihn am Leben erhält. Im Abschnitt „Durchsetzbarkeit" legen wir dar, wie ein Unternehmen die Vorgaben aus dem Prozess prüfen kann. Hierzu gehören organisatorische Aspekte genauso wie Prüfmethoden und Werkzeuge. Wir gehen aber auch auf Grenzen der Durchsetzbarkeit ein.

Abschließend stellen wir Tipps aus der Praxis vor, die die Autoren bei verschiedenen Projekten im Umfeld der SAP-Entwicklung gesammelt haben.

7.1 UMSETZBARKEIT

Wer erfolgreich Programmierrichtlinien in einem Unternehmen einführen möchte, muss zunächst das Management dafür gewinnen. Denn die Verbesserung der Quellcodequalität bedeutet zunächst eine Investition in Prozesse und Werkzeuge sowie in die Fortbildung der involvierten Mitarbeiter. Insbesondere muss das Management überzeugt sein, dass das Unternehmen durch diese Prozesse langfristig Kosten spart.

7.1.1 MOTIVATION FÜR EINEN PROZESS

Nachfolgend finden Sie Anhaltspunkte, welche Qualitätsaspekte bei der Einführung eines Prozesses zur Qualitätssicherung adressiert werden und welche Vorteile dies für Unternehmen hat:

Sicherheit

- Vorteil: Das Unternehmen verhindert, dass Anwender unbefugt an kritische Daten gelangen bzw. unbefugt kritische Daten verändern können.
- Risiken bei Qualitätsmängeln: Sabotage, Industriespionage, unerwünschte Pressemeldungen hervorgerufen durch Datenlecks, Stillstand der Produktivsysteme.

Compliance

- Vorteil: Das Unternehmen kann jederzeit nachweisen, dass die entwickelte Software den Anforderungen relevanter Compliance-Standards und gesetzlicher Regelungen genügt.
- Risiken bei Qualitätsmängeln: Scheitern der Wirtschaftsprüfung, Verstoß gegen Compliance-Anforderungen oder gesetzliche Regelungen (z.B. Datenschutz).

Performance

- Vorteil: Das Unternehmen stellt sicher, dass die vorhandene Hardware optimal genutzt werden kann und schützt damit die bisherige Investition in Hardware. Außerdem steigt die Zufriedenheit der Mitarbeiter, da die Nutzung der Anwendung produktiver wird.
- Risiken bei Qualitätsmängeln: Geringere Akzeptanz von den Anwendern bzw. zusätzliche Kosten für schnellere Hardware, um die Softwaremängel auszugleichen.

Robustheit

- Vorteil: Das Unternehmen stellt den kontinuierlichen Betrieb der Geschäftsanwendungen sicher und vermeidet Unproduktivität auf Grund von Systemausfällen.
- Risiken bei Qualitätsmängeln: Geringere Akzeptanz von den Anwendern und die Betriebskosten steigen wegen Unproduktivität der Anwender sowie durch Fehleranalysen und Wartungsarbeiten durch Techniker.

Wartbarkeit

- Vorteil: Das Unternehmen erreicht, dass die Applikation nachhaltig und kosteneffizient gewartet werden kann, weil die Programmstruktur leicht verständlich und gut dokumentiert ist.
- Risiken bei Qualitätsmängeln: Hohe Wartungskosten und generell erhöhte Fehleranfälligkeit der Applikation.

Erweiterbarkeit

- Vorteil: Durch qualitativ hochwertigen Quellcode und angemessene Dokumentation ist sichergestellt, dass Entwicklungen über den gesamten Lebenszyklus hinweg mit vertretbarem Aufwand erweitert und angepasst werden können.
- Risiken bei Qualitätsmängeln: Mehraufwand durch Analyse und Nachdokumentation bei Erweiterung vorhandener Funktionalität, Implementierung von Seiteneffekten durch falsche Interpretation des vorhandenen Quellcodes.

7.1.2 ERSTELLUNG UND PFLEGE DES PROZESSES

Für die Umsetzung dieser Verfahren in der Praxis hat sich die formalisierte Beschreibung eines Prozesses bewährt. Dazu zählen klare Verfahrensanweisungen und Verantwortlichkeiten. Die genaue Ausprägung des Prozesses ist unternehmensspezifisch und kann in diesem Leitfaden nicht abgebildet werden; der Verweis auf die Notwendigkeit ist jedoch allgemeingültig.

BEST PRACTICE

Definieren Sie den einzuhaltenden Prozess und dokumentieren Sie ihn in einer für alle zugänglichen Form. Definieren Sie auch, wie Änderungen und Verbesserungen am Prozess stattfinden sollen und wie Anmerkungen und Kritik eingebracht werden können. Dokumentieren Sie alle geprüften Regeln ausführlich mit den Kapiteln Hintergrund/Motivation, schlechte Beispiele, gute Beispiele, Hinweise zum Vorgehen zur Beseitigung der Qualitätsprobleme und Literatur bzw. Ansprechpartner im Unternehmen.

Motivation

Ein Prozess für Best Practices bei der Entwicklung hilft, die Qualität von Software proaktiv und effizient zu verbessern und damit Kosten im Unternehmen langfristig zu senken. Je früher ein Fehler bei der Entwicklung erkannt wird, desto einfacher und kostensparender kann er behoben werden. Je weniger Fehler eine Anwendung enthält, desto mehr entspricht ihre Nutzung den Erwartungen des Unternehmens. Insbesondere läuft sie ohne Nebeneffekte, die das Business negativ beeinflussen.

Welche Aspekte sind für den Prozess relevant?

Interne Entwicklung

Für die interne Entwicklung werden Richtlinien als Nachschlagewerk für die tägliche Arbeit und regelmäßige Trainings für aktuelle Risiken benötigt.

Externe Entwicklung

Bei externer Entwicklung sind klare Qualitätsvorgaben für Ausschreibungen nötig. Vor der Abnahme müssen die Anforderungen auch überprüft werden.

BEST PRACTICE

Die Vorgaben aus dem Prozess müssen möglichst weitgehend mit geeigneten Tools überwacht werden. Für alle Vorgaben, die nicht durch Tools validiert werden können, muss ein manueller, ggf. stichprobenhafter Prüfprozess eingeführt werden. Hierzu eignen sich z.B. Mechanismen wie das Pair-Programming[49] oder iterative Code-Audits [50]. Die manuelle Prüfung bezieht sich hauptsächlich auf die Bereiche der Softwarestrukturierung / -architektur, Wartbarkeit und Erweiterbarkeit, die bei langlebigen Applikationen einen hohen Kostentreiber darstellen können. Daher empfehlen wir neben der Durchführung von maschinellen Prüfungen auch die manuellen Stichproben frühzeitig und iterativ durchzuführen. Dieses Vorgehen bewährt sich vor allem bei größeren Projekten, da Fehlentwicklungen frühzeitig erkannt und mit Gegenmaßnahmen minimiert werden können.

Bei jeder Regel, die zur nachträglichen Qualitätssicherung herangezogen werden soll, muss festgelegt werden, wie diese werkzeuggestützt oder manuell überprüft werden kann. Wenn keine mechanische Überprüfung durch ein Werkzeug möglich ist, wird es in der Praxis mit organisatorischem Aufwand verbunden sein, auf die konsequente Einhaltung der Regel zu achten.

Formale Code-Reviews verursachen erhebliche Aufwände. Sowohl für die Durchführung selbst als auch für die Vor- und Nachbereitung einschließlich der Kontrolle von Korrekturen. Deshalb müssen Sie sich in aller Regel auf wenige nicht maschinell prüfbare Aspekte kritischer Entwicklungsobjekte beschränken. Wenn z.B. die Einhaltung von Performance-Vorgaben eine hohe Priorität besitzt, sollten in entsprechenden Code-Reviews nur Entwicklungsobjekte mit Zugriffen auf Datenbanken oder mit umfangreichen Berechnungen betrachtet werden.

7.2 DURCHSETZBARKEIT

7.2.1 MANUELLE PRÜFUNGEN

Viele der Prüfungen lassen sich automatisieren. Es gibt jedoch Bereiche, die sich für eine automatische Prüfung nicht eignen, wie beispielsweise Dokumentation, Architektur oder viele funktionale Anforderungen. Die menschliche Sprache ist sehr komplex, daher muss z.B. der Inhalt von Dokumenten und Dokumentationen manuell geprüft werden. Nur der menschliche Leser kann beurteilen, ob ein Text sinnvoll, vollständig, verständlich und korrekt ist. Eine automatische Prüfung kann dabei maximal die Existenz in den vorgegebenen Sprachen prüfen. Es empfiehlt sich dennoch ein automatischer Test auf nicht-funktionale Aspekte.

Für die manuelle Prüfung ist es empfehlenswert, eine vollständige Prüfung anhand der Auswertung von Transportstücklisten vorzunehmen. Dabei ist zu berücksichtigen, welche unternehmensinternen Richtlinien existieren. Abhängig von der Anzahl der Objekte muss eine vollständige oder zumindest stichprobenartige Prüfung erfolgen. Das Prüfergebnis wird dem Zuständigen zur Verbesserung oder Vervollständigung der Dokumente bzw. Dokumentationen übermittelt.

Ob ein Prozess alle Vorgaben erfüllt, kann nur mittels einer manuellen periodischen Prüfung ermittelt werden. Falls sich Vorgaben ändern bzw. Mängel im Prozess aufgedeckt werden, ist der Prozess entsprechend anzupassen oder ggf. neu zu definieren.

In der Praxis hat sich ein fest eingeplanter zyklischer Review des Prozesses bewährt.

49 Vgl. Dami Meyer Blog „Pair Programming"
50 Vgl. Wikipedia „Code-Audit"

Wann und wie sollte geprüft werden?

Die den Prüfungen zugrunde liegenden Konzepte müssen regelmäßig auf Aktualität und Konformität bzgl. der Vorgaben geprüft werden. Die Aktualität ist zusätzlich mit einem Upgrade auf ein neues Release (Enhancement Package) zu prüfen. Bzgl. der Vorgaben ist es durchaus sinnvoll, das Wirtschaftsprüfungsunternehmen hinzuzuziehen, das das Unternehmen prüft.

Für extern entwickelten Quellcode müssen diese Prüfungen vor der jeweiligen Abnahme durchgeführt werden.

Für die Akzeptanz der Prüfungen bzw. der Beanstandungen im Zuge der manuellen Prüfungen ist es sinnvoll, im Rahmen der Entwickler- und Qualitätssicherungstests (4-Augen-Prinzip) die manuell notwendigen Prüfungen durch andere Entwickler durchführen zu lassen.

Dasselbe gilt für Penetrations- und Belastungstests. Da es sich bei Penetrationstests auch um ein kritisches Sicherheitsthema handelt, kann es notwendig sein, hierfür in regelmäßigen Abständen auch externe Partner hinzuzuziehen.

7.2.2 AUTOMATISCHE PRÜFUNGEN

Automatische Prüfungen durch statische Code-Analysen (mit Code Inspector/ATC oder Drittanbieter-Tools) decken schnell einen Großteil der notwendigen Tests und Prüfungen ab. Als Hintergrundjob eingeplant, ist eine regelmäßige Wiederholung ohne zusätzlichen Aufwand möglich. Diese regelmäßig mit gleicher Qualität durchgeführten Tests ermöglichen so den Entwicklern, ihren Programmierstil zu verbessern.

Wann und wie sollte geprüft werden?

Die Entwickler sollen ein möglichst zeitnahes Feedback bzgl. der Konformität der Entwicklungen mit den Richtlinien erhalten. Dazu dienen täglich eingeplante Prüfungen im Entwicklungssystem, deren Ergebnis dem Entwickler zur Verfügung gestellt wird.

Wichtig ist, dass dieselben Tests und Metriken bei der Prüfung durch jeden einzelnen Entwickler und bei der Prüfung durch zentrale QS-Instanzen bzw. bei einer Transportfreigabe verwendet werden. Sollten für diese Prüfungen unterschiedliche Werkzeuge oder unterschiedliche Einstellungen verwendet werden, sinkt die Akzeptanz in Entwicklerkreisen ganz erheblich.

Als zentrale Schutzinstanz sollten die Prüfungen in das Transportmanagement integriert und spätestens mit der Freigabe des Transportauftrags (optimalerweise aber bereits mit der Freigabe der einzelnen Transportaufgaben) implementiert sein. Damit wird sichergestellt, dass keine ungeprüften bzw. nicht den Richtlinien entsprechenden Entwicklungen in die Folgesysteme und dann auch in das Produktivsystem transportiert werden.

Als „letztes Sicherheitsnetz" ist ein regelmäßiger Prüflauf (Fullscan) im Produktivsystem möglich. Dieser sollte in einer lastarmen Zeit mittels eines Hintergrundjobs eingeplant werden. Das Ergebnis wird dem QA-Zuständigen zur Verfügung gestellt, der dann die weiteren Schritte (ggf. Korrektur) mit hoher Priorität veranlasst.

Bei der Einführung von automatischen Prüfungen ist im Vorfeld zu definieren, wie mit altem Quellcode umgegangen wird. Sinnvoll kann es hierfür sein, einen Fahrplan zu erstellen, wann und wie die neuen Regeln auf welchen alten Quellcode angewendet werden.

Allerdings müssen firmenintern Kosten und Nutzen einer systematischen Überarbeitung von altem Quellcode abgewogen werden. Bei den Kosten muss auch der Testaufwand durch den Fachbereich sowie das Risiko neuer Fehler betrachtet werden.

BEST PRACTICE

Für den Umgang mit altem Quellcode empfehlen wir, die Prüfergebnisse des alten Quellcodes bei Beginn der regelmäßigen Prüfläufe festzuhalten und bei jedem weiteren Prüflauf zu überwachen, dass im Vergleich zur initialen Prüfung keine Verschlechterung der Prüfergebnisse eingetreten ist.

SAP bietet einige Werkzeuge, mit welchen die automatisierten Prüfungen durchgeführt werden können. Eine Übersicht dieser Werkzeuge ist in Abschnitt 2.9 zu finden.

7.3 ERFAHRUNGEN UND TIPPS AUS DER PRAXIS

7.3.1 QUALITÄTSSICHERUNG DES QUELLCODES

Um eine erfolgreiche Einführung der Qualitätssicherung zu gewährleisten, ist ein schrittweises und behutsames Vorgehen wichtig. Es empfiehlt sich, einen zweigeteilten Ansatz zu fahren. Zunächst sollte neuer Quellcode „fehlerfrei" erstellt und dies geprüft werden. Erst wenn sich dieser Prozess stabilisiert hat, sollte nach und nach bestehender Quellcode mit in die Checks aufgenommen werden. Ansonsten wird der zu bewältigende Berg für die Entwickler zu groß und die Motivation sinkt rapide.

Wenn die automatischen Codeprüfungen nicht auf der „grünen Wiese" mit einer neuen Entwicklung eingeführt werden, sollte vorher der Umgang mit „Altlasten" geklärt werden. Auch bei neuen Entwicklungen lassen sich Änderungen an schon bestehenden Objekten nicht vermeiden, die dann bei einer eingerichteten Transportprüfung zu Problemen führen können. Hilfreich ist in solchen Fällen eine Klärung der Verantwortlichkeit für Entwicklungsobjekte, die z.B. durch das entsprechende Feld in Paketdefinitionen dokumentiert werden kann. Diese Verantwortlichen müssen entscheiden, ob Fehler im bestehenden Quellcode sofort korrigiert werden müssen oder eine Ausnahmeregelung möglich ist.

Um die beschriebenen Probleme zu vermeiden, ist alternativ ein Konzept denkbar, in dem für älteres Coding keine Verschlechterung akzeptiert wird, in dem also in einer automatischen Qualitätsprüfung keine neuen Befunde auftreten dürfen. SAP arbeitet an einer Umsetzung dieses Konzepts für den ATC. Bis dahin gibt es einen SCN-Blog, der eine kundenindividuelle Umsetzung beschreibt[51]. Vorteil ist, dass hierbei auch z.B. neue Methoden in existierenden Klassen geprüft werden.

Generell ist es in vielen Fällen schon ausreichend, mit Standard-Bordmitteln, wie z.B. dem ABAP-Test-Cockpit, zu arbeiten, der sich auch um eigene Checks erweitern und somit an eigene Bedürfnisse anpassen lässt.

Eine weitere Strategie besteht darin, für Neuentwicklungen die Qualitätsanforderungen in den Prozess der Anforderungsanalyse mit einzubinden und direkt an das zu erstellende Objekt zu koppeln. Hierdurch lässt sich eine Unterteilung in IT-Governance getriebene und anforderungsspezifische Qualitätskriterien erreichen. Während die IT-Governance getriebenen Qualitätsanforderungen (z.B. die Sicherheitskriterien) verpflichtend zu erfüllen sind, ist für die Beurteilung und Ausnahmebestimmung anderer Kriterien der ABAP-versierte Produktverantwortliche zuständig. Technisch lässt sich das Konzept realisieren, indem der Produktverantwortliche in dem Paket hinterlegt wird und eine Versorgung des Pakets bzw. der enthaltenen Quellcode-Objekte mit manuell angelegten Klassifikationsattributen aus dem Classification Toolset[52] stattfindet. Die Ausprägung der Klassifikationsattribute sollte auf Faktoren wie z.B. erwartete Lebensdauer, Nutzungshäufigkeit, Kritikalität, Umfang, etc. basieren. Als Resultat ergibt sich ein Szenario, das die zentralen Qualitätsbeauftragten entlastet und die Teilverantwortung an das Entwicklungs- / Projektteam delegiert.

BEST PRACTICE

Qualitätssicherung über Zwangsprüfungen wie die ATC/Code-Inspector-Prüfung bei Transportfreigabe sind eine wirksame Möglichkeit Qualitätssicherung zu gewährleisten. Allerdings führt dies zu einer gewissen Ineffizienz, da zu diesem Zeitpunkt eventuell bereits fertiger Quellcode überarbeitet werden muss. Sinnvoller ist es, wenn Entwickler von vornherein Quellcode mit hoher Qualität liefern. Dies kann zum Beispiel durch die Anwendung von Clean-Code-Kriterien[53] erfolgen. Wird in einem Team grundsätzlich nach diesen Kriterien entwickelt, reduziert dies den nachträglichen Anpassungsaufwand.

7.3.2 TIME AND BUDGET QUALITY ASSURANCE (QA)

Um die Zeit und den Aufwand für die QA-Tätigkeiten möglichst gering zu halten, muss der Entwickler den Quellcode während seiner Entwicklertätigkeit selbstständig auf Fehler hin untersuchen können.

Tut er dies nicht, muss er automatisiert auf Fehler oder Verbesserungsmöglichkeiten hingewiesen werden. Tägliche Inspektionen mit einem entsprechenden Werkzeug und Verteilung der Ergebnisse stellen sicher, dass Fehler frühzeitig erkannt werden und die Entwickler sich noch an ihre Tätigkeiten vom Vortag erinnern, was die Fehlerbehebung deutlich vereinfacht. So wird gewährleistet, dass auch Entwickler, die den manuellen Aufwand scheuen oder unter Zeitdruck stehen, trotzdem die Möglichkeit erhalten, ihre Fehler im Quellcode zu beheben. Es gilt: Je später die QA einsetzt, desto

51 http://scn.sap.com/community/abap/blog/2016/05/23/how-to-filter-atc-findings-to-detect-only-new-findings

52 SAP-Help-Suche „Classification Toolset"

53 Siehe auch http://clean-code-developer.de/

höher ist der Aufwand für die Fehlerbehebung. Dieser zusätzliche Aufwand entsteht z.B. durch zusätzliche Transporte, wenn der Originaltransport bereits freigegeben wurde.

Deshalb ist es wichtig, bei Planung und Schätzung eines Projekts die Qualitätssicherung zu berücksichtigen, und zwar nicht nur am Ende, sondern projektbegleitend und anschließend im kompletten Software-Lifecycle.

Nicht zu unterschätzen ist auch der notwendige Schulungsaufwand, um bei den Entwicklern um Verständnis für den Prozess zu werben.

Bei einem Einsatz von externen Entwicklern müssen die Programmierrichtlinien und Namenskonventionen Bestandteil des Vertrags sein.

7.3.3 PROBLEME

Bei der Einführung einer Quellcode-QA treten eine Reihe von Problemen auf, auf die hier kurz eingegangen wird.

Ein Reibungspunkt ergibt sich durch die Frage, wer für die QA zuständig ist: Ersteller oder Änderer. Bei neuen Sourcen ist dies kein Problem, bei bestehendem Quellcode wird die Frage aber immer wieder aufgeworfen:

„Warum soll ich Quellcode überprüfen, in dem ich nur eine Zeile geändert habe?“

vs.

