Evaluation der Entwicklung von Programmen in der SAP Programmiersprache ABAP in den Programmierumgebungen Objekt Navigator im SAP Graphical User Interface im Vergleich zu Eclipse mit ABAP Development Tools

Hamza Yavuz (Matr. Nr: 3620816)

Seminar

RWTH Aachen Abteilung 5.2

Dezember 2025

Inhaltsverzeichnis

Abstract

Die digitale Transformation und die strategische Neuausrichtung der SAP auf S/4HANA führen in der ABAP-Entwicklung zu einem tiefgreifenden Paradigmenwechsel. Während der klassische Objekt Navigator (SE80) im SAP GUI über Jahrzehnte das dominierende Entwicklungswerkzeug war, treibt SAP zunehmend die Nutzung der modernen ABAP Development Tools (ADT) auf Eclipse-Basis voran. Die parallele Existenz beider Umgebungen, deren Funktionsumfang sich in vielen Bereichen überschneidet, erzeugt in der Praxis erhebliche Unsicherheiten hinsichtlich Werkzeugwahl, Effizienz und strategischer Ausrichtung der Entwicklungslandschaft.

Ziel dieser Seminararbeit ist die systematische Evaluation und der kritische Vergleich beider Programmierumgebungen. Untersucht wird, inwiefern sich SE80 und ADT hinsichtlich Funktionalität, Entwicklungseffizienz, Usability und Zukunftsfähigkeit unterscheiden und welche Implikationen diese Unterschiede für den ABAP-Entwicklungsprozess haben.

Methodisch erfolgt eine kriterienbasierte Analyse, die auf einer fundierten Darstellung der technischen Grundlagen und der jeweiligen Systemarchitektur aufbaut. Als zentrale Evaluationskriterien werden Entwicklungseffizienz, Benutzerfreundlichkeit, Refactoring-Unterstützung sowie die Abdeckung moderner ABAP-Syntax und -Entwicklungsparadigmen definiert. Anhand dieser Kriterien werden die Stärken und Schwächen beider Werkzeuge entlang des gesamten Entwicklungsprozesses, von der Code-Erstellung über Strukturierungs- und Anpassungsmaßnahmen bis hin zum Debugging, systematisch herausgearbeitet.

Die Arbeit schafft damit eine analytische Grundlage, die es Entwicklern und IT-Verantwortlichen ermöglicht, den Einsatz der beiden Entwicklungsumgebungen differenziert zu bewerten und auf dieser Basis eine strategisch fundierte Entscheidung über die zukünftige Ausrichtung ihrer ABAP-Entwicklungslandschaft zu treffen.

1 Einleitung

1.1 Problemstellung und Motivation

Die digitale Transformation und die strategische Neuausrichtung der SAP SE auf S/4HANA und Cloud-Technologien führen in der ABAP-Entwicklung zu einem tiefgreifenden technologischen Wandel. Über Jahrzehnte hinweg bildete der Objekt Navigator (Transaktion SE80) im SAP GUI den Standard für die Entwicklung von Geschäftsanwendungen. Dieses monolithische, serverzentrierte Werkzeug prägte die Arbeitsweise ganzer Entwicklergenerationen.

Mit der Einführung der ABAP Development Tools (ADT) auf Basis der offenen Eclipse-Plattform etablierte SAP jedoch eine konkurrierende Umgebung, die moderne Entwicklungsparadigmen adressiert. In der betrieblichen Praxis führt die parallele Existenz beider Werkzeuge häufig zu Unsicherheiten: Es fehlt oft an fundierten Entscheidungsgrundlagen, wann welches Werkzeug effizienzsteigernd eingesetzt werden kann und ob der Migrationsaufwand zu Eclipse ökonomisch gerechtfertigt ist.

1.2 Zielsetzung der Arbeit

Ziel dieser Seminararbeit ist die systematische Evaluation und der kritische Vergleich der beiden Entwicklungsumgebungen SE80 und Eclipse ADT. Es soll untersucht werden, inwiefern sich die Werkzeuge hinsichtlich ihrer Systemarchitektur, Funktionalität und Effizienz unterscheiden.

Im Fokus steht dabei die Beantwortung der Forschungsfrage, ob die ADT lediglich eine moderne Alternative darstellen oder eine notwendige Voraussetzung für die Entwicklung im S/4HANA-Kontext bilden. Die Arbeit dient somit als Entscheidungsgrundlage für die strategische Ausrichtung von Entwicklungsprozessen.

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in einen theoretischen und einen empirisch-analytischen Teil. Kapitel 2 legt die theoretischen Grundlagen und analysiert die unterschiedlichen Systemarchitekturen (Server-Side vs. Client-Side Rendering). Kapitel 3 definiert die Methodik und die Evaluationskriterien. Darauf aufbauend erfolgt in Kapitel 4 die detaillierte vergleichende Analyse anhand konkreter Anwendungsfälle („Use Cases“) aus der Entwicklungspraxis. Die Arbeit schließt in Kapitel 5 und 6 mit einer Diskussion der Ergebnisse und einem Ausblick auf die zukünftige Relevanz der Entwicklungsumgebungen.