„Warum soll ich Quellcode überprüfen, den ich schon Jahre nicht mehr angefasst habe?“

Beide Positionen sind verständlich, deswegen muss eine klare Entscheidung bezüglich Handhabung von existierendem Quellcode, der auf anderen/keinen Konventionen beruht, getroffen und durchgesetzt werden. Wenn der Ersteller noch verfügbar ist, empfiehlt sich, dass dieser in die QA miteinbezogen wird.

Auch folgender Fall ist problematisch: Im Rahmen der Wartung erfolgt eine kleine Änderung an einem Include-Programm. Durch eine automatische Qualitätsprüfung werden dabei Qualitätsfehler im Rahmenprogramm gefunden (bei mehrfach genutzten Include-Programmen evtl. sogar in mehreren Rahmenprogrammen). Die Korrekturen erfordern – je nach Umfeld – einen aufwändigen zusätzlichen Test dieser Rahmenprogramme durch den Fachbereich.

BEST PRACTICE

Um bei auftretenden Problemen die Fragen der Entwickler beantworten zu können, sollte eine zentrale Stelle geschaffen werden. Außerdem muss es einen Prozess geben, über den in Notfällen auch fehlerbehaftete Transporte freigegeben werden können. Gibt es diese Möglichkeit nicht, sinkt das Verständnis für die durchgeführten Maßnahmen. Es kann zum Beispiel ein Genehmigungsprozess installiert werden, mit dessen Hilfe auch fehlerbehafteter Quellcode freigegeben oder transportiert werden kann. Die meisten Tools am Markt (einschließlich SAP Code Inspector und ABAP-Test-Cockpit) bieten diese Möglichkeit standardmäßig an.

7.3.4 ENTSCHEIDUNGSFINDUNG BEI MODIFIKATIONEN

Die Hürde für Modifikationen sollte so hoch wie möglich gelegt werden. Dies ist besonders wichtig, wenn zeitnah Enhancement Packages der SAP eingespielt werden sollen (SPAU-Problematik siehe dazu Abschnitt 8.4). Als Ansatzpunkt hierzu kann der Modifikationsschlüssel dienen. Die Anzahl der Entwickler mit der Berechtigung, Modifikationsschlüssel zu generieren, muss möglichst gering sein. So ist es möglich, steuernd einzugreifen und die Modifikationen über Change Requests und einen zugehörigen Prozess abzuhandeln. Da durch modifikationsfreie Erweiterungen kein SAP-Coding geändert wird und dadurch bei Upgrades potenziell weniger Probleme entstehen, sind sie Modifikationen grundsätzlich vorzuziehen.

Die Frage, wodurch sich eine Modifikation rechtfertigt, ist unternehmensindividuell zu beantworten und konsequent umzusetzen. Eine Pauschalantwort auf diese Frage gibt es nicht. In jedem Fall sollte die Entscheidung jedoch auf gleichartigen, im Vorfeld definierten und kommunizierten Kriterien basieren.

Zum Thema Alternativen zu Modifikationen siehe Abschnitt 8.4.

BEST PRACTICE

Wenn SAP-Coding ergänzt oder geändert werden muss, stehen vier Möglichkeiten zur Verfügung: explizite und implizite Enhancements, Modifikationen und Kopieren des SAP-Objekts in den kundeneigenen Namensraum. Die einzelnen Optionen sollten in folgender Reihenfolge in Erwägung gezogen werden: explizite Enhancements vor impliziten Enhancements, implizite Enhancements vor Modifikationen und Modifikationen vor Kopien. Besonders kritisch sind Kopien, weil dadurch eventuelle Fehler mit kopiert werden, der kopierte Quellcode aber nicht mehr der SAP-Wartung durch Hinweise und Updates unterliegt.

7.3.5 ERFAHRUNGSBERICHT AUS DER PRAXIS: COMGROUP GMBH

Das nachfolgende Beispiel der Comgroup GmbH, dem IT-Dienstleister der Würth Gruppe, zeigt, wie Programmierrichtlinien und Namenskonventionen in einer Softwareentwicklung ein- und durchgesetzt werden können.

Die Comgroup GmbH startete mit der Quellcode QA im globalen Entwicklungssystem einer mehrstufigen Entwicklungslandschaft. Zur automatisierten Unterstützung wurde der Code Inspector eingesetzt. Da zeitgleich mit der Quellcode-QA auch ein neuer Namensraum eingeführt wurde, wurden zu Beginn des Projekts nur Objekte aus dem neuen Namensraum geprüft und bestehender Quellcode nicht berücksichtigt. Dies erleichterte die Selektion im Code Inspector, da diese keine Änderungs- oder Erstellungsdaten berücksichtigt. Außerdem wurden Performance- und Security-Checks aus dem Code Inspector vorerst nicht aktiviert, um den Aufwand in einem überschaubaren Rahmen zu halten.

Mit diesem Tool kann der Entwickler seinen Quellcode während der Entwicklung selbstständig überprüfen. Zusätzlich wurde ein Nachtlauf eingeplant, der den kompletten zu scannenden Quellcode analysiert und bei gefundenen Fehlern Mails an den jeweiligen Entwickler sendet.

Fehlende E-Mail-Adressen in Userstämmen führten zu Beginn zu Problemen, da die Mails nicht zugestellt werden konnten. Solche Mails werden nun an eine zentrale Adresse versendet. Unvollständige Datensätze können so mit einem geringen Aufwand identifiziert werden. Außerdem werden in diesem System nun keine Entwicklungs-User mehr ohne Mailadresse angelegt.

Die Mails werden nicht von einem anonymen Batch-User sondern von der Mailadresse des QA-Verantwortlichen gesendet, um dem Entwickler eine einfache Möglichkeit zu geben, Fragen zu stellen. Zu Beginn entstand hierdurch ein hoher Aufwand, die Anzahl der Fragen verringerte sich aber mit der Laufzeit des Projekts. Dies konnte durch Schulungen erreicht werden, durch die die Entwickler effizienter Fehler korrigieren bzw. diese schon bei der Entwicklung vermeiden können.

Die Entwicklungssprache in der Entwicklungslandschaft ist Englisch. Dies wird durch einen weiteren Job geprüft und Fehler dem Entwickler gemeldet. SAP bietet im Standard keine Möglichkeit, bei der Anlage eines Entwicklungsobjekts eine vorgegebene Sprache zu setzen. Deshalb gab es nur die beiden Möglichkeiten, entweder an passender Stelle eine entsprechende Prüflogik per Modifikation einzubauen oder damit zu leben, dass Objekte in falscher Sprache angelegt werden und im Nachgang umgezogen werden müssen.

Bei Anlage von Objekten werden zudem per Modifikation Namenskonventionen überprüft, sodass diese nur gemäß den Konventionen benannt werden können.

Um eigene Prüfungen bei der Transportfreigabe durchzuführen (z.B. eigene Namenskonventionen/mindestens deutsche und englische Übersetzung) wurde eine Implementierung für das Business Add In CTS_REQUEST_CHECK angelegt und die Methode CHECK_BEFORE_RELEASE genutzt.

Nachdem sich der Prozess im globalen Entwicklungssystem stabilisiert hatte, wurde dieser an die nachfolgenden Entwicklungssysteme und Namensräume ausgerollt. Bestehender Quellcode wird bisher noch nicht gecheckt. Zusätzlich ist geplant, ein externes Tool einzusetzen, das die Qualitätssicherung vereinfacht.

8 INFRASTRUKTUR UND LIFECYCLE MANAGEMENT

In diesem Kapitel stehen die Infrastruktur und die Betrachtung des Lebenszyklus einer Softwarekomponente im Fokus. Sie stellen neben methodischen Empfehlungen und Werkzeugen bei der Softwareentwicklung im SAP-System wichtige Rahmenbedingungen für erfolgreiches Arbeiten dar.

8.1 INFRASTRUKTUR

Eine SAP-Systemlandschaft ist i.d.R. mehrstufig aufgebaut. In Abhängigkeit von der Releasestrategie sind zwei sinnvolle Alternativen zu betrachten. Sofern regelmäßig in kurzen Zeitabständen kleinere Änderungen, im Sinne von kurzfristigen Releases, transportiert werden, ist eine klassische 3-Systemlandschaft zu präferieren. Kommt hingegen eine Release-Strategie mit langen Release-Zyklen zum Einsatz, spricht dies für eine 5- bzw. 6-Systemlandschaft. Nachfolgend werden die einzelnen Systeme und ihre Bedeutung dargestellt.

8.1.1 KLASSISCHE 3-SYSTEMLANDSCHAFT

8.1.1.1 Entwicklung

Im Entwicklungssystem wird entwickelt, Customizing geändert und es werden erste Entwicklertests durchgeführt. Aufgrund einer durch laufende Entwicklungen und Customizing-Tätigkeiten verursachten Instabilität, ist ein Test durch Dritte (Fachbereich) in diesem System nicht sinnvoll.

Entwickler und Modulbetreuer haben im Entwicklungssystem sehr weitreichende Berechtigungen. Meist gibt es nur wenige Testdaten.

8.1.1.2 Qualitätssicherung

Im Qualitätssicherungssystem herrscht Customizing- und Entwicklungsverbot (Systemeinstellung „nicht änderbar"). Customizing- und Workbench-Objekte werden ausschließlich über Transporte importiert[54]. Importe nach Produktion erfolgen grundsätzlich über das Qualitätssicherungssystem. Damit wird eine mit der Produktion übereinstimmende Systemumgebung sichergestellt.

Das Qualitätssicherungssystem sollte für z.B. umfangreichere Validierungsaktivitäten regelmäßig aus dem Produktivsystem kopiert werden. Damit wird die Übereinstimmung mit der operativen (Daten-)Umgebung sichergestellt. Im Qualitätssicherungssystem erfolgt die Abarbeitung von Testplänen nach Änderungen und Neuentwicklungen.

Entwickler und Modulbetreuer haben im Qualitätssicherungssystem weitreichende Berechtigungen. Einschränkungen müssen individuell festgelegt werden, z.B. für besonders sensible Daten (HR etc.). In Zusammenhang mit personenbezogenen Daten ist der Datenschutz zu berücksichtigen[55]. Es empfiehlt sich, zusätzlich Benutzer mit produktionsidentischen Berechtigungen zu verwenden, um hier auch den Aspekt Berechtigung zu testen.

8.1.1.3 Produktion

Im Produktivsystem herrscht ebenfalls Customizing- und Entwicklungsverbot. Customizing- und Workbench-Objekte werden ausschließlich per Transportauftrag in dieses System importiert.

Entwickler und Modulbetreuer haben im Produktivsystem nur eingeschränkte Berechtigungen. Notfall-Tabellenänderungen (&SAP_EDIT) sind mit Datum, Uhrzeit und Begründung zu dokumentieren. Hierzu empfiehlt sich die Verwendung eines Tools, um die Meldungen zu standardisieren. Der Umgang mit speziellen Notfall-Benutzern, die weitergehende Berechtigungen haben, muss technisch und organisatorisch festgelegt werden.

8.1.2 5- BZW. 6-SYSTEMLANDSCHAFT

Wenn lange Release-Zyklen zum Einsatz kommen, ist diese Landschaft sinnvoll. Im Entwicklungssystem erfolgt die Entwicklung des neuen Releases. Die Wartung bzw. Fehlerbearbeitung des produktiven Releases erfolgt in zwei separaten Systemen.

8.1.2.1 Entwicklung

Das Entwicklungssystem entspricht in dieser Landschaft dem Entwicklungssystem der 3-Systemlandschaft.

54 *Auch in der 3-Systeme-Landschaft kann mit Transporten von Kopien gearbeitet werden, wie in Abschnitt 8.1.4 Best Pratice beschrieben.*

55 *Vgl. DSAG-Leitfaden Datenschutz*

8.1.2.2 Test

Im Testsystem herrscht absolutes Customizing- und Entwicklungsverbot. Customizing- und Workbench-Objekte werden ausschließlich per Transportauftrag importiert.

Importe in dieses System erfolgen im Regelfall durch Transporte von Kopien (siehe Abschnitt 8.1.4 Best Pratice). So wird gewährleistet, dass bearbeitete Entwicklungsobjekte im Entwicklungssystem über den gesamten Release-Zeitraum gesperrt bleiben. Der Original-Transportauftrag wird erst zur Produktivsetzung freigegeben. Durch die Transporte von Kopien können Entwicklungen und Customizing schon vorab im Testsystem getestet werden. Damit ist es möglich, viele Entwicklungen und Customizing im Entwicklungssystem zu bündeln und so die Anzahl der Release-Transporte für das Qualitätssicherungs- und das Produktivsystem zu minimieren. Wenn dieser Vorteil nicht benötigt wird, kann auf das Testsystem verzichtet werden.

Das Testsystem wird bei Bedarf aus dem Produktivsystem kopiert. So ist es möglich, bereits vorab umfangreiche Tests durchzuführen.

Entwickler und Modulbetreuer haben im Testsystem sehr weitreichende Berechtigungen. Einschränkungen müssen individuell festgelegt werden, z.B. für besonders sensible Daten (HR etc.). Es empfiehlt sich aber auch, Testbenutzer mit produktionsnahen Berechtigungen zu verwenden.

8.1.2.3 Qualitätssicherung

Das Qualitätssicherungssystem entspricht in dieser Landschaft dem Qualitätssicherungssystem der 3-Systemlandschaft.

Die Versorgung des Qualitätssicherungssystem erfolgt:

- in einer 6-Systemlandschaft nur mit Releasetransporten,
- in einer 5-Systemlandschaft mit den laufenden normalen Transportaufträgen.

8.1.2.4 Wartung

Im Wartungssystem erfolgt die Wartung bzw. Betreuung der produktiven Software für den Zeitraum, in dem im Entwicklungssystem das neue Release entwickelt wird. Die Übernahme aller Korrekturen aus dem Wartungssystem in das neue Release muss sichergestellt werden.

Nach einem Produktivgang, d. h. dem Transport eines Releases auf die Produktion, muss das Wartungssystem wieder konsistent mit der Produktion werden. Dies kann auf unterschiedlichen Wegen erfolgen. Ein typisches Vorgehen ist der Import der Releasetransporte auf das Wartungssystem.

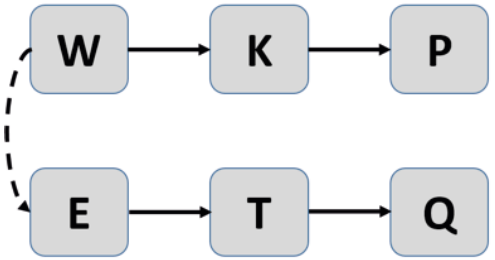
8.1.2.5 Konsolidierung

In diesem System erfolgt der Test der Änderungen aus dem Wartungssystem.

8.1.2.6 Produktion

Das Produktivsystem entspricht in dieser Landschaft dem Produktivsystem der 3-Systemlandschaft.

8.1.2.7 Schematische Darstellung 6-Systemlandschaft



System	Verwendung
E	Entwicklung
T	Test
Q	Qualitätssicherung
P	Produktion
W	Wartung
K	Konsolidierung

Abbildung 2: Schematische Darstellung 6-Systemlandschaft

8.1.3 SANDBOX

Die Sandbox ist ein reines Test- und „Spielsystem". In der Sandbox gelten keine Einschränkungen in Bezug auf Berechtigungen, das gilt gleichermaßen für Customizing- und Workbench-Entwicklungen. Aus der Sandbox erfolgen keine Transporte in andere Systeme. Komplexe Änderungen können so vor der Durchführung im Entwicklungssystem ausprobiert werden. Der Rückbau komplexer Entwicklungen in der Entwicklungsumgebung, z.B. wenn eine Neuentwicklung nicht weiter verfolgt wird, kann so vermieden werden. Zur Bereinigung der prototypischen Entwicklungen im Sandbox-System sollte es regelmäßig neu aufgesetzt werden, z.B. als Kopie aus dem Produktivsystem.

8.1.4 TRANSPORTWESEN

Der Transportweg ist wie folgt festgelegt:

- 3-Systemlandschaft:

  Entwicklung → Qualitätssicherungssystem → Produktion (nach Qualitätssicherungssystem evtl. mit Transporten von Kopien)

- 5-/6-Systemlandschaft:

  Entwicklung → Test (in dem Fall immer Transporte von Kopien)

  Entwicklung → Test -> Qualitätssicherungssystem → Produktion (Release-Transporte)

  Wartung → Konsolidierung → Produktion (Wartung/Betreuung während Release-Entwicklung)

Die Sandbox sollte aus dem Transportweg ausgeschlossen werden. Importe aus dem Entwicklungssystem in die Sandbox erfolgen nur auf individuelle Anforderung.

Die Freigabe von Transporten im Entwicklungssystem erfolgt üblicherweise durch den Entwickler. Bei mehreren mit Transportaufgaben an einem Thema beteiligten Entwicklern gibt ein Entwickler in der Rolle des Koordinators den Transportauftrag frei. Es können automatische Qualitätsprüfungen zum Beispiel mit dem ABAP-Test-Cockpit (siehe Abschnitt 2.9) oder dem Code Inspector (siehe Abschnitt 7.2.2 „Automatische Prüfungen") konfiguriert werden, die die Transportfreigabe verhindern können und Nachbesserungen oder eine Ausnahmegenehmigung erfordern.

Die Importe in Test- bzw. Qualitätssicherungssysteme müssen in Abhängigkeit von der Systemlandschaft organisiert werden. Oft werden Importe in diese Systeme zyklisch und automatisiert durchgeführt, um die Mitarbeiter der Basisabteilung zu entlasten und Wartezeiten für Entwickler und Tester zu reduzieren.

Wenn mehrere Entwickler an einem Entwicklungsobjekt Änderungen vornehmen, kann dies unter Umständen zu Problemen führen, wenn die Transportaufträge nicht in der richtigen Reihenfolge in die produktiven Systeme transportiert werden oder Objekte aus anderen Transportaufträgen fehlen.

BEST PRACTICE

Um Probleme mit der Importreihenfolge von freigegebenen Transporten, die dieselben Entwicklungsobjekte betreffen, zu vermeiden, sollte für die Belieferung der Test- und Qualitätssicherungssysteme mit Transport von Kopien gearbeitet werden. Die geänderten Entwicklungsobjekte bleiben bis zur Produktivsetzung durch den eigentlichen Transportauftrag im Entwicklungssystem gesperrt. Ein Entwickler wird durch das System darauf hingewiesen, wenn ein anderer Entwickler das Objekt bereits bearbeitet, und kann sich mit diesem Entwickler abstimmen. Nach Abschluss des Projekts oder der Änderung wird nur der eigentliche Transportauftrag über das Qualitätssicherungssystem ins Produktivsystem transportiert.

Das im SAP Solution Manager enthaltene Change and Request Management (ChaRM) arbeitet intern mit diesem Verfahren.

Der Import in das Produktivsystem sollte in der Regel durch interne, verantwortliche Mitarbeiter durchgeführt werden. Voraussetzung ist die formale Freigabe (im Sinne einer dokumentierten Entscheidung, siehe Abschnitt 8.2 Change Management). Hierfür wird ebenfalls die Verwendung eines geeigneten Tools (z.B. Solution Manager ChaRM) zur Standardisierung bzw. Formalisierung empfohlen.

8.1.5 SICHERSTELLUNG DER KONSISTENZ VON NEUENTWICKLUNGEN UND ERWEITERUNGEN

Bei parallel laufenden Projekten besteht die Gefahr von Überschneidungen. Es kann zur überschneidenden Verwendung von Objekten kommen, die es im Zielsystem (noch) nicht gibt. Dies führt zum Fehler beim Import. Hieraus ergibt sich die Pflicht zur Prüfung der von Dritten (Non-SAP) erstellten Objekte bei deren Verwendung. Neuentwicklungen bzw. Erweiterungen müssen in geeigneten Paketen bzw. Transportaufträgen gekapselt werden. Es wird empfohlen, die Transportaufträge für ein Projekt auf je einen Transportauftrag für Workbench, Customizing und Berechtigungsrollen einzuschränken. „Vorabtransporte" sollten nur mit Hilfe von Transport von Kopien erlaubt sein.

Die endgültige Freigabe und der Transport erfolgt erst zum Projektabschluss. Alle Projektmitarbeiter verwenden innerhalb eines Projekts nur Aufgaben zu einem jeweils vorgegebenen Transportauftrag. Es gibt keine „persönlichen" Transportaufträge für Projektmitarbeiter.

Generell erfolgt ein Import nur nach formeller Freigabe (siehe Change-Verfahren) durch einen Process Owner, die Qualitätssicherung oder ähnliche Instanzen. Die Abfolge sowie die beteiligten Bereiche müssen unternehmensspezifisch festgelegt werden.

8.1.6 RÜCKBAU VON NEUENTWICKLUNGEN

Wurden im Entwicklungssystem Entwicklungen erstellt, die im produktiven Release nicht nutzbar bzw. unzugänglich sein sollen, so existieren verschiedene Möglichkeiten, dies zu erreichen:

- Kompletter Rückbau der Änderungen, sofern diese auch in späteren Releases nicht benötigt werden.
- Deaktivierung der Entwicklung, so dass der Quellcode im Produktivsystem nicht durchlaufen wird. Dies kann durch unterschiedliche Strategien erreicht werden:
  - Durch Auskommentieren des Quellcodes (bei kleineren Code-Anderungen).
  - Durch Einbauen einer Weiche im Quellcode, die sicherstellt, dass der nicht freigegebene Quellcode nicht durchlaufen wird. Die Ausgestaltung der Weiche hängt vom konkreten Fall ab.
  - Customizing muss ggf. komplett entfernt oder zurückgebaut werden.

Im Falle der Nutzung von TMS Quality Assurance (QA)[56] als Genehmigungsworkflow gilt:

- Die Ablehnung durch den Qualitätsverantwortlichen verbietet lediglich den Transport der Funktionalität in nutzbarer bzw. zugänglicher Form ins Produktivsystem. Der Quellcode kann zwar geliefert werden, aber er darf in keinem Fall durchlaufen werden.
- Ein Prozess muss sicherstellen, dass die abgelehnte Entwicklung in allen relevanten Systemen in einen nicht zugänglichen Stand überführt wird. Wie oben bereits erwähnt kann dies durch einen vollständigen Rückbau oder eine Deaktivierung über Weichen im Quellcode erreicht werden.
- Die zuständigen Entwickler müssen von abgelehnten Transporten erfahren. Die Pflicht zur Informationsweitergabe liegt bei den Qualitätsverantwortlichen.

Zum Rückbau von „verwaisten" Entwicklungen ist im SCN ein kleines Z-Programm vorgestellt worden (Report for Rolling Back Abandoned Developments).

56 *https://help.sap.com/saphelp_nw70/helpdata/de/9c/a544c6c57111d2b438006094b9ea64/content.htm*

8.2 CHANGE MANAGEMENT

Um eine SAP-Systemlandschaft wartbar und beherrschbar zu gestalten, ist ein formales Change-Verfahren die Voraussetzung für jede Systemänderung. Dabei gilt das grundsätzliche Verfahren gleichermaßen für Änderungen am SAP-Standard und für Kundenentwicklungen.

Das Basiswerk zur Einführung eines Change- und Releasemanagements ist die Information Technology Infrastructure Library (ITIL)[57]. In diesem Referenzleitfaden sind umfangreiche und allgemeingültige Best Practices beschrieben.

In diesem Kapitel stellen wir ein konkretes Beispiel aus der Entwicklung dar, das firmen- bzw. branchenspezifisch adaptiert werden kann.

Wir empfehlen grundsätzlich, die folgenden Aspekte bei der Einführung eines Change-Control-Verfahrens (auch: Change-Request-Verfahren) zu berücksichtigen:

- Fachliche Anforderung
- Begründung
- Bewertung (Aufwandschätzung)
- Genehmigung
- Freigabe der Änderung für das Produktivsystem

Der Genehmiger- bzw. Freigeberkreis (Process Owner, QS ...) ist abhängig vom Gegenstand der Änderung (betroffener Bereich, betroffene Anwendung, gegebenenfalls auch Aufwand).

Die Verwendung eines geeigneten Tools (z.B. Solution Manager ChaRM) wird dringend empfohlen. Dabei gilt jedoch: Eine papierbasierte Lösung ist besser als keine!

Beispiel für ein Change-Control-Formular (CC):

CC Nummer: _____ (von IT ausfüllen) Datum: _____

CC Titel: _____ (von IT ausfüllen)

Anforderung

Anforderer: _____ Kostenstelle: _____

Abteilung: _____

Wuschdatum: _____ Priorität: _____ (niedrig/mittel/hoch)

Änderungsart: _____ (Änderung/Autorisierung)

Änderung: (Kurzbeschreibung) _____

Ausführliche Beschreibung als Anlage anfügen!

Process Owner: _____ Datum/Unterschrift: _____

Process Owner Fremd: _____ Datum/Unterschrift: _____

Bearbeitung

Applikation/Modul: _____ Bearbeiter: _____

Genehmigt SAP-Koordination: _____ Datum/Unterschrift: _____

Genehmigt SAP-Leitung _____ Datum/Unterschrift: _____

Bemerkung: _____

Freigabe/Produktionsübergabe

Geprüft durch Anforderer: _____ (Datum/Unterschrift:)

Geprüft SAP-Koordination: _____ (Datum/Unterschrift:)

Geprüft IT-Leitung: _____ (Datum/Unterschrift:)

Transport/Übergabe durch Bearbeiter: _____ (Datum/Unterschrift:)

Abbildung 3: Change-Control-Formular (CC)

57 siehe https://de.wikipedia.org/wiki/IT_Infrastructure_Library

Das abgebildete CC-Formular enthält beispielhaft die wesentlichen Daten, die begleitend zur Abwicklung einer Systemänderung erforderlich sind.

Basisprozess und Rollen:

- Der Anforderer füllt den Teil „Anfordernde Abteilung" aus und sorgt für die Unterschrift des Process Owners und/oder des Process Owners Fremd
- Der Process Owner ist in der Regel ein Vorgesetzter des Anforderers. Er ist verantwortlich für einen bestimmten Teil der Daten bzw. für die Nutzung bestimmter Teile der SAP-Software, z.B. der Einkaufsleiter für die SAP-Einkaufsdaten und -programme
- Der Process Owner Fremd ist immer dann einzubeziehen, wenn eine Änderung auch Daten oder Programme betrifft, die außerhalb der Zuständigkeit des Process Owners liegen.
  Beispiel: Der Einkäufer benötigt Berechtigungen aus dem Bereich der Anlagenbuchhaltung; hier muss der Process Owner zustimmen, der diesen Teil des Systems verantwortet (z.B. der Leiter der Anlagenbuchhaltung).
- Eine ausführliche Beschreibung der Änderung ist dem CC in jedem Fall als Anlage anzufügen; CCs ohne eine ausführliche Beschreibung werden zurückgewiesen. Der Anforderer gibt das CC an die IT weiter.
- Der SAP-Koordinator ist derjenige, der die anfallenden Aktivitäten für SAP oder einen Teil von SAP koordiniert und den einzelnen Modulbetreuern die Aufgaben zuweist. Diese Aufgabe kann – in Abhängigkeit von der Größe der Organisation – auch von einem Gruppen- oder Abteilungsleiter innerhalb der IT übernommen werden. Die betreffende Person trägt die Applikation bzw. das Modul im Formular ein und weist das CC einem Bearbeiter zu. Sie kann das CC auch aufgrund formaler Fehler (z.B. fehlende oder unzureichende Beschreibung, fehlende Unterschrift des Process Owners oder des Process Owner Fremd) zurückweisen. Der SAP-Koordinator vergibt eine eindeutige CC-Nummer und den CC-Titel; diese CC-Nummer kann z.B. auch von einem Projektmanagement-Tool übernommen werden.
- Der IT-Leiter oder der SAP-Leiter genehmigt / lehnt ab / stellt zurück (mit entsprechender Begründung), nachdem der Koordinator genehmigt hat.
- Im Falle der Genehmigung erhält der Bearbeiter das CC zur weiteren Bearbeitung; eine Änderung zwecks Weitertransport darf von einem Bearbeiter ausschließlich mit einem vollständig genehmigten CC durchgeführt werden.
- Nach Fertigstellung lässt der Bearbeiter die Änderung durch den Anforderer prüfen.
- Entspricht die Umsetzung den Anforderungen, gibt der Anforderer die Änderung zum Transport frei; der Anforderer bestätigt mit seiner Unterschrift die ordnungsgemäße Umsetzung; ein Transport darf erst nach Freigabe durch den Anforderer durchgeführt werden.
- SAP-Koordinator und IT-Leitung bestätigen die ordnungsgemäße Umsetzung. Die ordnungsgemäße Umsetzung sollte anhand einer Checkliste überprüft und durch den Einsatz von Prüfwerkzeugen unterstützt werden (siehe nachfolgende Best Practice-Empfehlungen). Ein Transportauftrag darf nicht ohne vorangegangene Freigabe durch die SAP-Koordination und die IT-Leitung ins Produktivsystem importiert werden.
- Der Bearbeiter übergibt den Transport in das Produktivsystem und leitet das CC an die SAP-Koordination und die IT-Leitung weiter.

Der Genehmigungs- und Freigabeprozess und damit auch der Inhalt des Formulars können branchenspezifisch stark variieren. In pharmazeutischen Unternehmen ist beispielsweise grundsätzlich die QA in das CC-Verfahren eingebunden. Darüber hinaus ist es in allen Unternehmen üblich, dass bei Überschreitung von bestimmten (geschätzten) Projektkostenlimits weitere Genehmiger der Umsetzung zustimmen müssen. Die Genehmigungsstruktur hängt also auch von der jeweiligen Unternehmensorganisation ab.

Das vorgestellte Formular enthält daher lediglich die Minimalanforderungen an ein CC ohne Berücksichtigung branchen- oder organisationsspezifischer Anforderungen.

Weitere Genehmigungs- bzw. Freigabeschritte oder zusätzliche Felder mit Bezug zu anderen Dokumenten (z.B. Validierungsdokumente) sind bei Bedarf individuell zu ergänzen und der Prozess entsprechend zu erweitern.

BEST PRACTICE

Checkliste vor Freigabe eines Workbench-Transports durch den SAP-Koordinator und die IT-Leitung zum Transport in das Produktivsystem:

- Automatische Prüfungen des Quellcodes durchlaufen? Der Code Inspector bzw. ATC oder die erweiterte Programmprüfung wurden für alle Programme des Transportauftrags durchgeführt. Die Ergebnisliste darf dabei keine Fehler oder Warnungen enthalten, Informationsmeldungen sind zulässig. Optimalerweise werden die Prüfungen automatisch bei jeder Transportfreigabe ausgeführt (siehe Abschnitt 2.9).
- Drittanbietertools? Sofern weitere Testtools für Themenbereiche wie Security, Performance, etc. vorhanden sind, wurden diese ausgeführt?
- Gegebenenfalls manuelle Prüfungen, eventuell stichprobenartig.
- Manuelle Vor- oder Nacharbeiten? Es ist zu prüfen, ob eine vollständig abgearbeitete Checkliste für manuelle Vor- und Nacharbeiten zum Transport vorliegt.
- Mehrsprachigkeit? Sofern im Transportauftrag übersetzungsrelevante Objekte enthalten sind, ist zu prüfen, ob die Übersetzungen gemäß Übersetzungsstrategie versorgt sind.
- Abhängigkeiten in den Transporten? Es muss auf Abhängigkeiten in den Transporten geprüft werden.
- Systeminterne Dokumentation? Siehe Kapitel 6.2

WEITERE QUELLEN

1. Mathias Friedrich, Torsten Sternberg, Change Request Management mit dem SAP Solution Manager, SAP Press, 2009

8.3 SOFTWAREWARTBARKEIT

Die Wartbarkeit von Software ist ein Kriterium bei der Entwicklung von Software und zeigt an, mit welcher Energie und mit welchem Aufwand Änderungen in einem Systemzusammenhang von Applikationen durchgeführt werden können[58].

Technisch ist ein modularer Aufbau erforderlich (siehe Abschnitt 2.3 Lesbarkeit und Modularisierung). Wiederverwendbarer Quellcode muss in globalen Klassen oder Funktionsbausteinen organisiert werden. Die Identifikation von wiederverwendbaren Objekten lässt sich durch den Einsatz von Paketschnittstellen realisieren.

In Systemumgebungen mit verschiedenen Entwicklungs- und Produktivsystemen (Transportstrecken) sollte der Grundsatz gelten: Gleicher Objektname (Transaktionscode, Programm, Include, Tabelle etc.) bedeutet auch identisches Coding bzw. identische Objekteigenschaften.

Alle Entwicklungen, Änderungen und Korrekturen sind zu dokumentieren (Vgl. Kapitel 6 Dokumentation).

8.4 ANPASSUNGEN DER SAP-FUNKTIONALITÄT

Um die Funktionalität eines SAP-Systems an eigene Bedürfnisse anzupassen, gibt es verschiedene Möglichkeiten, die jeweils Vor- und Nachteile mit sich bringen:

- Erweiterung (User-Exit, Customer-Exit, BTE, BAdI, Enhancement Points und Sections, CDS Extensions)
- Implizite Erweiterung
- Modifikation
- Z-Kopie, Kopie im Kundennamensraum
  Von der Verwendung dieser Möglichkeit wird ausdrücklich abgeraten!

Zu den problemlos nutzbaren Techniken zählen User-Exits, Customer-Exits, BTEs und BAdIs. Wenn sie an geeigneter Stelle und mit geeigneter Schnittstelle vorhanden sind, sollten sie genutzt werden.

Es folgt eine kurze Beschreibung dieser Techniken.

58 *Wikipedia „Wartbarkeit"*

User-Exit

User-Exits sind Unterprogramme, die sich in Includes im SAP-Namensraum befinden, aber nur einmalig von SAP ausgeliefert werden und deshalb ohne Probleme „modifiziert" werden können.

Customer-Exit

Customer-Exits sind Funktionsbausteine, die zu- und abschaltbar sind und vom Kunden implementiert werden können, um die Standardfunktionalität zu erweitern.

Business Transaction Event (BTE)

Im FI-Umfeld stellen BTEs eine zusätzliche Möglichkeit der Erweiterung dar. BTEs sind vergleichbar mit Customer-Exits, beschränken sich jedoch weitestgehend auf das FI-Modul und stellen eine vordefinierte Schnittstelle zur Verfügung, an die der Entwickler Erweiterungen anhängen kann. Weitere Informationen finden sich in der SAP-Standarddokumentation.

Business Add-In (BAdI)

Mit BAdIs versuchte SAP, die folgenden Nachteile der bisherigen Erweiterungstechniken zu umgehen:

- Zugriff auf alle globalen Variablen (User-Exits)
- Nur einfach verwendbar (Customer-Exits)
- Keine Dynpro-Erweiterungen (BTEs)
- Keine Menü-Erweiterungen (BTEs)
- Keine Verwaltungsebene (BTEs)

Deshalb können BAdIs mehrfach verwendbar sein und alle Erweiterungsarten (Programm-, Menü- und Dynpro-Exit) zur Verfügung stellen. Sind für einen Erweiterungswunsch mehrere Erweiterungstechniken verfügbar, so wird die Verwendung von BAdIs empfohlen.

Enhancement Framework

Mit dem neuen Enhancement Framework versuchte SAP, die Nachteile der bisherigen Erweiterungstechniken zu beheben. Zum Enhancement Framework zählen:

- Explizite Enhancements (Enhancement Points und Enhancement Sections)
- „Neue" BAdIs (wobei Implementierungen der alten BAdI-Technologie automatisch migriert werden können)
- Implizite Enhancements

Enhancement Point

Diese bieten die Möglichkeit, an festgelegten Stellen Quellcode einzufügen. Dabei gilt:

- Mehrere aktive Implementierungen sind parallel möglich und
- alle aktiven Implementierungen werden ausgeführt.

Enhancement Section

Diese bietet die Möglichkeit einen definierten Abschnitt eines Programms durch eigenen Quelltext zu ersetzen. Dabei gilt:

- Mehrere aktive Implementierungen sind parallel möglich aber
- nur eine aktive Implementierung wird ausgeführt.
- Es ist nicht klar, welche aktive Implementierung ausgeführt wird.

Hinweis: Implementierungen von Enhancement Sections können durch das Einspielen von SAP Enhancement Packages oder das Aktivieren von Business Functions durch neue aktive Implementierungen oder neu aktivierte Implementierungen ersetzt werden. Es ist dann sehr schwierig, ersetzte und nicht mehr ausgeführte Implementierungen zu identifizieren. Somit kann sich durch Änderungen im SAP-Standard das Verhalten der Erweiterung ändern. Dies erhöht den notwendigen Testaufwand (TCO) massiv und führt leicht zu Disruption bei einem SAP-Release-Upgrade und EHP. Die Verwendung von Enhancement Sections ist daher sehr sorgfältig zu prüfen.

ABAP CDS Extensions

Diese erlauben die modifikationsfreie Erweiterung von CDS Views. Die Erweiterung ermöglicht das Hinzufügen von View-Feldern und Assoziationen. Die Erweiterung unterliegt Einschränkungen, die in der ABAP-Schlüsselwort-Dokumentation des jeweiligen NetWeaver Relaeses beschrieben sind.[59]

Kann mit den bereits genannten Erweiterungstechniken das gewünschte Ergebnis nicht erreicht werden, gibt es weitere Möglichkeiten, deren Verwendung aber von Fall zu Fall abgewogen werden muss.

Implizite Enhancements

Jeweils am Anfang und am Ende von Prozeduren kann Code eingefügt oder der komplette Code (bei Methoden) ausgetauscht werden.

Der Unterschied zwischen impliziten und expliziten Enhancements ist groß: Implizite Enhancements ähneln Modifikationen, mit teilweise den gleichen Nachteilen. Explizite Enhancements ähneln BAdIs.

Bei der Verwendung von impliziten Enhancements ist die SPAU_ENH abzuarbeiten, da diese Enhancements in der regulären SPAU-Transaktion nicht angezeigt werden.

Die Entscheidung, ob implizite Erweiterungsmöglichkeiten genutzt werden sollen, ist nicht nur abhängig vom Realisierungsaufwand, sondern auch von einem möglichen Folgeaufwand.

Modifikation

Modifikation des von der SAP ausgelieferten Quellcodes ist grundsätzlich problematisch, da

- Modifizierte Entwicklungsobjekte nicht der Wartung der SAP unterliegen und somit im Rahmen der SAP-Supportticket-Bearbeitung durch den Kunden zurückgebaut werden müssen.
- Modifikationen spätestens bei jedem Upgrade des SAP-Systems und unter Umständen auch beim Einbau von SAP-Hinweisen oder Service Packs, auf Kompatibilität mit dem SAP-Quellcode geprüft und ggf. angepasst werden müssen.

Grundsätzlich sollte nur dann modifiziert, wenn:

- Customizing oder Personalisierung Ihre Anforderung nicht umsetzen können und
- Keine geeigneten Erweiterungsmöglichkeiten vorgedacht sind.

Im Change-Prozess sollte der Sonderfall Modifikation aus Gründen der Nachvollziehbarkeit separat abgebildet werden.

Kopien in eigenen Namensraum/Z-Namensraum

Kopien von Quellcode aus dem SAP-Standard in den Kundennamensraum sind sehr pflegeaufwändig. Es wird empfohlen, keine Kopien durchzuführen. Es gibt kein automatisiertes Verfahren und keine manuelle Regelung, wie ein späterer Abgleich (bspw. nach Support Packages oder Hinweiseinbau) zwischen Original und Z-Kopie erfolgen kann.

59 *CDS View SAP NetWeaver 740*
CDS View SAP NetWeaver 750

BEST PRACTICE

- Priorität hat die Verwendung der von SAP vorgesehenen Erweiterungsmöglichkeiten (BAdIs, User Exits, Customer Exits, BTEs, Enhancement Points oder Sections).
- Grundsätzlich sind Workbench-Modifikationen nur unter Verwendung des Modifikationsassistenten erlaubt!
- Eine Z-Kopie ist u. U. mit sehr hohem Realisierungsaufwand bzw. Folgekosten verbunden. Darüber hinausgehen Weiterentwicklungen im Standard unberücksichtigt an der Z-Kopie vorbei, d.h. es resultiert ebenfalls ein Aufwand für die Anpassung bzw. Erstellung einer neuen Z-Kopie. Nach dem Einspielen von Enhancement Packages können verwendete Standard-Includes zu Problemen führen.
- Die Entscheidung Modifikation vs. Z-Kopie vs. implizites Enhancement ist nicht nur abhängig vom Realisierungsaufwand, sondern auch von einem möglichen Folgeaufwand.
- Keine der Möglichkeiten (Modifikation/Z-Kopie/implizites Enhancement) hat nur Vor- oder nur Nachteile. Es ist von Fall zu Fall zu prüfen, welche Technik im konkreten Szenario die wenigsten Nachteile mit sich bringt.
- Für jede Art der Erweiterung sollte eine zentrale, formalisierte, technische Dokumentation erstellt werden. Dafür sollten geeignete Templates bereitgestellt werden und eine Pflicht zu deren Verwendung bestehen.