2 Theoretische Grundlagen und technischer Kontext

2.1 Historische Evolution der ABAP-Entwicklungsumgebungen

Im Zuge der strategischen Neuausrichtung von SAP hin zu S/4HANA und Cloud-basierten Technologien geriet das klassische Entwicklungsmodell auf Basis der SAP GUI zunehmend an seine Grenzen. Die bisherigen Werkzeuge waren stark auf traditionelle Entwicklungsprozesse ausgelegt und boten nur eingeschränkte Unterstützung für moderne Anforderungen wie bessere Toolintegration, höhere Entwicklungsproduktivität oder zeitgemäße Test- und Qualitätskonzepte.

Vor diesem Hintergrund entschied sich SAP, die ABAP-Entwicklung auf eine neue technische Grundlage zu stellen und führte die ABAP Development Tools (ADT) auf Basis der Eclipse-Plattform ein. Ziel war es, eine offenere und erweiterbare Entwicklungsumgebung bereitzustellen, die sich besser in moderne Entwicklungsprozesse einfügt und gleichzeitig die bestehenden ABAP-Konzepte weiterhin unterstützt. Dieser Schritt kann als ein grundlegender Wandel weg von der rein serverzentrierten Entwicklungsweise hin zu einer stärker clientbasierten Arbeitsweise verstanden werden, bei der ein Großteil der Entwicklungsarbeit lokal in der IDE erfolgt, während die Anbindung an das SAP-System erhalten bleibt (vgl. Hardy, 2021, Kap. 1).

2.2 Der Objekt Navigator (SE80) im SAP GUI

Einen visuellen Eindruck der klassischen Arbeitsweise vermittelt Abbildung 1. Der Objekt Navigator (Transaktion SE80) stellt sich als monolithische Entwicklungsumgebung innerhalb des SAP GUI dar. Das zentrale Steuerungselement bildet hierbei der Repository Browser am linken Bildschirmrand. Dieser erlaubt die Navigation durch die hierarchische Baumstruktur der Entwicklungsobjekte.

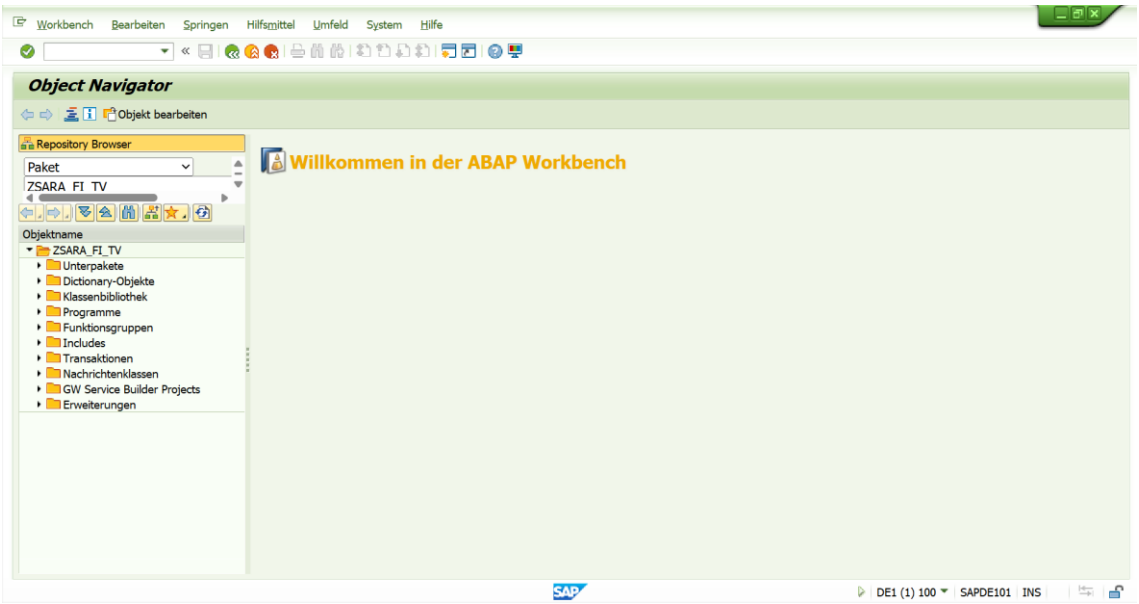

Abbildung 1: Der Objekt Navigator (Transaktion SE80) mit dem hierarchischen Repository Browser und der integrierten Werkzeugleiste im SAP GUI.