8.5 TESTBARKEIT VON ANWENDUNGEN

8.5.1 TESTPROZESSGRUNDLAGEN FÜR DIE ERSTELLUNG VON SOFTWARE-PRODUKTEN

Das Testen von Anwendungen ist ein Werkzeug der Softwarequalitätsmessung[60] und -verbesserung. Es dient u.a. der Überprüfung von funktionalen und nicht funktionalen Produktanforderungen, der Feststellung von Anomalien im Verbrauch von Systemressourcen und der Sicherstellung einer korrekten Orchestrierung im Rahmen von zeitlich voneinander abhängigen Prozessschritten. Dabei gilt: Je früher ein Mangel identifiziert und beseitigt werden kann, desto geringer sind die Kosten.

Fehlerkorrekturkosten

Gut veranschaulicht wird die Korrelation zwischen Fehlerkorrektur und deren Folgekosten in der mittlerweile in die Jahre gekommenen, aber vom Grundsatz weiterhin gültigen Studie von Barry W. Boehm zum Thema „Relativer Aufwand der Fehlerkorrekturen des Entwicklungszyklus" aus dem Jahr 1981. Nach Boehm ist ein in der Wartungs- bzw. Betriebsphase gefundener Fehler bis zu ein 100faches teurer, als wenn er in der Analysephase korrigiert worden wäre. In Anlehnung an Balzert[61] soll die nachfolgende Tabelle als exemplarische Orientierungshilfe dienen und verdeutlichen, in welcher Phase des Entwicklungszyklus mit welcher Fehlerhäufigkeit zu rechnen ist und wie wahrscheinlich der Fehler entdeckt wird.

Phase	Analyse	Design	Implementierung	Systemtest	Abnahmetest	Wartung / Betrieb
Relativer Aufwand	0,2	0,5	1	2	5	20
Fiktive Korrekturkosten	1€	2,5€	5€	10€	25€	100€
Eingebrachte Fehler	55%	30%	15%			
Gefundene Fehler	5%	10%	40%	45%		

WEITERE QUELLEN

1. SAP-Schulungen BC425 und BC427

60 *Vgl. https://de.wikipedia.org/wiki/Softwarequalit%C3%A4t#QS-Schwerpunkt_Softwaretest*
61 *Vgl. Balzert, Softwaremanagement, Spektrum 2008 S. 484ff*

Maßnahmen zur Fehlervermeidung

Die Entdeckung von und der Umgang mit Mängeln lässt sich durch den Einsatz verschiedener Hilfsmittel unterstützen. Angefangen bei der Definition eines Prozessmodells (z.B. I2M, ITIL, V-Modell) für die Festlegung der Testebenen und Quality Gates, der Ausrichtung an einem Qualitätsstandard (z.B. SQuaRE, ISO/IEC 9126) zur Definition von Prüfkriterien, der Verwendung verschiedener Produktentwicklungsmethoden (z.B. SAP Agile SE, KANBAN, Scrum) für den Umgang mit Fertigungs- und Abnahmekriterien, bis hin zur Einführung eines effektiven Risikomanagements und der Festlegung von Schutzbedarfsklassen für Anwendungskategorien, ist das Spektrum für den Umgang mit Tests breit gestreut und von verschiedenen Faktoren abhängig. Hierzu gehören u.a. die Branchenzugehörigkeit des Unternehmens, gesetzliche Rahmenbedingungen und die potenzielle Schadensauswirkung der Anwendung auf den Geschäftsprozess.

Teststufen

Bei der Fertigung von Softwareprodukten lassen sich grundsätzlich drei verschiedene Testverfahren voneinander unterscheiden:

- White Box
- Grey Box
- Black Box

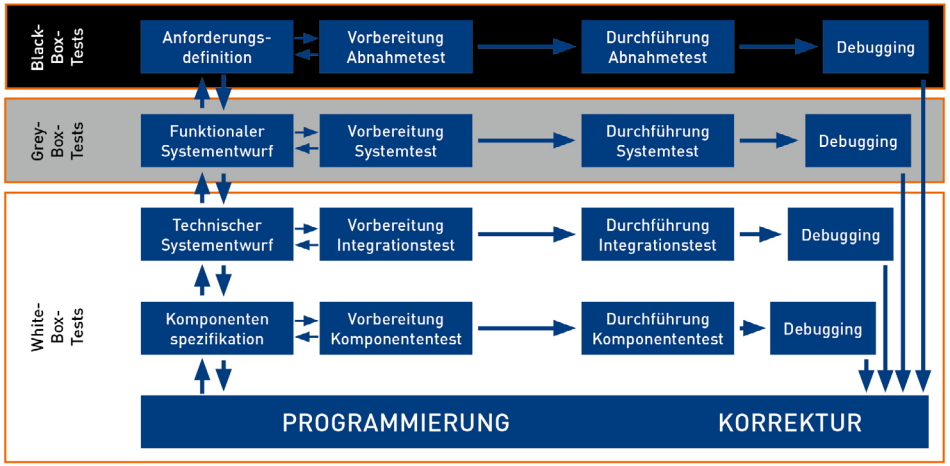


Abbildung 4: W-Modell mit Teststufen [61]

Typische Vertreter von White-Box-Tests sind Komponenten- und Integrationstests. Die Gemeinsamkeit dieser Tests liegt darin, dass die Struktur, Implementierung oder Reihenfolge der getesteten Softwareartefakte bekannt ist. Die Tests werden in der Regel im Rahmen der Entwicklung direkt durch das Entwicklerteam auf dem Entwicklungssystem erstellt, durchgeführt und weiterentwickelt.

Das Grey-Box-Testverfahren ist dem Akzeptanztest vorangestellt und wird nach Fertigstellung des Produkts bzw. eines Produktinkrements in Form von Systemtests durchgeführt. Meistens findet die Erstellung dieser Tests auf einer Produktivsystem-nahen Umgebung durch ein separates Test-Team statt. Die organisatorische Aufteilung zwischen Grey- und White-Box-Tests hat den Vorteil, dass sich das Entwicklungsteam auf seine Kernaufgaben konzentrieren kann. Das Test-Team validiert die Anwendung aus einer anderen Perspektive und versucht, mögliche Schwachstellen oder Mängel und deren Ursachen zu identifizieren.

Das Black-Box-Testverfahren betrachtet das fertiggestellte Produkt ohne Kenntnis der internen Softwarestruktur. Die Testfälle leiten sich aus der Spezifikation und der darin geforderten Funktionalität ab und dienen als Akzeptanztest zur Abnahme des erstellten Produkts. Die Durchführung der Black-Box-Tests findet auf einer möglichst produktivsystemnahen Umgebung durch den Auftraggeber statt.

62 *In Anlehnung an http://www.informatik.hs-bremen.de/spillner/ForschungSpillnerWmo.pdf*

BEST PRACTICE

- Achten Sie darauf, dass die definierte Teststrategie zu Ihrer Aufbau- und Ablauforganisation passt, die Aufgabenverantwortung klar definiert ist, und dass Sie die Produktivität des Entwicklungsteams nicht durch unnötige organisatorische Schnittstellen hemmen.
- Legen Sie die Testintensität und den Testumfang in Abhängigkeit der im Fehlerfall zu erwartenden Schadenshöhe und dessen Eintrittswahrscheinlichkeit fest. Achten Sie darauf, nur die wirklich notwendigen Testfälle durchzuführen und vermeiden Sie Redundanzen.
- Involvieren Sie den Auftraggeber frühzeitig in die Definition von Testfällen. Bereits während der Anforderungsdefinition sollte ein grobes Testszenario zur Validierung der Anforderung zusammen mit dem Auftraggeber festgehalten werden. Als positiver Nebeneffekt ergibt sich häufig eine bessere Produktspezifikation. Die zunächst grobe Beschreibung der Testfälle aus der Anforderungsdefinition lässt sich in den weiteren Entwicklungsphasen verfeinern.

8.5.2 TESTAUTOMATISIERUNG

Eine Testautomatisierung eignet sich für alle Bereiche, in denen regelmäßig eine Folge von manuellen Arbeitsschritten durchgeführt werden muss, um das korrekte Verhalten einer Funktionalität zu überprüfen. Die Entscheidung, welche Aktivitäten zu automatisieren sind, sollte von der Risikokategorie, dem Komplexitätsgrad, der Ausführungshäufigkeit und der Relation von manuellen Kosten gegenüber den Automatisierungskosten abhängig sein. Automatisierte, wiederholbare Tests können sehr effizient sein.

Es gibt zwei Ansätze:

- Auf Quellcode-Ebene: Unit-Tests mit ABAP-Unit
- Auf übergeordneter Ebene: automatisierte Funktionstests mit eCATT oder Drittanbieterwerkzeugen

Vorteile von automatischen Tests:

- Die Funktionsfähigkeit der Software kann nach Änderungen praktisch ohne Zeitaufwand geprüft werden.
- Unit-Tests helfen schon in der Entwicklungsphase bei der Fehlersuche und unterstützen den Entwickler bei einem guten Design (Modularisierung, Einfachheit).

Nachteile von automatischen Tests:

- Nach Änderungen müssen teilweise bestehende Tests inklusive Testdaten angepasst werden.
  - Das sollte bei Unit-Tests bei guter Modularisierung eher selten nötig sein.
  - Bei automatisierten Funktionstests hängt es davon ab, ob sich die Änderungen „in der Tiefe“ bis auf die getestete Ebene auswirken. Problematisch sind meist Änderungen der Oberfläche, da die Tests in der Regel hierüber angesteuert werden und dann die Ansteuerung angepasst werden muss.
- Initialer Mehraufwand
- Eventuell aufwändige Umsetzung, z.B. bei Unit-Tests, wenn Datenbank-Aufrufe nicht gekapselt sind.

Diese Kontexte sprechen besonders für den Einsatz von Unit-Tests:

- Häufige Anpassungen der Software sind zu erwarten
- Softwareteile, die vermutlich wiederverwendet werden
- Komplexe Funktionalität
- SAP-NetWeaver-Softwarestand

- Hilfsmittel Test-Doubles ab SAP NetWeaver 7.40 SP9 verfügbar
  - Alternativ gibt es das Open-Source-Werkzeug MockA63, das für Releases ab SAP NetWeaver 7.01 verfügbar ist
- Hilfsmittel Test-Seams ab SAP NetWeaver 7.50 verfügbar

Diese Kontexte sprechen eher gegen den Einsatz von Unit-Tests:

- Klassen, deren Zweck der Datenbankzugriff ist („Datenbankschicht", „Model" im Model-View-Controller-Muster), und die keine komplexe Logik beinhalten.
- Klassen, deren Zweck die Ansteuerung und Kommunikation mit der Oberfläche ist („View" im Model-View-Controller Muster), und die keine komplexe Logik beinhalten.
- Existierende Software mit schlechter Modularisierung (vor allem, wenn Test-Seams noch nicht verfügbar sind, siehe oben)

BEST PRACTICE

- Erwägen Sie den Einsatz von automatisierten Tests.
- Bauen Sie Kompetenz in diesem Bereich auf.

Im ABAP-Umfeld ist in vielen Fällen die Integration der Datenbank oder der Oberfläche ein Hindernis, wenn es um die Erstellung von wiederholbaren, automatisierten Tests geht. Für Unit-Tests liegt die Lösung in der ohnehin sinnvollen Modularisierung.

BEST PRACTICE

Trennen Sie in Ihren Anwendungen die direkte Interaktion mit der Datenbank, der Benutzeroberfläche und entfernten Systemen von dem eigentlichen Anwendungskern.

63 siehe https://github.com/uweku/mockA

Wenn dieser Anwendungskern z.B. nicht mehr direkt mit der Datenbank kommuniziert, bietet sich die Möglichkeit, für die Durchführung von Unit-Tests die Datenkonstellationen zu simulieren. Hierfür sind insbesondere die oben erwähnten Testdoubles sinnvoll.

Auch für automatisierte Funktionstests kann die Trennung der Schichten manchmal hilfreich sein, der Fokus bei eCATT liegt allerdings auf der Ansteuerung über SAP GUI.

BEST PRACTICE

- Nutzen Sie für die Entscheidung, welche Prozesse für automatisierte Regressionstests in Frage kommen, die im SAP System zur Verfügung stehenden Aufrufstatistiken (Transaktionsstatistik via ST03N / Usage Procedure Logging).
- Starten Sie bei der Einführung von Unit-Tests bzw. automatisierten Funktionstests mit einem kleinen engagierten Team. Dessen Erfahrungen und Erfolge lassen sich als Erfolgsgeschichten bei der weiteren Einführung des Themas in Ihrer Organisationseinheit nutzen. Hierbei ist es wichtig, dass das Team von seiner Tätigkeit überzeugt ist und anhand von Beispielen den Nutzen dieser Tätigkeit belegen kann.

WEITERE QUELLEN

1. http://www.testbarkeit.de
2. http://de.wikipedia.org/wiki/Testbarkeit
3. http://www.testbarkeit.de/Publikationen/TAE05_Artikel_jungmayr.pdf
4. ABAP-Unit-Tests
5. ABAP Test Double Framework
6. ABAP Test Double Framework vs. mockA
7. Test-Seams and Injections

9 ECLIPSE-ENTWICKLUNGSUMGEBUNG

In der Vergangenheit fanden Entwicklungen im SAP-Umfeld ausschließlich in den ABAP-Workbench-Werkzeugen statt (SE80). Im Rahmen der neuen Technologien sind in den letzten Jahren weitere Entwicklungswerkzeuge wie zum Beispiel die SAP Web IDE (SAP-UI5-Entwicklung im Browser) oder das SAP HANA Studio (Eclipse-basierte Umgebung für die Administration und native Entwicklung auf/mit HANA)[64] entstanden. Für die ABAP-Entwicklung steht mit den ABAP Development Tools (ADT) für Eclipse seit AS ABAP 7.31 SP4 der Nachfolger zur ABAP-Workbench zur Verfügung. ADT basiert auf der Entwicklungsumgebung Eclipse erweitert um entsprechende Plug-ins für die Entwicklung in ABAP.

9.1 VORAUSSETZUNGEN UND INSTALLATION

Wie bereits oben erwähnt ist mindestens Release AS ABAP 7.31 SP4 und Kernel 7.21 notwendig, um die ADT für Eclipse einsetzen zu können. Der in ADT verfügbare Funktionsumfang ist jedoch nicht nur von der verwendeten ADT-Version abhängig sondern auch von der verwendeten AS-ABAP-Version. Eine ständig aktualisierte

Übersicht über die mit den verschiedenen AS-ABAP-Versionen verfügbaren Funktionen ist unter [1] im SCN verfügbar.

Im Gegensatz zur ABAP-Workbench ist für ADT eine zusätzliche Installation auf dem Arbeitsplatz des Entwicklers notwendig. Eine aktuelle Installationsanleitung für ADT ist unter [2] verfügbar.

Alternativ zu dieser Anleitung ist je nach Firmengröße eine Alternative, ein Grundpaket (Eclipse IDE + ADT) über den Standardverteilungsmechanismus verfügbar zu machen und lokal nur das Update auf die aktuelle Version zu machen (über SAP-Updatesite oder hausintern).

9.2 NOTWENDIGKEIT

ADT für Eclipse ist der von SAP designierte Nachfolger der ABAP-Workbench. Neue Objekttypen wie z.B. ABAP Core Data Services (CDS) können mit der ABAP-Workbench (SE80 oder anderen SAP GUI-Transaktionen) nicht mehr bearbeitet werden, da sich diese nur noch in der Wartung befindet. Zusätzlich wächst der Funktionsumfang von ADT mit jedem Release. Für verbliebene, noch nicht in ADT implementierte Funktionen (z.B. Anlegen von Erweiterungen), werden die SAP GUI-Transaktionen nahtlos in ADT eingebettet aufgerufen. Mittelfristig wird sich eine Notwendigkeit zum Umstieg auf ADT ergeben.

9.3 VORTEILE

Die Vor- und Nachteile, die sich aus der Verwendung von ADT ergeben, hängen maßgeblich von der individuellen Arbeitsweise als auch vom konkreten Projektumfeld ab. In den folgenden Abschnitten haben wir die aus unserer Sicht wichtigsten Vor- und Nachteile bei der Arbeit mit ADT aufgeführt. Eine gute Übersicht über die mit ADT insgesamt verfügbaren Funktionen bietet [1] sowie das FAQ-Dokument zu ADT [7].

ADT-Arbeitsplatz

Bei der Arbeit mit ADT ist man beim Editieren von ABAP-Quellcode nicht mehr an SAP GUI-Modi gebunden. Das heißt, es können viele Sourcen gleichzeitig geöffnet sein und diese bleiben auch nach Ab-/Anmeldung in exakt gleicher Form erhalten. ADT-Links ermöglichen es zeilengenau Links zu Stellen im ABAP-Quellcode zu erstellen und zu teilen (siehe [3]). Weiterhin ermöglicht es das Eclipse-Tool Mylyn, den ADT-Arbeitsplatz und die geöffneten ABAP-Quellcodes anhand von Aufgaben (z.B. aus einem Ticketsystem) zu organisieren (siehe [4]).

Konfigurierbarkeit des ADT-Arbeitsplatzes

Eclipse und somit auch ADT ermöglicht es, den Arbeitsplatz flexibel zu konfigurieren. So ist es durch Drag and Drop einfach möglich, Views zu vergrößern oder die Position bestimmter Views zu ändern. Weiterhin ermöglicht das Verknüpfen von Views, dass z.B. im View der ABAP-Dokumentation immer die Dokumentation zum aktuell ausgewählten Schlüsselwort angezeigt wird.

Editorfunktionen

Der Editor selbst bietet eine Reihe zusätzlicher Funktionen, die die Entwicklereffizienz erhöhen:

- Laufende Syntaxprüfung (nicht erst bei manuellem Aufruf wie in der SE80 per Knopfdruck im Editor)
- Quickfix-Funktionen: das System schlägt Möglichkeiten vor, wie der aktuelle Syntaxfehler behoben werden kann (z.B. halbautomatisches Anlegen einer neuen Methode mit den verwendeten Parametern)

64 *Dies stellt keine Empfehlung für das HANA Studio zur Entwicklung nativer HANA-Anwendungen dar.*

- Fundstellen zu Syntax- oder ATC-Prüfungen werden im Editorfenster gekennzeichnet und die zugehörigen Meldungen können per Mouse-Over angezeigt werden
- Code Element Information: Pop-ups mit Information zu im Code verwendeten Entwicklungsobjekten
- ABAP-Doc als integrierte Dokumentationsmöglichkeit

Refactoring-Funktionen

ADT bietet eine ganze Reihe von automatisieren Refactoring-Funktionen. So ist es möglich, eine Variable und alle ihre Verwendungen umzubenennen oder nicht verwendete Variablendeklarationen automatisch zu entfernen. Komplexere Refactoring-Funktionen erlauben die automatisierte Extraktion einer Methode oder eines Attributs.

Verbesserte Laufzeitanalyse

Die in ADT integrierte Laufzeitanalyse (siehe [5]) ermöglicht es, ABAP-Traces graphisch zu visualisieren. Diese Visualisierung erlaubt es sehr einfach, performancekritischen Quellcode zu identifizieren und zu optimieren.

Verbesserte Versionshistorie

Die in ADT integrierte Versionshistorie kann remote über Systeme hinweg verwendet werden und hat auch eine lokale (arbeitsplatzbezogene) Historie und ermöglicht so eine maximale Flexibilität.

SQL-Tools

In ADT haben sich auch die klassischen Werkzeuge des Entwicklers im SQL-Bereich neu erfunden. Zum Beispiel steht eine SQL-Konsole als auch eine integrierte Data-Preview-Funktion zu Verfügung.

Erweiterbarkeit

Da es sich bei Eclipse um eine offene Plattform handelt, ist diese sehr flexibel erweiterbar. So stehen zum einen SAP-Tools zur Verfügung. Zum anderen ist ADT auch mit existierenden Tools aus dem Eclipse-Ökosystem erweiterbar. Zusätzlich ist für ADT ein SDK (Software Development Toolkit) verfügbar (siehe [6]), das es ermöglicht, ADT um eigene Funktionen zu erweitern. Zusätzlich dazu wird auch eine SAP-Codejam zu diesem Thema angeboten.

9.4 ZU BEACHTENDE PUNKTE

Neben den oben genannten Vorteilen existieren bei der Arbeit mit ADT für Eclipse jedoch auch einige Nachteile.

Einstiegs- und Umstiegshürde

Der Einstieg in jedes neue Werkzeug ist mit einer initialen Ein- bzw. Umgewöhnungsphase verbunden. Insbesondere erfordert der Umstieg von der ABAP-Workbench auf ADT auch die Gewöhnung an ein neues Entwicklungsparadigma. Während in der ABAP-Workbench formularbasierte Editoren überwiegen (z.B. für Klassen und Methoden), verwendet ADT überwiegend Quellcode-basierte Editoren. So ist zusätzlich zu dem neuen Werkzeug auch die Gewöhnung an eine neue Form der Bearbeitung des Quellcodes notwendig. Dieser Umstand stellt aus unserer Erfahrung eine nicht zu unterschätzende Einstiegshürde dar.

Schlussendlich wird man jedoch nach dem Umstieg schnell feststellen, dass ein Sourcecode-basiertes Arbeiten deutlich effizienter ist.

Fehlende Spezialtransaktionen

ADT unterstützt nicht alle Spezialtransaktionen, die über die SAP GUI verfügbar sind. Obwohl der Funktionsumfang von ADT stetig erweitert wird (z.B. um Modellierungsfunktionen für BOPF), werden auch in Zukunft nie alle Spezialtransaktionen in ADT umgesetzt werden. Diese können zwar über die SAP GUI Integration innerhalb von ADT aufgerufen werden. Jedoch ist die Verwendung über die SAP GUI-Integration nicht immer ideal. So ist zum Beispiel das Arbeiten mit den SAP-CRM-WebUI-Tools in ADT nicht immer konsistent. Bestimmte Objekte in den SAP-CRM-WebUI-Tools werden zur Bearbeitung in der ABAP-Workbench geöffnet, während andere Objekte in den ADT-Editoren geöffnet werden.

9.5 PROBLEME UND HILFESTELLUNGEN FÜR DEN UMSTIEG

Die ersten Tage der Arbeit mit ADT werden aus der Erfahrung heraus etwas holprig. Insbesondere die freie Konfigurierbarkeit der Entwicklungsumgebung und das Quellcode-basierte Entwicklungsparadigma stellen für erfahrene SAP-Entwickler ein Hindernis dar. Sie werden im ersten Moment nicht als nennenswerter Vorteil wahrgenommen. Vielmehr werden sie als unnötige Veränderung ohne Mehrwert erlebt. Dies ändert sich aber nach Überwindung der Einstiegshürde und die Entwicklungseffizienz nimmt zu.

Trotz der Möglichkeit, alle Tastenkürzel zu ändern, empfehlen wir, sich an die Standardkürzel zu gewöhnen. So wird für unterschiedliche Arbeitsplätze und Remote-Arbeitsumgebungen nicht unnötig „Einstellungsarbeit" generiert.

Auch ist es ein Problem, wenn für mehrere Systeme mit unterschiedlichen Release-Ständen in ADT entwickelt werden soll. In diesem Fall kann es vorkommen, dass bestimmte Funktionen nicht in allen Systemen verfügbar sind (vergleiche [1]).

Falls man eine größere Anzahl an Entwicklern im Unternehmen hat, hat es sich bewährt, die Einführung von ADT mit einer Schulung beziehungsweise mit einer Vorstellung des Werkzeugs zu beginnen (Begrifflichkeiten und Grundeinstellungen).

Alternativ ist in ADT der sogenannte „Feature Explorer" integriert. Der Feature Explorer ist eine Art Lerntutorial für das Selbststudium, der die grundlegende Arbeitsweise mit ADT erklärt. Hier wird beispielsweise gezeigt, wie man ein Entwicklungssystem in ADT oder mit Hilfe der Refactoring-Werkzeuge Teile einer Methode extrahieren kann.

Eine weitere Möglichkeit für den Einstieg ist, an einem SAP-Codejam für ABAP in Eclipse teilzunehmen.[65]

9.6 FAZIT

Die Vorteile von ADT überwiegen spätestens seit Release 7.40 die Nachteile. Der Umstellungsaufwand ist überschaubar (allerdings je nach Lernmotivation und -fähigkeit verschieden). Insbesondere wird die Umstellung jüngeren Mitarbeiterinnen bzw. Mitarbeitern, die in Entwicklungsumgebungen außerhalb der ABAP-Workbench gearbeitet haben, leichter fallen.

BEST PRACTICE
Wir empfehlen aus unserer Erfahrung die Verwendung von ADT ab AS ABAP Release 7.31 SP6.

9.7 WEITERE QUELLEN

[1] ABAP in Eclipse Feature Matrix

[2] ABAP in Eclipse Installationsanleitung

[3] How ADT links change the way you work

[4] Use mylyn tasks to organize your ABAP in Eclipse workspace

[5] ABAP Profiling in Eclipse

[6] First version: SDK for ABAP development tools

[7] FAQs zu ADT

[8] BOPF Modelling in ADT

[9] ADT Feature Explorer

[10] Get Started with the ABAP-Development-Tools for SAP NetWeaver

65 *Vgl. http://scn.sap.com/community/events/codejam*

10 USER INTERFACE (UI)

10.1 UI-TECHNOLOGIEN IN DER PRAXIS

UI-Technologien spielen eine immer wichtigere Rolle in Entwicklungsprojekten. Aus diesem Grund möchten wir auch in diesem Kapitel darauf eingehen.

Der Fokus des UI-Kapitels liegt auf den „neuen" SAP-UI-Technologien, da wir in den SAP-Produkten einen klaren Trend zu SAPUI5 als UI-Framework und SAP Fiori als zentralen Einstieg in die SAP-Welt sehen.

Dennoch bewerten wir z.B. Web Dynpro oder klassisches Dynpro als wichtige UI-Komponenten in der SAP-Welt. Für diese schon lange etablierten Technologien gibt es bei vielen SAP-Kunden bereits Empfehlungen und auch ein breites Grundwissen.

Um sich einen ersten Überblick über die verschiedenen UI-Technologien zu verschaffen, empfiehlt sich der EA Explorer[66] der SAP. Dort findet man einen guten Einstieg in verschiedene Aspekte von User Interfaces. In diesem Leitfaden sind die wichtigsten UI-Technologien in der nachfolgenden Tabelle aufgelistet. Die Tabelle beschränkt sich auf einen Überblick sowie einer kurzen Bewertung zum jeweiligen Gebrauch bei (Neu-) Entwicklungen.

UI-Technologie	SAP-Roadmap	Kommentar	Empfehlung für Neuentwicklungen
Dynpro (klassisch)	nur noch Support	SAP rät von Neuentwicklungen ab. Geringer Entwicklungsaufwand, vor allem bei einfachen Reports mit generiertem Selektionsbild. Bei Power-Usern beliebt.	Für kleinere Entwicklungen in vielen Fällen weiterhin sinnvoll
Business Server Pages (BSP)	nur noch Support	Durch Web Dynpro abgelöst	Nicht sinnvoll
WebClient UIF	nur noch Support	Im CRM auf Basis der BSP-Technologie entwickelt und im Einsatz	Für klassische CRM-Apps weiterhin relevant. SAP Hybris C4C setzt hier auf SAPUI5/SAP Fiori
Web Dynpro Java	nur noch Support	Sollte nicht mehr verwendet werden	Nicht sinnvoll
Web Dynpro ABAP inkl. Floorplan Manager	Kleinere Erweiterungen	In Kombination mit Floorplan Manager weniger aufwändig als Standalone.	Sinnvoll für größere Neuentwicklungen. Auch SAPUI5 in Erwägung ziehen.
SAP Screen Personas	Kleinere Erweiterungen	Konfiguration und Scripting (JavaScript), um existierende Anwendungen auf Basis klassischer Dynpros attraktiver und besser bedienbar zu machen	Für UI-Überarbeitung existierender Dynpro-Programme sinnvoll.
SAPUI5	Strategisch		Sinnvoll. Siehe Vor-/Nachteile im Folgenden

Derzeit ist SAPUI5 die favorisierte UI-Technologie moderner SAP-Anwendungen. Nichtsdestotrotz ist ein Abwägen über die zu verwendende UI-Technologie zu Beginn von Entwicklungsprojekten ratsam. Folgende Gegenüberstellung der Vor- und Nachteile soll bei der Entscheidung bzgl. des Einsatzes von SAPUI5 helfen.

66 SAP EA Explorer

Vorteile SAPUI5	Nachteile SAPUI5
• Umfassende Sammlung von Standard-GUI-Elementen, die die Implementierung stark vereinfachen • Modernstes Aussehen • Theoretisch ist alles möglich, was HTML5 in Verbindung mit JavaScript erlaubt • Nutzung auf Tablets und Smartphones • Kein Client zu installieren • Responsive UI (Automatische Anpassung an das jeweilige Endgerät) • Nutzung der Endgerätfähigkeiten wie z.B. Kameras • Native SAP-Fiori-Launchpad-Integration • Relativ neue Technologie, daher optimale Integration in aktuelle Web-Browser	• JavaScript erforderlich (ABAP nur im Backend). Daher eventuell Skill-Aufbau notwendig. • SAP Gateway erforderlich (bei der empfohlenen separaten Installation zusätzliche Kosten) • In Spezialfällen u. U. fehlende Features und schlechtere Performance gegenüber SAP GUI / ALV • Relativ neue Technologie, daher Probleme beim Einsatz der Tools und als Endprodukt möglich • Komplexe Apps erfordern mehr Aufwand (Stateless Apps)

SAP Fiori bezeichnet die neue User Experience (UX) aktueller Lösungen der SAP, die auf Basis moderner Design-Prinzipien entstanden sind. Das Thema SAP Fiori werden wir im Kapitel SAPUI5 nur ansatzweise behandeln, da die darunterliegenden Technologien in der On-Premise-Welt aktuell SAPUI5 und SAP Gateway sind. Auf das Thema HANA Cloud Platform (HCP) wird in dem UI-Teil nicht explizit eingegangen, da sich der SAPUI5 Teil nicht groß von der On-Premise-Welt unterscheidet und sich die Best Practices von der On-Premise-Welt fast eins zu eins übernehmen lassen.

Auch das Thema Design Thinking bekommt in den letzten Jahren einen immer größeren Stellenwert im Bereich UX und sollte für einen voll umfänglichen End-to-End-Prozess ebenfalls betrachtet werden.

Eng mit dem Design Thinking ist auch das Thema Mock-up verknüpft. Hier gibt es schon etablierte Tools, in denen es fertige SAP Fiori Design Stencils67 gibt, die das SAP Fiori UX Pattern abbilden. Ein neues Produkt von SAP in diesem Bereich ist Splash / BUILD, aber Stand Sommer 2016 ist dieses Produkt noch in der Beta-Phase und es gibt erst wenige Erfahrungen dazu.

67 SAP Fiori Design Stencils

10.2 SAPUI5

Im Rahmen der SAP-User-Experience-Strategie[68] wird mit SAPUI5 ein modernes Toolkit für HTML5 basierte Entwicklungen zur Verfügung gestellt. Mit SAPUI5 entwickelte Anwendungen zeichnen sich dadurch aus, dass sie eine responsive Web-Oberfläche sowohl auf dem Desktop-Browser als auch auf mobilen Endgeräten zur Verfügung stellen.

Unter dem Namen OpenUI5[69] wird SAPUI5 auch unter einer Open-Source-Lizenz (Apache 2.0) vertrieben. Hierbei sind allerdings einige Komponenten wie Diagramme und Smart Controls nicht in der Distribution enthalten.

Aktuell realisierte Anwendungen sollen dabei nach den SAP Fiori Design Guidelines[70] realisiert werden, um eine optimale Multi Device User Experience zu gewährleisten und sich in das „Look and Feel" von SAP-Anwendungen zu integrieren. In SAPUI5 sind aus den Anfängen der Bibliothek noch rein Desktop orientierte Komponenten enthalten (sap.ui.commons), die aber nicht mehr verwendet werden sollten (deprecated).

10.2.1 ANFORDERUNGEN

Für den Einsatz von SAPUI5 wird im Frontend ein unterstütztes Endgerät benötigt, das von SAP innerhalb der SAP NetWeaver 7.5 Browser Support PAM[71] unterstützt wird.

Abbildung 5: SAP NetWeaver 7.5 Browser Support PAM

68 SAP-User-Experience-Strategie (UX)
69 OpenUI5-Homepage
70 SAP Fiori Design Guidelines
71 SAP NetWeaver 7.5 Browser Support PAM

SAPUI5 ist von hoher strategischer Bedeutung und ist Bestandteil innerhalb der Auslieferung von folgenden SAP-Systemen:

- SAP NetWeaver AS ABAP (z.B. SAP Business Suite oder SAP Gateway)
  Ab NetWeaver Version 7.40 sind sowohl UI- als auch SAP-Gateway-Komponenten Bestandteile des SAP NetWeaver. In vorherigen Versionen können sie teilweise als Add-on nachgerüstet werden.

- SAP HANA
  SAP HANA liefert eine Version des SAPUI5 SDKs passend zum HANA SP aus (XS Classic). Ab SPS 11 und XS Advanced muss derzeit noch das SDK via CDN eingebunden werden.

- SAP HANA Cloud Platform (HCP)
  Die SAP HCP bietet die Möglichkeit SAPUI5 und Fiori-Apps zu entwickeln und stellt mit der SAP Web IDE eine Entwicklungsumgebung für SAPUI5 zur Verfügung.

- SAP Enterprise Portal (EP)
  Die SAP Web IDE ermöglicht das direkte Deployment einer SAPUI5-Anwendung in das Portal als iView.

Alle Systeme stellen außerdem ein SAP Fiori Launchpad (FLP) zur Verfügung, allerdings in unterschiedlichen Ausbaustufen.

BEST PRACTICE

- Wir empfehlen für die SAPUI5-Entwicklung (Stand 2016) den Google Chrome Browser zu verwenden. SAP liefert mit dem UI5-Inspector[72] (Google Chrome Extension) ein Tool aus, auf das man nicht mehr verzichten möchte. In der SAP Web IDE unterstützt der Layout-Editor Stand 2016 nur Google Chrome. Der Endanwender kann die entwickelten Anwendungen mit allen gängigen Browsern und Devices verwenden, sofern diese laut PAM unterstützt werden.
- Jedes der oben genannten Systeme unterstützt neben dem SAPUI5 SDK auch das SAP Fiori Launchpad (FLP). Über FLP können rollenbasiert SAPUI5-Anwendungen gestartet werden, wenn diese nach den SAP Fiori Design Guidelines konzipiert wurden und eine eigenständige Komponente implementieren.

10.2.2 ENTWICKLUNG

SAP folgt und unterstützt bei der modernen UX-Entwicklung den Ansatz des Design Thinkings[73], der sich bei der Entwicklung neuer Prozesse und Benutzeroberflächen nutzen lässt. Design Thinking umfasst die drei Phasen Discover, Design und Deliver.

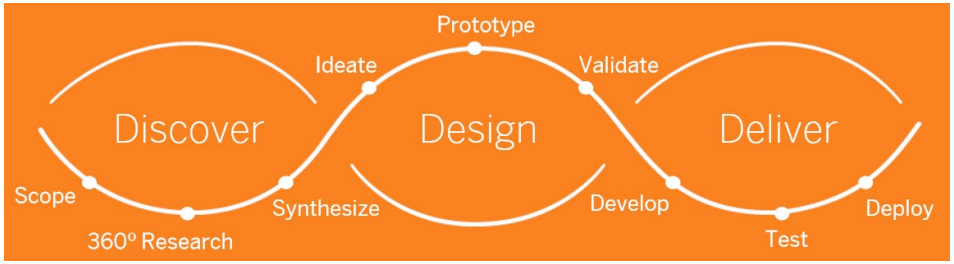


Abbildung 6: Phasen des Design Thinkings

72 UI5-Inspector (Google Chrome Extension)
73 Wikipedia Definition Design Thinking

10.2.2.1 Discover

Im Rahmen der Discovery-Phase wird versucht, das aktuelle Kundenproblem zu verstehen und kreativ im Team einen Prozess zur Lösung herauszuarbeiten. Die Lösung beschreibt dabei einen konkreten Arbeitsablauf für eine definierte Zielgruppe.

10.2.2.2 Design

Innerhalb der Design-Phase wird mit der Zielgruppe ein Prototyp skizziert, der den bestmöglichen Arbeitsablauf zur Erreichung der Anforderung aus der Analyse abbilden soll.

Je nach Know-how und Zielgruppe können folgende Tools verwendet werden:

- Papier und Stift
  Immer noch die einfachste Variante, Ideen festzuhalten.

- Whiteboards / Whitewalls
  Die exklusive Papier und Stift Variante.

- SAP Splash and BUILD
  Die SAP stellt mit Splash und BUILD[74] ein Browser-basiertes Tool zur Verfügung, mit dem via Drag & Drop Prototypen erstellt werden können, die sich an SAPUI5/ SAP Fiori Vorlagen orientieren und auch dessen Widgets enthalten. Ein BUILD-Prototyp kann zur weiteren Entwicklung in die SAP Web IDE importiert werden[75].

- SAP Web IDE
  Die SAP Web IDE[76] ist eigentlich ein Werkzeug für die Entwicklung und das Deployment von SAPUI5-Anwendungen. Stand 2016 verfügt sie aber mit dem Layout-Editor auch über die Möglichkeit, UIs via Drag & Drop zu erstellen. Zusätzlich können hier Mock-up-Daten hinterlegt werden, sodass ein funktional erweiterter Prototyp inkl. Testdaten online zur Verfügung gestellt werden kann, ohne dass vorab Daten-Services entwickelt werden müssen.

BEST PRACTICE

- Die Anwenderzielgruppe sollte möglichst frühzeitig in den Prozess des Prototyping involviert werden, da das Design einer Anwendung genau auf die Anforderung und Arbeitsweise des Anwenders ausgerichtet wird. Auch die Akzeptanz wird so gesteigert, da die Zielgruppe in die Design-Phase integriert wurde und das Design miterarbeitet hat.
- In gemeinsamen Workshops kommt man immer noch am schnellsten mit dem Whiteboard zu allgemeinen Ergebnissen. Der SAPUI5-Explorer[77] sowie die SAP Fiori Demo Cloud[78] sind dabei hilfreiche Werkzeuge, um neuen Anwendern das Thema SAPUI5 näherzubringen.
- Stand 2016 haben Splash und BUILD noch einige Macken, die teilweise eine produktive Verwendung inkl. der Code-Übernahme verhindern.
- Der openSAP-Kurs „Build Your Own SAP Fiori App in the Cloud – 2016 Edition[79]“ bietet einen umfassenden Einstieg und Überblick für das Thema SAP Fiori UX.

74 *SAP Splash and BUILD (Design Great UX in the Cloud)*
75 *Kickstart your Fiori Web IDE project with Splash and BUILD*
76 *SCN – SAP Web IDE – Quickstart*

10.2.2.3 Deliver

Wurde im Rahmen des Prototypen-Designs ein entsprechender Prototyp verabschiedet, kann dieser nun in eine SAPUI5-Anwendung überführt werden.

Werkzeuge

Die SAP unterstützt die SAPUI5-Entwicklung mit den folgenden Werkzeugen:

- SAP Web IDE (in der Cloud)
  Bei der SAP Web IDE handelt es sich um das favorisierte Tool der SAP, das in der SAP HCP angeboten wird. Neben einer gratis Testversion (für nicht produktive Zwecke empfohlen) gibt es Stand 2016 auch eine schmale Einsteigervariante[80] für 5 benannte Entwickler. Die IDE unterstützt den kompletten Vorgang vom Prototyping inkl. grafischem Layout-Editor bis hin zum Deployment in die Cloud, auf das SAP-System (bei Verwendung des SAP HANA Cloud Connector), oder in das SAP-Portal (bei Verwendung des entsprechenden SAP Web IDE Plugins).

- SAP-Development-Tools für Eclipse
  Alternativ zur SAP Web IDE kann eine Eclipse-basierte Umgebung genutzt werden, bei der sich die benötigte SAPUI5-Funktionalität über die SAP-Development-Tools für Eclipse[81] nachrüsten lassen. Innerhalb von Eclipse wird dann lokal entwickelt und getestet. Die fertige SAPUI5-Anwendung/Komponente kann direkt auf das SAP-System deployed werden (ab AS ABAP 7.31 mit Team-Provider-Add-on). Das Plug-in befindet sich derzeit bei der SAP in Wartung, d.h. es findet keine Weiterentwicklung mehr statt. Kompatibilitäts-Updates werden noch ausgeliefert.

BEST PRACTICE

- Die Empfehlung durch den SAP Fiori 2.0 Guide ist, SAPUI5 User Interfaces mit der SAP Web IDE zu entwickeln und für SAP Backend Services Eclipse ADT zu verwenden, da für Core Data Services (CDS) auch kein Support im SAP GUI existiert (Stand 2016).
- Die SAP Web IDE bietet die umfassendsten Tools und Möglichkeiten zur SAPUI5-Entwicklung in der Cloud. Für das Testen gegen einen OData Service auf dem SAP Gateway (http Destination) und das Deployment wird zusätzlich der SAP HANA Cloud Connector benötigt, der über den Internet-Proxy einen Tunnel und damit eine Vertrauensstellung zur SAP Cloud aufmacht. Im SAP Cloud Connector können dann noch freigegebene SAP-Systeme und -Services autorisiert werden.
- Ist die Verwendung des Cloud-Connectors nicht möglich, kann die mit der SAP Web IDE entwickelte App exportiert (ZIP Datei) und anschließend via Eclipse ADT oder über den Report „/UI5/UI5_REPOSITORY_LOAD“ in das ABAP-System deployed werden.
- Alternativ kann Eclipse mit den entsprechenden Tools verwendet werden. Man muss dann allerdings auf graphische Editoren (WYSIWYG Layout-Editor), Templates und SAP-Fiori-Extension-Support[82] verzichten.

Test

Das SAPUI5 SDK stellt Funktionalitäten für den Unit- und Komponententest zur Verfügung, über die die Testfälle weitestgehend automatisiert werden können.

77 SAPUI5 Explored (UI Explorer)
78 SAP Fiori Demo Cloud
79 openSAP Kurs „Build Your Own SAP Fiori App in the Cloud – 2016 Edition“
80 SAP Web IDE Einsteigervariante für 5 Entwickler
81 SAP-Development-Tools für Eclipse
82 SAPUI5 Extension using component configuration

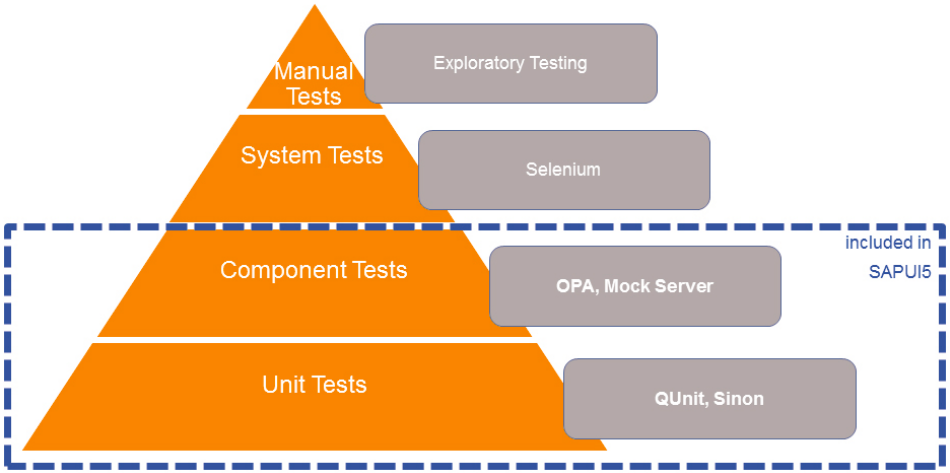


Abbildung 7: Unterstützte Testphasen in SAPUI5

SDK

Alle relevanten Informationen rund um die SAPUI5-Entwicklung finden sich im SAPUI5-Demo-Kit[83]. Hierbei handelt es sich um eine von SAP bereitgestellte Sammlung von Wissen rund um das Thema SAPUI5.

BEST PRACTICE

- Jeder SAPUI5-Entwickler sollte vor einem Projekt den Developer Guide (Bestandteil des SAPUI5-Demo-Kit) durchgearbeitet haben. Insbesondere das Walkthrough Tutorial demonstriert die grundsätzliche Anwendung zentraler Funktionalitäten.
- Bei der Entwicklung können oft Code-Beispiele aus der Explorer-App direkt in das eigene Coding übernommen und dort angepasst werden.
- Eine nach den SAP Fiori Guidlines orientierte Entwicklung setzt auf die sap.m Bibliothek auf. Die Komponenten der sap.ui.commons sind deprecated, d.h. auf die Verwendung sollte nach Möglichkeit verzichtet werden. Die hier aufgeführten Controls sind in neuerer Form weitestgehend in den sap.m-Komponenten redundant enthalten.

Schnittstellen

Für die Kommunikation der SAPUI5-Anwendung mit einem SAP-Backend-System (AS ABAP) wird dem SAP Gateway ein Werkzeug zur Verfügung gestellt, das als Service-Provider sowohl die benötigten OData Rest Services als auch das SAPUI5 SDK anbietet. Die Service-Implementierung erfolgt dabei mit Hilfe von ABAP Objects.

Innerhalb des SAPUI5-Frameworks erfolgt die Service Kommunikation i.d.R. über entsprechende Datenmodelle. Je nach Anwendungszenario existieren hierzu folgende Modelle:

- OData[84]
  Das OData-Model definiert Datentypen und Strukturen und erlaubt einen REST-basierten Zugriff auf Backend-Systeme. Das SAP Gateway unterstützt OData über entsprechende Services und ab NW 7.40 über Core Data Services (CDS). Ab NW 7.50 kann der CDS direkt über eine Annotation als OData Servcie verfügbar gemacht werden, vorher muss dieser noch manuell in der SEGW angelegt werden werden.

83 SAPUI5-Demo-Kit

84 OData-REST-Spezifikation

- JSON
  Datenaustausch über Standard-JSON-Strukturen über http(s) Aufrufe. Im AS-ABAP-System kann hierzu der JSON-RPC-Service genutzt werden, oder ein HTTP-Handler.

- XML
  Datenaustausch über XML-Strukturen. Der AS-ABAP-Server stellt hierzu XML-Transformationen zur Verfügung.

Zusätzlich zu den Modellen, die einen Request-basierten Zugriff auf Backend-Systeme ermöglichen, existiert mit der WebSocket api eine weitere Technologie:

- WebSocket
  Push-Service-Technologie, über die ein Backend-System aktiv Daten an den Client pushen kann. Hierzu kann das SAP-spezifische SAP Push Channel Protocol (PCP) verwendet werden. Das SAP Gateway stellt hierzu APC/AMC-Dienste zur Verfügung.

BEST PRACTICE

- Das OData-Model in Version 2 ist das von SAP favorisierte Datenaustauschmodell für den Zugriff auf Backend-Systeme (Stand Q2/2016). Im SAP Gateway sind sowohl Version 2 und 4 implementiert
- Nach Möglichkeit sollte die gesamte Kommunikation auf Basis von OData Services erfolgen, wobei eine Komponente mit nur einem OData Service kommunizieren sollte (Empfehlung SAP-Fiori-Design und gleichzeitig Restriktion der SAP Web IDE).
- In bestimmten Szenarien bietet es sich an, zusätzlich die WebSocket-Kommunikation mit in das Anwendungsszenario aufzunehmen. Hierdurch kann auf Server-seitige asynchrone Prozesse in Realtime reagiert werden.

10.2.3 GENERELLE EMPFEHLUNGEN

- Neue Implementierungen sollten sich immer an den SAP Fiori Design Guidelines[85] orientieren und die sap.m Library verwenden. Nur hier ist gewährleistet, dass man nicht auf sogenannte „deprecated"-Komponenten setzt, die nicht mehr weiterentwickelt werden und irgendwann aus der Library entfernt werden könnten.

- Für das UI-Design sollten ausschließlich XML-Views verwendet werden, da nur diese von Tools wie dem SAP Web IDE unterstützt werden. Das SAPUI5 Extension Framework unterstützt nur XML-Views im Rahmen der Erweiterung von SAP-Fiori-Apps.

- Die Bibliothek sap.ui.commons beinhaltet Komponenten, die nicht Bestandteil des SAP Fiori Design Guidelines sind, und die auf der UI5con als deprecated genannt wurden. Sofern möglich, sollte vollständig auf diese Bibliothek verzichtet werden. Viele Kunden haben auf dieser Basis größere Anwendungsszenarien gebaut, die mittelfristig auf sap.m migriert werden müssen, um Upgrade-fähig zu bleiben.

- Für den Einstieg bietet sich die Verwendung eines SAP-Web-IDE-Templates an. Leider unterscheiden sich die Templates relativ stark in der Qualität, da sich diese an den SAPUI5-Möglichkeiten zum Erstellungszeitpunkt orientieren. Am besten sollte ein passendes Template auf Basis des höchsten Versionstands genutzt werden.

- In der SAP Web IDE können Projekte auf Basis der SAP Fiori Reference Apps[86] angelegt werden, die einen Best-Practice-Ansatz für die Themen Shop, Approve Purchase Order und Manage Products abbilden.

- Vor der Entwicklung relevanter Services bietet sich ein UI-Prototyp mit Mock-up-Daten an, da sich die relevanten SAP Annotations[87] in der SAP Web IDE einfacher beschreiben und testen lassen als im SAP Gateway Designer.

- Der Zugriff auf Business-Daten sollte über OData-Entitäten erfolgen, die vom SAP Backend-System (z.B. Gateway) zur Verfügung gestellt werden. Die gesamte Geschäftslogik wird dadurch im Backend kontrolliert und verwaltet. Die SAPUI5-Oberfläche mappt dabei nur die relevanten Feldinformationen und steuert Workflows und Sichtbarkeiten.

85 SAP Fiori Design Guidelines
86 SAP Fiori Reference Apps
87 SAP Annotations for OData Version 2.0

- Nach Möglichkeit sollte man versuchen, Smart Controls[88] (zumindest Smart Fields) einzusetzen, damit die Datentypsteuerung durch den OData Service erfolgen kann. Wenn man keine OData Services basierend auf annotierten CDS Views verwenden kann (nur ab NW 7.50), lassen sich die benötigten SAP Annotations auch manuell im SAP Gateway deklarativ verwenden.

- SAPUI5 unterstützt derzeit keine kundenspezifischen Namensräume (z.B. „/SAP/FLIGHT"). Bei der Verwendung von SmartControls sollten die relevanten Gateway Services im Z-Namensraum liegen.

- Für die Adaption und Erweiterung bestehende Anwendungen und Komponenten sollte man das Extensibility-Konzept[89] von UI5 anwenden.

- Für das Theming wird i.d.R. der UI Theme Designer verwendet, der auf allen unterstützten Systemen vorhanden ist. I.d.R. basiert ein eigener Theme auf dem SAP BlueCrystal Theme. Seit SAPUI5 1.38 wird an dem neuen Belize Theme in zwei Ausprägungen gearbeitet, der das neue Design von SAP Fiori 2.0 abbilden soll. Der Belize Theme wird mit Version 1.40.x der neue Standard Theme und ersetzt BlueCrystal.

10.2.4 WEITERE QUELLEN

- SAP Fiori Design Guidelines
- SAP User Experience Community
- SAP Fiori Cloud Demo
- SAPUI5 SDK
- SCN SAPUI5 Developer Center
- SAP Web IDE
- OpenUI5
- openSAP-Kurse
  a. Developing Web Apps with SAPUI5
  b. Build Your Own SAP Fiori App in the Cloud – 2016 Edition

88 SAPUI5 Smart Controls
89 Extensibility-Konzept

10.3 SAP GATEWAY

Der SAP Gateway dient als Vermittler zwischen Web-Technologien und klassischen SAP-Lösungen. Mit dem SAP Gateway werden Daten aus ABAP-basierten Backend-Systemen exponiert.

10.3.1 EINSATZ VON SAP GATEWAY

Die Funktionalität von Backend-Systemen, die u. U. größeren Änderungen unterliegt, sollte nicht direkt nach außen exponiert werden. Das Ziel ist, mittels APIs verschiedensten Konsumenten stabile Zugriffspunkte auf die Systeme anzubieten (siehe API Economy[90]).

Das Hauptanwendungsgebiet für den SAP Gateway ist die User-Interface-Integration. Typische Konsumenten sind Websites, native Mobile Apps, Web Apps auf Java-Script-Basis (siehe Fiori und UI5) oder auch Anwendungen einer Cloud-Plattform. Während bisher in der SAP-Entwicklung meist RFC-basierte Technologien eingesetzt wurden, dominieren für die o.g. Anwendungsszenarien RESTful Services, meist kombiniert mit dem OData-Protokoll.

OData

Das OData-Protokoll ist ein Datenaustauschstandard für das Web, der volle CRUDQ-Funktionalität (Create, Read, Update, Delete, Query) besitzt. Daher wird es auch als ODBC für das Web bezeichnet. OData basiert auf dem Entity Data Model, das jede modellierte Entität (Objekt) sowie deren Beziehungen untereinander darstellt.

Als Datenformat werden JSON (JavaScript Object Notation) sowie ATOM/XML unterstützt.

90 CW: Artikel API Economy

Deployment-Optionen

Der SAP Gateway kann in drei verschiedenen Varianten betrieben werden:

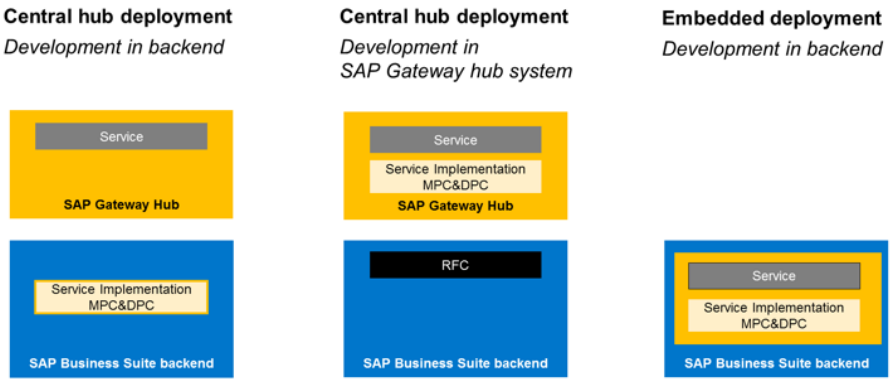


Abbildung 8: SAP Gateway Deployment Optionen [91]

BEST PRACTICE

Die Empfehlung ist, die erste Variante (Central Hub Deployment, Backend Development) zu nutzen. Die Hauptgründe liegen in der besseren Skalierbarkeit und der besseren Einbettung in die Netzwerkinfrastruktur (z.B. Gateway in DMZ). Die Service-Implementierung (Business-Logik) sollte im Backend erfolgen. Die zweite Variante sollte nur zum Einsatz kommen, wenn das Backend-System nicht den notwendigen Release-Stand hat bzw. keine Möglichkeit besteht, die notwendigen Voraussetzungen (Komponente IWBEP, NW-Basis-Release < 7.40) zu deployen. Die Variante des Embedded Deployment kann einen schlanken Start bieten, sollte in der Variante allerdings nicht als Dauerlösung betrieben werden.
Mit der SAP Fiori Cloud Edition gibt es noch eine vierte Deployment Option, bei der SAP Gateway Hub in der HANA Cloud Platform betrieben wird.[92]

10.3.2 ENTWICKLUNG MIT SAP GATEWAY

RESTful Services sind ein wichtiger Baustein in der Realisierung von stateless Web-Apps, d.h. SAPUI5-basierten Apps. Hierbei wird im Backend-System kein Zustand der App bzw. Nutzersession gehalten, sondern es ist immer nur der Zustand der letzten Kommunikation bekannt. Dies steht im Gegensatz zur klassischen ABAP-Programmierung mit SAP GUI, wo das Backend immer den Zustand der Clients verwaltete (stateful-Anwendungen). Das hat Konsequenzen in der Programmierung, da bei Änderungen von Daten im Umfeld hochfrequentierter, paralleler Zugriffe nicht mehr die klassischen Sperrlogiken verwendet werden können. Weiterhin findet kein Session-Handling im klassischen Sinne statt, d.h. beim Schließen der Browser-Sitzung ist immer nur der letzte Zustand nach dem letzten Aufruf eines OData-Services im Backend bekannt. Wenn Service-Aufrufe idempotent sind, d.h. immer das gleiche Resultat im Backend hervorrufen (Anlage, Löschen), gibt es keine Probleme. Schwierig wird es bei Änderungen, da Sperren nur solange gesetzt werden sollten, wie der OData-Aufruf im

91 SAP Help: SAP Gateway Deployment Optionen

92 SCN SAP Gateway Deployment Options in a nutshell

Backend erfolgt. Bei erneuter Abfrage besteht daher die Möglichkeit, dass von anderer Seite die Daten in der Zwischenzeit geändert wurden. Hier muss bei der Entwicklung entschieden werden, ob die Änderung relevant ist oder nicht. Eine nicht relevante Änderung wäre z.B. ein Genehmigungsschritt, wo nach der FCFS-Regel der erste Änderer gewinnt, bei weiteren Aufrufern dann nur ein Refresh der Web-App mit der Änderung erfolgt. Für kritischere Schnittstellen, in denen die Datensynchronisierung zwischen Backend und dem API-Konsumenten eine große Rolle spielt, ist die ETag-Funktionalität[93] des Gateway zu empfehlen. Damit kann die Aktualität der Daten in der Anwendung kontrolliert und entsprechend darauf reagiert werden. Nichtsdestotrotz gibt es Möglichkeiten klassische Sperrkonzepte[94] weiterhin zu verwenden. Hierfür sollte man allerdings einen höheren Entwicklungsaufwand einplanen. Weiterhin können Konzepte herangezogen werden, wie man sie bspw. aus der IDoc-Eingangsverarbeitung kennt.

Bei der Entwicklung mit dem SAP Gateway werden meist drei Phasen unterschieden. Anhand dieser Phasen werden Vorschläge für Namenskonventionen sowie Best Practices dargestellt. Für die Entwicklung von OData-Services bietet SAP den Gateway Service Builder an (Transaktion SEGW).

OData-Modellierung

In Entwicklungsprojekten kommt dem API-Design eine wichtige Rolle zu. Hier hat man einen klassischen Zielkonflikt zwischen Wiederverwendbarkeit der APIs und dem schnellen, möglichst effizienten Verfügbarmachen der Daten. Auf keinen Fall sollte man einfach das Interface von (Standard-)Funktionsbausteinen 1:1 exponieren. Besser geeignet ist bspw. die Kapselung der Funktionalität durch Wrapper-Funktionsbausteine. Das API-Design sollte von den Anforderungen des API-Konsumenten (z.B. einer SAPUI5-Web-App) getrieben sein.

Für die OData-Modellierung empfehlen wir daher die folgenden Hinweise zu beachten:

- Entitätstypen: Entitäten sollten auf den Anwendungsfall zugeschnitten werden und nicht vom zugrundeliegenden Datenbankmodell getrieben sein. Der Anwendungsfall sollte sich im Entitätsnamen widerspiegeln.
- Komplexe Typen: Dienen der Strukturierung von Entitäten.
- Entitätsmengen: Gruppe eines Entitätstyps, SAP schlägt Suffix-Set vor, alternativ die Mehrzahl einer Entität.
- Assoziationen: Beziehungen sollten soweit möglich gebildet werden. Einerseits wird die Beziehung der Daten untereinander verdeutlicht als auch die Anzahl der Aufrufe verringert.
- Navigationseigenschaften: Sollten bei Relation 1 den Entitätstypen bzw. bei Relation N die Entitätsmenge[95] verwenden.

Service-Implementierung

Bei der Implementierung sollten die gängigen ABAP Objects-Richtlinien und -Namenskonventionen angewandt werden.

Bei Verwendung des Gateway Service Builders ist zu beachten, dass die generierten Klassen nur die ersten 20 Zeichen des Gateway-Projektnamens verwenden. Idealerweise sollte der Projektname entweder so gewählt werden, dass die ersten 20 Zeichen bereits eine korrekte Service-Identifizierung erlauben, oder der Name muss für die zu generierenden Klassen angepasst werden. Ansonsten wird automatisch ein Zähler an die generierten Klassen (Model Provider Class, Data Provider Class) angehängt, was die Identifizierung der Klassen bei der Entwicklung erschwert.

Service-Konfiguration

Der externe Servicename sollte Rückschluss auf die Funktionalität geben.
Vorschläge zur Namenskonvention:

- Technischer Servicename:
- Z + <externer Servicename>
- Der _SRV Suffix wird automatisch seitens SAP angehangen.
- Externer Servicename:
- Namespace: /<Kunden Namespace>/ (falls Kundennamesraum verwendet wird, ansonsten Z*)
- Optional: GW_* als Präfix oder Infix erlaubt eine einfache Zuordnung als Gateway-Service
- System-Alias: Da System-Aliase transportiert werden können, ist die Verwendung des Alias-Namen als Landschaftsnamen zu empfehlen.

93 SAP-Help: ETag Handling
94 SCN-Blog: Stateful Sessions
95 SCN-Blog: OData Templates

10.3.3 GENERELLE EMPFEHLUNGEN

Der folgende Abschnitt enthält einige Leitlinien, die in der Entwicklung mit SAP Gateway zu beachten sind. Bei einigen Punkten wird entsprechend auf weiterführende Quellen verwiesen.

OData/ Funktionalität

- Eine gute Übersicht über OData Best Practices[96] wird auf SAP-Help angeboten.
- Bei der OData-Service-Entwicklung sollte dem „Separations of Concerns" Designprinzip gefolgt werden. Business-Logik gehört ins Backend, die Service-Implementierung sollte sich auf „Glue Code" beschränken.
- Es wird nicht die komplette OData-Syntax angeboten. Der SAP-Hinweis 1574568 enthält eine Übersicht was unterstützt wird, z.B. existieren Einschränkungen für $filter -Optionen.
- Der UI-Entwicklungsprozess wird durch die Verwendung von OData Annotations[97] stark vereinfacht. Insbesondere ab Release-Stand 7.50 ist daher die Verwendung von Annotations in OData Services zu empfehlen.
- Bei Verwendung mehrerer Backend-Systeme innerhalb einer SAPUI5 Web App ist die Multi-Origin-Composition (MOC)-Funktionalität empfehlenswert.

Performance

- Statt vieler paralleler Aufrufe sollten $batch und $expand[98] genutzt werden.
- OData bietet mit der $select-Option eine Möglichkeit zum Einschränken der zu übertragenden Daten. Generell sollten immer nur so wenige Daten wie notwendig übertragen werden.

Sicherheit

- TLS-Nutzung ist verpflichtend (HTTPS)
- Bei Verwendung des SAP Gateway als Hub sollte eine Vertrauensbeziehung[99] (Trusted-RFC-Verbindung) zwischen dem Gateway und dem gerufenen Backend-System bestehen.
- Der SAP-NetWeaver-ABAP-Applikationsserver unterstützt unterschiedliche Authentifizierungsmethoden, die in SAP Gateway genutzt werden können. Abhängig vom Szenario (Desktop-Anwendungen, Web-Anwendungen, ...) werden unterschiedliche Methoden empfohlen (Kerberos, SAML 2.0, ...) empfohlen.[100]

10.3.4 WEITERE QUELLEN

- SAP Community Network
- SAP-Online-Hilfe

96 *SAP-Help: OData do's and dont's*
97 *SCN-Blog: Annotations in ABAP*
98 *SCN-Blog: Perf do's and dont's*

99 *SAP-Help: Trusted System*
100 *SAP NetWeaver Gateway Authentication and Single Sign-On*

11 DIE AUTOREN

Die folgenden Autoren waren maßgeblich an der Erstellung der 2. Auflage des Leitfadens beteiligt:

Dr. Christian Drumm
Leiter Anwendungsentwicklung & Beratung, FACTUR Billing Solutions GmbH
Dr. Christian Drumm arbeitet seit 2004 im SAP-Umfeld. Nach seiner Zeit bei SAP-Research arbeitet er in verschiedenen Rollen (Entwickler, Projektleiter, Architekt) in der Beratung mit den Schwerpunkten SAP CRM und SAP IS-U. Seit 2013 leitet er die Abteilung Anwendungsentwicklung und Beratung der FACTUR Billing Solutions GmbH.

Martin Fischer
Portfolio Unit Manager SAP Database & Technology, bridgingIT GmbH
Martin Fischer ist seit 2001 im SAP-Umfeld tätig. Gestartet ist er als Modulbetreuer im Bereich SAP FI/CO. Seit 2007 hat er den Schwerpunkt auf Software-Entwicklung und -architektur mit Schwerpunkt ABAP gelegt. Nach Stationen bei verschiedenen Beratungshäusern ist er seit 2011 bei bridgingIT. Dort verantwortet er fachlich den Portfolioobereich SAP Database & Technology.

Judith Forner
Senior Consultant Finance & Controlling, Mundipharma Deutschland GmbH & Co. KG
Judith Forner ist seit 1999 als SAP-Beraterin und ABAP-Entwicklerin tätig. Neben der klassischen Prozessberatung in den Modulen FI, CO und PM liegen ihre Schwerpunkte in der anwenderfreundlichen Erweiterung des SAP-Standards und der Integration externer Systeme.

Edo von Glan
SAP-Entwickler, Drägerwerk AG & Co. KGaA
Edo von Glan hat als Software-Entwickler für IBM und die Commerzbank, sowie sieben Jahre in der Produktentwicklung bei SAP gearbeitet. Seit 2008 arbeitet er in der SAP-Entwicklungsabteilung bei Dräger, davon sechs Jahre in leitender Position. Der Verantwortungsbereich umfasst die Applikations- und Schnittstellentwicklung sowie Custom Code Lifecycle Management und Software-Qualität für die im Unternehmen zentral verwalteten SAP-Systeme.

Florian Henninger
Senior Consultant SAP Development, FIS GmbH
Florian Henninger ist zertifizierter ABAP-Entwickler und seit 2010 im Bereich SAP-ABAP-Entwicklung tätig. Sein fachlicher Schwerpunkt liegt im Bereich Core Development/Enhancements sowie dem Thema Output Management bei ERP-Einführungsprojekten und Bestandssystemen. Außerdem ist er Mitglied des Qualitäts Managements, Coach und SAP-Mentor.

Martin Hoffmann
Head of Software Engineering, Miele & Cie. KG
Martin Hoffmann ist seit 1987 im Entwicklungs-nahen Umfeld tätig und konnte Erfahrungen als Entwickler und in verschiedenen leitenden Positionen im Entwicklungsumfeld sammeln. Seit 2007 verantwortet er bei Miele den Bereich Software-Engineering.

Valentin Huber
Senior IT Consultant, msg systems ag
Valentin Huber ist seit 2012 als ABAP-Entwickler und Sofware-Architekt für kundeneigene Entwicklungen tätig. Zuvor sammelte er Erfahrung in der C++ und Java-Entwicklung. Als Certified Professional for Software Architecture (Foundation Level) nach iSAQB und Clean-Code-Development-Verfechter liegen ihm insbesondere die Erweiterbarkeit und Wartbarkeit von Softwaresystemen am Herzen.

Jens Knappik
SAP System Architect, thyssenkrupp Materials Services GmbH
Jens Knappik ist seit 2004 als ABAP-Entwickler in einem internationalen Projektumfeld tätig. In seiner Rolle als Lead Developer begleitete er bis 2012 die Produktentwicklung in den Modulen SD-CAS und CS. Seitdem ist er als Berater für die strategischen Ausrichtung und Definition von Standards im ABAP-Umfeld für das zentrale Business-Area-Template aktiv.

Dr. Christian Lechner
Principal IT Consultant, msg systems ag
Dr. Christian Lechner ist seit 2005 im Bereich der SAP-Softwareentwicklung (Standard und Custom Development) in verschiedenen Rollen (Entwickler, Projektleiter, Softwarearchitekt) tätig. Seit 2015 leitet er die Architekturabteilung des Geschäftsbereichs Development der msg systems ag.

Steffen Pietsch
Head of Backoffice, Haufe-Lexware GmbH & Co.KG
Steffen Pietsch ist seit 2003 in Entwicklungs-nahen SAP-Themenbereichen tätig und konnte umfangreiche Praxiserfahrung als Entwickler sowie in verschiedenen leitenden Positionen im Beratungs- und Entwicklungsumfeld sammeln. Seit 2009 setzt er sich als Sprecher des DSAG-Arbeitskreises Development für die Interessen der Kunden und Partner in Zusammenarbeit mit der SAP ein.

Daniel Rothmund
IT Business Analyst SAP, Geberit Verwaltungs GmbH
Daniel Rothmund ist seit 2008 in verschiedenen Bereichen im SAP-Umfeld tätig. Seit 2016 ist er für das zentrale SAP-Development-Team bei Geberit verantwortlich. Seit 2015 setzt er sich als Sprecher der DSAG-Arbeitsgruppe UI Technologie für die Interessen der Kunden und Partner in Zusammenarbeit mit der SAP ein.

Holger Schäfer
Business Unit Manager, UNIORG Solutions GmbH
Holger Schäfer ist seit 1999 im SAP-Umfeld in den Schwerpunktthemen User Interfaces, Integration, HANA/HCP und E-Commerce tätig. Er engagiert sich in der Arbeitsgruppe UI-Technologien und in der OpenUI5 Community. Seit 2010 verantwortet er die strategische Geschäftsfeldentwicklung im Bereich neuer SAP-Technologien.

Denny Schreber
Senior Solution Architect, cbs Corporate Business Solutions Unternehmensberatung GmbH
Denny Schreber arbeitet seit 2007 in SAP-nahen Technologiethemen. Sein Schwerpunkt liegt dabei auf Mobile & UI-Technologien sowie Integrationsthemen. Zuvor war er im Bereich der Java-Software-Entwicklung tätig. Seit 2015 ist er stellvertretender Sprecher in der DSAG-AG UI-Technologien.

Andreas Wiegenstein
Chief Technology Officer (CTO) und Mitgründer, Virtual Forge GmbH
Andreas Wiegenstein arbeitet seit 2002 im Bereich SAP-Security mit dem Schwerpunkt Penetrationstests. Er ist Co-Autor des Buches „Sichere ABAP-Programmierung“ (SAP Press) und hält regelmäßig Vorträge zum Thema SAP-Sicherheit auf internationalen Konferenzen, wie z.B. Troopers, RSA, Black Hat, BIZEC, IT Defense, Deep Sec, Hack In The Box, SAP TechEd und auf DSAG-Veranstaltungen.

Bärbel Winkler
Systemanalystin SAP-Basis/Programmierung, Alfred Kärcher GmbH & Co. KG
Bärbel Winkler arbeitet seit 2000 im SAP-Umfeld in der ABAP-Programmierung und Projektarbeit und ist seit 2011 bei der Alfred Kärcher GmbH & Co. KG. in kundeneigene Entwicklungen und deren Koordination eingebunden. Zu ihren Aufgaben gehört die regelmäßige Aktualisierung der Entwicklungsrichtlinien, die Beurteilung und Schätzung eingehender Entwicklungsanforderungen sowie die Begleitung der Umsetzung bei extern durchgeführter Programmierung.

ANHANG A: NAMENSKONVENTIONEN

Beim Thema Namenskonventionen gibt es für ABAP zwei konträre Ansätze. Der wesentliche Unterschied liegt darin, ob der Datentyp (Struktur, Tabelle, Referenz, Elementar, ...) mit einem Präfixbuchstaben in den Objektnamen „kodiert" (English: encoded) wird oder nicht.

Namenskonventionen, bei denen der Datentyp als Präfix zum Objektnamen vorgeschrieben wird, fordern in der Regel auch noch weitere technische, fachliche oder organisatorische Kategorien, die ebenfalls in den Präfix aufgenommen werden sollen. Insofern könnte man von „umfassenden" Namenskonvention sprechen. Es besteht eine Verwandtschaft zu der bei Microsoft erfundenen Ungarischen Notation[101].

Bei genauer Betrachtung wird die umfassende Namenskonvention in der Regel nur halbherzig umgesetzt, da die Konvention bei geschachtelten Datentypen nur auf der ersten Ebene stattfindet (LS_CUSTOMER-PARTNER-HISTORY statt LS_CUSTOMER-S_PARTNER-T_HISTORY).

Im Gegensatz dazu steht die „minimale" Namenskonvention, wo bisweilen nur die Präfixe zur Unterscheidung von Parametern (I_ für IMPORTING, E_ für EXPORTING, usw.) vorgeschrieben werden, und stattdessen der Fokus auf einem sinnvollen „sprechenden" Namen liegt. Für diese Art der Namenskonvention sprechen sich unter anderem die Autoren der „ABAP-Programmierrichtlinien" (jetzt Teil der F1-Hilfe) aus.

Vorteile von umfassenden Namenskonventionen

- Beim Lesen des Quellcodes ist der Objekttyp sofort ersichtlich, ohne dass z.B. in den DDIC navigiert werden muss.
- Ungewollte Verschattungen (z.B. Typen in Methoden gegenüber Typen in der Klasse) können besser vermieden werden.
- Objektnamen wirken einheitlich und ordentlich durch den einheitlichen Präfix (auch wenn der sprechende Teil der Namen vielleicht schlecht gewählt ist).

Nachteile

- eine vollständige und eindeutige Systematik ist umfassend und komplex, siehe die Diskussion im Buch „ABAP-Programmierrichtlinien".
- Die Lesegeschwindigkeit von Quelltext sinkt, da technische mit semantischen Informationen vermischt werden.
- Es entsteht zusätzlicher Aufwand, wenn sich der technische Typ bzw. die Sichtbarkeit ändert. Ebenso wenn z.B. die Zuordnung zur Anwendungshierarchie / Organisationseinheiten Teil des Namens ist und die Objekte in einen anderen Bereich umziehen sollen.
- Semantische Informationen können auf Grund von Platzmangel verloren gehen.
- Das Erlernen, Kontrollieren und Aktualisieren der Konventionen ist mit zusätzlichem Aufwand verbunden.

Vorteile von minimalen Namenskonventionen

- Mehr Platz und Aufmerksamkeit für den sprechenden Namen, sowohl beim Schreiben als auch beim Lesen.
- keine redundanten Informationen (Datentyp lässt sich durch Vorwärtsnavigation / F2 (ADT) ermitteln, Herkunftssystem steht im Objektkatalog, die Zuordnung zur Anwendungshierarchie lässt sich über das Paket ableiten, usw.).

Die erste Version dieses Leitfadens verfolgte eine Variante der umfassenden Namenskonvention, von der wir uns aufgrund unserer eigenen Erfahrungen, sowie dem aktuellen Diskusionsstand in der Fachliteratur und im SCN (siehe Anhang A.3) distanzieren.

Im Folgenden finden Sie ein Beispiel, wie eine eher schlanke Namenskonvention aussehen kann.

Allgemeine Hinweise:

- Objekte im ABAP Dictionary haben unterschiedliche Begrenzungen in der Anzahl zur Verfügung stehender Zeichen. Bei der Benennung der Objekte ist dies zu berücksichtigen.
- Nachfolgend wird der Kundennamensraum Y... verwendet. Stattdessen kann, wie in Kapitel 2 beschrieben, alternativ der Namensraum Z... oder ein kundeneigener Namensraum /.../ verwendet werden.

101 https://de.wikipedia.org/wiki/Ungarische_Notation

A.1 NAMENSKONVENTIONEN REPOSITORY-OBJEKTE

Namenskonventionen im ABAP Repository verfolgen das Ziel, Namenskonflikte durch Namensüberlagerung zu vermeiden und die Namen der Objekte durch Anwendung einer standardisierten Vorgehensweise verständlicher zu machen. Zusätzlich sollen sie dem Entwickler Entscheidungen abnehmen, so dass er sich auf die Umsetzung von Anforderungen konzentrieren kann und keine Aufwände in eine individuelle Vermeidung von Namensüberlagerung investieren muss. Des Weiteren dient die Konvention im Quellcode als Hilfestellung für den Einsatz der zu dem Objekt passenden Operatoren (Beispiel INSERT statt APPEND beim Anhängen einer neuen Zeile in eine sortierte Tabelle). Repository-Objekte unterscheiden sich in den meisten Fällen[102] durch ihren Objektkatalogeintrag voneinander, weshalb aus technischer Sicht bei der Namensvergabe nur sehr wenigen Konflikte entstehen können. Daneben werden für diverse Objekte, wie z.B. Ausnahmeklassen (Präfix = CX) oder Sperrobjekte (Präfix = E), Namenskonventionen durch die Entwicklungsumgebung erzwungen[103].

Die nachfolgenden Konventionen erweitern die Vorgaben aus oben genanntem Hinweis. Das Ziel der Richtlinien ist die Bereitstellung praxistauglicher, pragmatischer Regeln die auf Überregulierung weitestgehend verzichten. Hierzu werden nur für konkurrierende Objekte Namenskonventionen festgelegt, die sich am SAP-Standard orientieren (z.B. Präfixkonventionen für Typinformationen aus dem BOPF-Framework).

Die Namenskonventionen setzen sich aus folgenden Bereichen zusammen und werden zum Teil durch Unterstriche voneinander getrennt:

Bereich	Bedeutung	Beispiel
[Namensraum]	Kundennamensraum oder reservierter Kundennamensraum.	Z = Entwicklung ohne Transportschicht Y = Transportrelevante Entwicklung
[Typinformation]	Technisches Objektkürzel für einen bestimmten Typ. Nicht für jedes Objekt notwendig.	CL = Globale Klasse
[Kontext]	Zusammenfassendes Arbeitsgebiet, ein Produkt oder die Zuordnung zur Anwendungshierarchie bzw. Softwarekomponente. In der Regel wird der Kontext über ein Struktur- oder Hauptpaket gebildet.	Anwendungshierarchie SD: Sales & Distribution
[Semantische Information]	Informationen über ein Domänenobjekt, dessen Absicht oder bekannte Anwendungsmuster basierend auf einer allgegenwärtigen, verständlichen Sprache (s. Kapitel 2.3).	OPEN_ITEMS_LIST
[Muster]	Ein Muster ist ein optionaler Teil der semantischen Information und beschreibt eine etablierte Vorgehensweise im Rahmen des Designs / der Programmierung.	Anwendungen: Worklist / Chart / Overview Design: Master / Detail Implementierung[103] Factory / Data Access Object / Gateway

102 Eine Ausnahme sind Tabellen und Strukturen. Diese haben beide den TADIR-Objekttyp TABL.
103 Weitere Details zu den betroffenen Objekten sind im SAP-Hinweis 16466 hinterlegt

104 Da im Rahmen der Implementierung häufig mehrere Muster gleichzeitig zum Einsatz kommen, sollten die Details über die eingesetzten Muster besser im Quellcode und nicht im Objejktnamen kommentiert werden.

A.1.1 PAKETHIERARCHIE

Typ	Struktur- / Haupt- / Entwicklungspakete: Pakete eignen sich optimal für die Gruppierung mehrerer Entwicklungsobjekte zu einer semantisch zusammenhängenden Einheit.
Namensbildung	[Einzelnes Substantiv | Zusammengesetzte Substantive]

Beispiele	Anwendungshierarchie / Softwarekomponente:
	- Y_DEVELOPMENT_FOUNDATION (Strukturpaket)
	- Y_ERP_CENTRAL_APPLICATIONS (Strukturpaket)
	  - Y_ACCOUNTING (Strukturpaket)
	  - Y_...
	  - Y_LOGISTICS (Strukturpaket)
	  - Y_LOGISTICS_EXECUTION (Hauptpaket)
	  - Y_PROCUREMENT (Hauptpaket)
	  - Y_SALES (Hauptpaket)
	  - Y_COMPUTER_AIDED_SELLINGS (Hauptpaket)
	  - Y_CAS_COMMON (Entwicklungspaket)
	- Y_NETWEAVER_PLATFORM (Strukturpaket)
	- Y_SAP_DELIVERABLES (Strukturpaket)
	  - Y_RAPID_DEPLOYMENT_SOLUTIONS (Hauptpaket)
	  - Y_SAP_BEST_PRACTICES (Hauptpaket)
	  - Y_SAP_NOTES (Hauptpaket)
	  - Y_SNOTE_2294645 (Entwicklungspaket)

Typ	Unterpakete: Unterpakete teilen das übergeordnete Paket in spezialisierte Arbeitsbereiche auf. Um die Zugehörigkeit der Objekte hervorzuheben empfiehlt es sich, für alle in dem Paket / der Unterpakethierarchie enthaltene Objekte den gleichen Präfixnamensraum zu verwenden.
Namensbildung	[Abkürzung des übergeordneten Paketes] [semantische Information]

Beispiele	- Y_COMPUTER_AIDED_SELLINGS (Hauptpaket)
	  - Y_CAS_BUSINESS_OBJECTS (Entwicklungspaket)
	  - Y_CAS_COMMON (Entwicklungspaket)
	  - Y_CAS_CORE (Entwicklungspaket)
	  - Y_CAS_CONFIGURATION (Entwicklungspaket)

Typ	Paketschnittstellen: Die Bezeichnung von Paketschnittstellen dient der Bekanntmachung einer Absichtserklärung, wie mit bestimmten, über die Paketschnittstelle propagierten Objekten, umgegangen werden soll.
Namensbildung	[Paketbezeichnung] [[Sichtbarkeit] | [Zugriffsart] | [Konsument]]
Beispiele	- Y_CAS_SALES_ACTIVITY_PUBLIC - Y_CAS_CONFIGURATION_READ - Y_CAS_CONFIGURATION_WRITE

A.1.2 DICTIONARY OBJEKTE

Typ	Elementare Datentypen: Datenelemente und Domänen beschreiben die technischen und semantischen Eigenschaften eines Daten- oder Referenztypen. Da sie sich durch ihren Objektkatalogeintrag technisch voneinander unterscheiden, ist die Verwendung von identischen Namen möglich.
Namensbildung	[Einzelnes Substantiv | Zusammengesetzte Substantive]
Beispiele	Y_CAS_ACTION_IDENTIFIER (Domäne) Y_CAS_ACTION_IDENTIFIER (Datenelement)

Typ	Strukturtypen: Mehrdimensionale Datentypen enthalten ein oder mehrere Datentypen als Komponenten. Sie unterteilen sich in einzeilige und mehrzeilige Strukturtypen.
Namensbildung	[Einzelnes Substantiv | Zusammengesetzte Substantive]
Strukturen: Strukturen teilen sich historisch bedingt den gleichen Objektkatalogeintrag mit transparenten Tabellen. Daher ist es notwendig, die beiden Typen zwingend per technischem Präfix voneinander zu unterscheiden. Beispiel: YS_CAS_ACTION YS_CAS_MASTER_LINE	
Transparente Tabellen: In der Regel sollte die Präfixbezeichnung D für Transparente Tabellen ausreichen. Falls eine weitere Unterteilung notwendig ist, empfehlen wir auf Grund der Längenbeschränkung von 16 Zeichen das Präfix um die technische Auslieferungsklasse der Tabelle zu erweitern bzw. T für die Kennzeichnung von Texttabellen zu verwenden. Beispiel: YD_CAS_CONFIG YDC_CAS_ACTIONS (Konfigurationstabelle) YDT_CAS_ACTIONS (Texttabelle) YDA_CAS_MASTER (Stamm- und Bewegungsdaten) YDL_CAS_FILEINPT (Temporäre Daten)	
Views: Falls die Präfixbezeichnung V für Views nicht ausreicht, empfehlen wir eine weitere Unterteilung auf Basis des View-Typen einzuführen. Beispiel: YV_CAS_CONFIG YVC_CAS_ACTIONS (Pflege-View) YVH_CAS_ACTIONS (Help-View) YVD_CAS_MASTER (Datenbank-View) YVP_CAS_MASTER (Projektions-View)	
Tabellentypen: Ist die Anlage mehrerer Tabellentypen für denselben Zeilentyp notwendig, empfehlen wir die Zugriffsart in die Präfixbezeichnung aufzunehmen und die semantische Bezeichnung um Informationen über die Schlüsseldefinition zu ergänzen. Anderenfalls ist die Präfixbezeichnung T ausreichend. Beispiel: YT_CAS_ACTIONS YTH_CAS_ACTIONS (Interne Tabelle mit Hash-Zugriff) YTS_CAS_ACTIONS_BY_ACTIVITY (Interne Tabelle sortiert nach dem Feld ACTIVITY)	
Core Data Services: Neu angelegte CDS-Views erzeugen drei Repository-Objekte. Die DDL-Source, ein CDS-View und ein CDS-Datenbank-View. Um CDS-Datenbank-Views von normalen Datenbank-Views zu unterscheiden, empfehlen wir den SAP Standard Beispielen[104] zu folgen und CDS-Datenbank-Views mit dem Präfix SQL zu kennzeichnen. Beispiel: YSQL_CAS_MASTER (CDS-Datenbank-View)	

A.1.3 CONTAINER FÜR QUELLTEXTOBJEKTE

Typ	Ausführbare Anwendungen Aus dem Namen einer Anwendung sollte sich direkt ableiten lassen, was die Anwendung leistet, welches Geschäftsobjekt im Fokus steht und ggf. mit welchem Verfahren die Anwendung zu bedienen ist (Worklist, Wizard, ...). Wir empfehlen bei der Namensvergabe bekannte Bedienverfahren aus dem Floorplan Manager[105] bzw. aus den Fiori Design Guidelines[106] zu adaptieren.
Namensbildung	[Einzelnes Substantiv | Zusammengesetzte Substantive] [Muster]
Beispiele	Y_CAS_CUSTOMER_VISIT_PLANNER Y_CAS_SALES_ACTIVITY_WORKLIST Y_CAS_DATA_MIGRATION_WIZARD

105 SAP-Help: 'Implement the CDS View as Data Model'
106 SAP-Help: 'Floorplans Concept'
107 Fiori Design Guidelines: 'Floorplan Overview'

Typ	Includes für Rahmenprogramme und Funktionsgruppen Includes sollten nach Möglichkeit nicht mehr als wiederverwendbare Modularisierungstechnik verwendet und in Neuentwicklungen durch ABAP Objects abgelöst werden. Ist der Einsatz dennoch notwendig, sollte die Modularisierung am Dynpro bzw. dem aktuellen Geschäftsobjekt ausgerichtet werden.
Namensbildung	[Rahmenprogramm | Abkürzung] [Container Typ] [Dynpro* | Zähler]
Beispiele	Y_CAS_SALES_ACTIVITY_WORKLIST (Report) Y_CAS_SALES_ACTIVITY_WL_TOP (Globaler Deklarationsteil) Y_CAS_SALES_ACTIVITY_WL_O0100 (PBO Dynpro 0100) Y_CAS_SALES_ACTIVITY_WL_I0100 (PAI | POH | POV Dynpro 0100) Y_CAS_SALES_ACTIVITY_WL_F01 (Form Routinen) Y_CAS_SALES_ACTIVITY_WL_C01 (Lokale Klassen) Y_CAS_SALES_ACTIVITY_WL_T01 (Unit Tests)

Typ	Funktionsgruppen für Tabellenpflegeviews Generierte Funktionsgruppen für Tabellenpflegedialoge sollten denselben Namen wie der dazugehörige Pflege-View besitzen.
Namensbildung	[Name des Pflege-Views]
Beispiele	YVC_CAS_ACTIONS (Pflege-View) YVC_CAS_ACTIONS (Funktionsgruppe für Tabellenpflegeview)

Typ	Enhancements Enhancementspots dienen als Container für die über BAdI-Definitionen festgelegten Erweiterungen von Objekten an definierten Stellen. Die Ausprägung kann spezifische Erweiterungsimplementierungen mit diversen BAdI-Implementierungen, Filtern etc. enthalten.
Namensbildung	[Name des zu erweiternden Objekts] [[Filterinformationen] | [Variante]]
Beispiele	Y_CAS_SALES_ACTIVITY_WORKLIST (Erweiterungs-Spot) - Y_CAS_SAWL_DEFAULT_EXTENSION (Erweiterungsimplementierung) - /ABCDEF/CAS_SAWL_EXTENSION (Erweiterungsimplementierung) - /ZYXWVU/CAS_SAWL_EXTENSION (Erweiterungsimplementierung) Y_CAS_SAWL_ADD_VALIDATION (BAdI Definition) - Y_CAS_SAWL_DEFAULT_VALIDATION (BAdI-Implementierung) - /ABCDEF/CAS_SAWL_DV_D0100 (BAdI-Implementierung) - /ZYXWVU/CAS_SAWL_DV_D0200 (BAdI-Implementierung)

ABAP Objects

Wir empfehlen den Vorgaben der SAP für die Benennung von ABAP-Objects-Komponenten[108] zu folgen. Davon ausgenommen sind die Vorgaben vom Typ „Empfehlung“ und die methodenlokalen Konventionen für Parameter. Für Parameter sollten Sie sich an die Empfehlungen im Anhang A.2.3 Signaturen halten. Anstelle der Präfixempfehlungen für z.B. Konstanten und lokale Klassen, sollten Sie besser semantisch passende und gut lesbare Bezeichner verwenden.

108 *SAP-Help: „Naming Conventions in ABAP Objects“*

Beispiel:

```
“ Local class in the local types context of a global class
CLASS sales_activity_constants DEFINITION
  CREATE PRIVATE ABSTRACT FINAL.
  PUBLIC SECTION.
    CONSTANTS:
      BEGIN OF partner_function,
        key_account_manager TYPE tpar-parvw VALUE 'KA',
        ...
      END OF partner_function.
ENDCLASS.
```

A.2 NAMENSKONVENTIONEN ABAP-QUELLTEXT

Namenskonventionen für Quelltext sollen die Verständlichkeit erhöhen, indem für besonders wichtige Typinformationen ein Präfix verwendet wird. Ansonsten liegt der Fokus auf einer klaren, verständlichen Namensgebung (siehe Kapitel 2.3).

A.2.1 KLASSISCHE BENUTZERDIALOGE (SELEKTIONSBILDER / DYNPROS)

Auf Grund der Längenrestriktionen von Komponenten klassischer Benutzerdialoge empfehlen wir beigefügte Präfixbezeichner einzusetzen:

Typ	Präfix
Parameter	p_
Select-Options	s_
Tables Dialogstrukturen für Dynpro-Binding	ohne

A.2.2 SICHTBARKEIT

Unbeabsichtigte Verschattung lokaler Deklarationen lässt sich durch die Definition von Präfixbezeichnern für Signaturen und globale Variablen / Typen vermeiden. Aus technischer Sicht ist die Einführung weiterer Präfixkonventionen nicht notwendig.

Typ	Präfix
Globale Datenobjekte	g_

Ausnahme:

Für lokale Deklarationen in Methoden kann unter folgender Bedingung die Präfixkonvention l_ verwendet werden:

- Die lokale Deklaration verschattet ein statisches Klassenattribut.
- Die Methode muss auf das statische Klassenattribut zugreifen.
- Der Klassenname ist sehr lang.
- Der Zugriff über den Komponentenselektor (=>) führt zu einer schlechten Lesbarkeit.

A.2.3 SIGNATUREN

Wir empfehlen, die Verwendung von Signaturen in den verschiedenen Prozedurtypen (Unterprogramme / Funktionsbausteine / Methoden) zu vereinheitlichen und nachfolgende Präfixbezeichner zu verwenden.

Typ	Präfix
Importing / Using / Tables	i_
Exporting / Tables	e_
Changing / Tables	c_
Returning	r_

A.3 WEITERFÜHRENDE INFORMATIONEN ZU NAMENSKONVENTIONEN

Für Anhänger und Verfechter einer umfangreichen Namenskonvention verweisen wir aus Gründen der Einheitlichkeit auf beigefügten de-facto-Standard für umfangreiche Namenskonventionen aus der SCN-Community:
http://scn.sap.com/people/uwe.schieferstein/blog/2009/08/30/nomen-est-omen--abap-naming-conventions

Eine weitere, etwas reduzierte Variante finden Sie hier:
http://scn.sap.com/community/abap/blog/2016/02/05/fanning-the-flames-prefixing-variableattribute-names

Die entsprechenden Gegenbewegungen zu dem Thema möchten wir Ihnen natürlich nicht vorenthalten:
http://scn.sap.com/community/abap/blog/2013/05/23/abap-code-naming-conventions--a-rant

http://scn.sap.com/community/abap/blog/2015/09/22/hungarian-beginners-course--a-polemic-scripture-against-hungarian-notation

Besonders empfehlenswert sind die zu den oben aufgeführten Blogs dazugehörigen Kommentarpassagen. Ergreifen Sie die Gelegenheit und versetzen Sie sich in die Sicht der jeweiligen Fraktion und lernen die entsprechend positiven und negativen Facetten kennen.

Letzten Endes müssen Sie die Konventionen festlegen, die Ihnen, Ihrem internen und ggf. externen Teams den besten Nutzen im Hinblick auf die Produktivität, gewünschte Qualität und Lebenszykluskosten bringt. Unsere Empfehlung lautet dabei, sich auf jeden Fall an einem vorhandenen Standard zu orientieren und das Rad nicht neu zu erfinden. Hierdurch erhöht sich die Chance, dass neue Mitarbeiter den verwendeten Standard bereits kennen und ohne große Einarbeitungszeit Mehrwert für das Unternehmen generieren können.

Abschließend verweisen wir auf die offiziellen ABAP-Programmierrichtlinien, die uns bei der Definition der hier aufgeführten Namenskonventionen als Orientierungshilfe gedient haben:
http://help.sap.com/abapdocu_750/de/abennaming_guidl.htm

A.4 SONSTIGES / LESSONS LEARNED

A.4.1 KUNDENNAMENSRÄUME

Sollten Sie mehrere Systemlandschaften im Einsatz haben, zwischen denen Entwicklungen hin und her transportiert werden, empfiehlt sich die Einbindung der (hoffentlich eindeutigen) System-ID in den Präfixnamensraum. Hierdurch ersparen Sie sich evtl. auftretende Namenskonflikte zwischen den jeweiligen Systemen.

Beispiele für System ID = D01:

- YD01*
- /ZYXD01/*

A.4.2 VERMEIDUNG ÜBERFLÜSSIGER BEZEICHNERINFORMATIONEN

Vermeiden Sie überdimensionierte Konventionen für Organisationsbestandteile oder schnelllebige Informationen im knapp bemessenen Objektnamen. Nutzen Sie stattdessen Meta-Informationen die z.B. über das Classification Toolset[109] definiert werden können.

Anderenfalls entstehen schwer zu interpretierende, kryptische Bezeichner, deren Interpretation aufwendig ist und sich erst nach einem Langzeitstudium der Konventionen ergibt.

Beispiel:

- Name: /NAME/CL_T1_ERP_BC_SYS_DPC_607
- Bedeutung: Data Provider Klasse, die im Unternehmenstemplate T1 auf Basis von ECC 6 EHP 7 Systeminformationen bereitstellt.

Beispiele für die im Objektnamen zu vermeidenden Meta-Informationen sind:

- [RICEF-ID] / [Change Request ID] / [Ticket-ID]
- [Unternehmenstemplate] / [Systemtyp] / [Version]
- [Releasestand]*

109 *SAP-Help-Suche „Classification Toolset"*

Die Angabe des Release-Standes macht nur für die Produktentwicklung Sinn, die tatsächlich für verschiedene Release-Stände entwickelt wird und sich von der verfügbaren ABAP-Syntax oder dem Datenmodell voneinander unterscheidet. Die Verwendung sollte eher die Ausnahme als die Regel darstellen.

A.5 FORMULARE

Typ	Adobe Interactive Forms – Interfaces Formularschnittstellen sollten sich von den regulären Formularen durch Hinzufügen eines Suffixes hervorheben.
Namensbildung	[Formularname] [Interfacekürzel]
Beispiele	Y_CAS_SALES_ACTIVITY_IF

A.6 SCHUTZ VON NAMENSKONVENTIONEN IN DER ABAP-WORKBENCH

Namenskonventionen für Repository-Objekte lassen sich über die Bordmittel des Change and Transportsystems (CTS) verwalten. Hierzu stellt das CTS-Mechanismen für die Definition sogenannter Präfixkonventionen bereit. Die Konfiguration der Präfixkonventionen lässt sich durch die Tabellenpflegeviews V_TRESN, bzw. CTSRESNAME für reservierte Kundennamensräume, realisieren. Mit den Views ist es möglich, Namenskonventionen für Repository-Objekte pro Paket festzulegen[110].

Die Pflege einzelner Objekttypen pro Paket ist sehr zeitintensiv und mit entsprechend hohem Änderungsaufwand verbunden. Eine generische Pflege für alle Objekte eines einzelnen Pakets funktioniert nur bei reservierten Kundennamensräumen und ist deshalb nicht für alle Kunden relevant. Ein zusätzliches Manko besteht darin, dass die Konventionen keine „Überlappung" der Präfixe für verschiedene Pakete und deren enthaltenen Elemente zulassen. Es ist beispielsweise nicht möglich, das Präfix /ZYX/123 für das Paket /ZYX/SOME_PACKAGE und parallel das Präfix /ZYX/1234 für das Paket /ZYX/SOME_OTHER_PACKAGE einzusetzen, da das zweite Präfix eine Untermenge des ersten Präfix darstellt. Eine Gewährleistung, dass alle im Paket enthaltene Elemente mit dem definierten Präfix beginnen müssen, ist nicht sichergestellt, da Objekte ohne definiertes V_TRESN-Präfix problemlos dem Paket zugeordnet werden können.

Eine Neuerung für moderne Pakethierarchien wurde mit dem SAP-Hinweis 2297645 kurz vor Redaktionsschluss zur Verfügung gestellt und ist in den neusten, seit Mai 2016 verfügbaren Service Packs der Komponente SAP_BASIS 700 enthalten. Weitere Details dazu sind im SCN-Blog erläutert.[111] Zwar stellt die Neuerung sicher, dass Elemente mit dem definierten Präfix nur in einem Paket der Pakethierarchie abgelegt werden dürfen, sie behält aber die bereits aufgeführten Schwächen bei.

BEST PRACTICE

Auf Grund der eingeschränkten Funktionalität empfehlen wir die Verwendung der Tabellenpflegeviews V_TRESN, bzw. CTSRESNAME sorgfältig zu überdenken und eher nicht zu verwenden.

110 Vgl. Definition von Namenskonventionen (SAP-Bibliothek – Softwarelogistik)

111 SCN-Blog: „News about ABAP Package Concept: Naming Conventions for Package Hierarchies"

ANHANG B: WEITERFÜHRENDE BEISPIELE

B.1 ERWEITERUNG DER ABAP-SCHLÜSSELWORTDOKUMENTATION

Die individuellen Erweiterungsoptionen für die ABAP-Schlüsselwortdokumentation sind zum heutigen Stand kein offiziell unterstütztes Feature. Zurzeit existieren für die Erweiterung der Programmierrichtlinien nur sehr wenige Informationen, weshalb die nachfolgenden Hinweise auf eigenes Risiko hin verfolgt werden können:

Sie finden die Werkzeuge für die Pflege der Transaktion ABAPDOCU in dem Paket SABAPDOCU. Dort enthalten ist das Programm ABAP_DOCU_TREE_MAINTAIN, mit dessen Hilfe Sie die einzelnen Artikel in den Navigationsbaum einhängen und editieren können. Beim Speichern des Navigationsbaums durchläuft das Programm den gleichnamigen Funktionsbaustein ABAP_DOCU_TREE_MAINTAIN, in dessen lokaler Klasse TREE_MAINTAIN eine Methode EDIT_STORE_TO_DATABASE gerufen wird, die zu Beginn über eine Konstante auf ein zentrales SAP-Dokumentationssystem verprobt.

TREE_MAINTAIN ▸ EDIT_STORE_TO_DATABASE

```
864
865       " --- check if SAP system
866       IF cl_abap_docu_system=>id < cl_abap_docu_system=>sap.
867         MESSAGE text-nos TYPE 'S' DISPLAY LIKE 'W'.
868         RETURN.
869       ENDIF.
```

Am Ende der Methode wird die Speicherung des geänderten Navigationsbaums in einem Transportauftrag an die Methode UTIL_TRANSPORT_TREE delegiert, welche zu Beginn eine erneute Validierung des hart hinterlegten SAP-System durchführt.

TREE_MAINTAIN ▸ UTIL_TRANSPORT_TREE

```
1306
1307      IF cl_abap_docu_system=>id < cl_abap_docu_system=>home.
1308        MESSAGE text-tos TYPE 'S' DISPLAY LIKE 'W'.
1309        RETURN.
1310      ENDIF.
1311
```

Nach Umschiffung der beiden Proben können Sie den geänderten Navigationsbaum speichern und die Änderungen in andere Systeme transportieren.

IMPRESSUM

HINWEIS:

Wir weisen ausdrücklich darauf hin, dass das vorliegende Dokument nicht jeglichen Regelungsbedarf sämtlicher DSAG-Mitglieder in allen Geschäftsszenarien antizipieren und abdecken kann. Insofern müssen die angesprochenen Themen und Anregungen naturgemäß unvollständig bleiben. Die DSAG und die beteiligten Autoren können bezüglich der Vollständigkeit und Erfolgsgeeignetheit der Anregungen keine Verantwortung übernehmen. Sämtliche Überlegungen, Vorgehensweisen und Maßnahmen hinsichtlich des Verhaltens gegenüber SAP verbleiben in der individuellen Eigenverantwortung jedes DSAG-Mitglieds. Insbesondere kann dieser Leitfaden nur allgemeine Anhaltspunkte zu vertragsrechtlichen Themen geben und keinesfalls eine individuelle Rechtsberatung bei der Verhandlung und Gestaltung von Verträgen durch IT-rechtliche Experten ersetzen.

Deutschsprachige SAP® Anwendergruppe e.V.
Altrottstraße 34 a
69190 Walldorf | Deutschland
Telefon +49 6227 35809-58
Telefax +49 6227 35809-59
E-Mail info@dsag.de
www.dsag.de

WEITERE INFORMATIONEN:
Arbeitskreis Development, www.dsag.de/ak-development