Architektonisch basiert die SE80 auf dem klassischen Multiple Document Interface (MDI). Konkret impliziert dies, dass sämtliche Werkzeuge vom Editor über den Screen Painter bis zur Tabellendefinition in einem einzigen Fensterrahmen eingebettet sind. Charakteristisch für diese Architektur ist die enge Kopplung an den SAP Applikationsserver. Technisch wird die Darstellung als Server-Side Rendering klassifiziert. Zwar gewährleistet dieser Ansatz eine hohe Datenkonsistenz, führt in der Praxis jedoch häufig zu Latenzen, da jeder Arbeitsschritt die Antwort des Servers abwarten muss (vgl. Lordieck/Sprenger, 2024, Kap. 2)

2.3 Die Eclipse-Plattform und ABAP Development Tools (ADT)

Demgegenüber verfolgen die ABAP Development Tools (ADT) einen architektonisch differenzierten Ansatz. Anstatt auf einer monolithischen Server-Struktur zu verharren, setzen sie auf der offenen, lokal installierten Eclipse-Plattform auf. Abbildung 2 illustriert, wie sich die Darstellung von der starren Ein-Fenster-Logik der SE80 löst. Eclipse organisiert die Arbeitsfläche stattdessen modular in sogenannten Perspektiven und Views (vgl. Lordieck/Sprenger, 2024, Kap. 1).

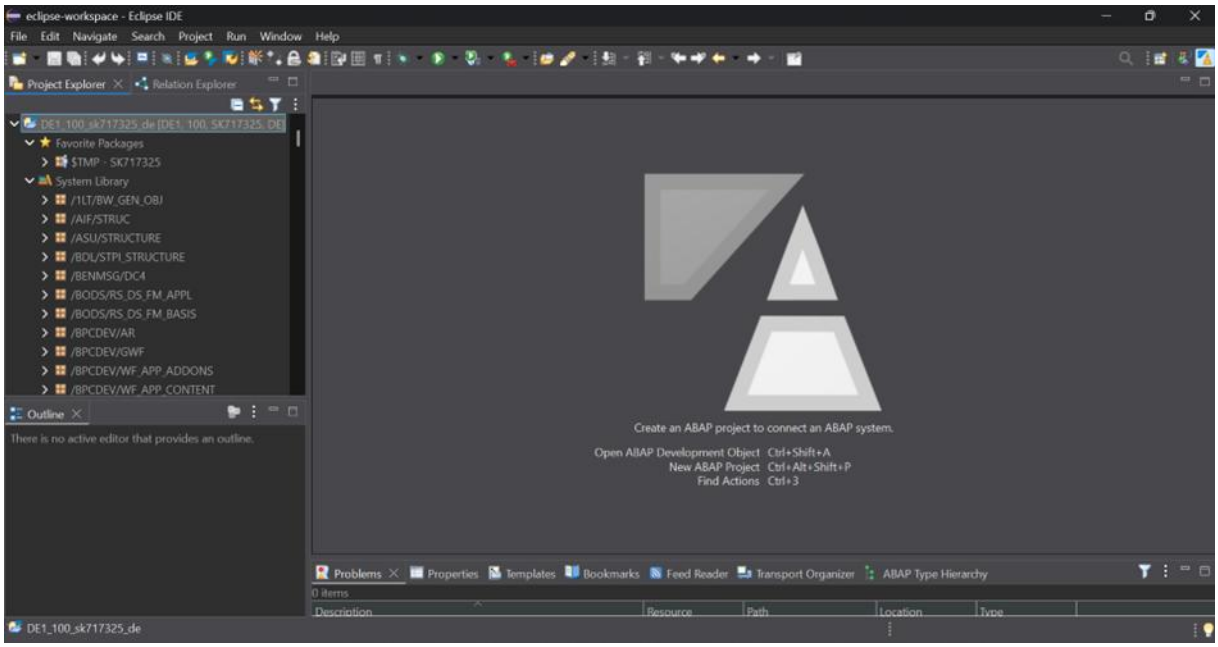

Abbildung 2: Die ABAP Development Tools in Eclipse mit dem Project Explorer zur Navigation in der Paketstruktur und dem zentralen Editor-Bereich.

Der Screenshot verdeutlicht dies am Beispiel der Standard-ABAP-Perspektive: Der links angeordnete Project Explorer übernimmt die Funktion des Navigators. Im Kontrast zur SE80 erlaubt dieser nicht nur die Sicht auf ein isoliertes System, sondern ermöglicht die parallele Verwaltung mehrerer ABAP-Projekte. Zwar bleibt die hierarchische Strukturierung erhalten, jedoch werden Inhalte hier bedarfsgerecht vom Server nachgeladen ("Lazy Loading").

Ein wesentlicher Unterschied ist das parallele Arbeiten an mehreren Objekten. Während die SE80 oft durch das Modus-Konzept limitiert ist, ermöglichen die ADT das gleichzeitige Bearbeiten beliebig vieler Artefakte in Tabs, was die Effizienz bei komplexen Aufgabenstellungen signifikant erhöht (vgl. Pęgiel, 2021, Kap. 4).

Gleichzeitig ist zu berücksichtigen, dass die höhere Funktionsdichte und die Komplexität der Eclipse-Plattform insbesondere für langjährige SAP-GUI-Nutzer eine erhöhte Einstiegshürde darstellen können und anfänglich zu Produktivitätseinbußen führen (vgl. Pęgiel, 2021, Kap. 1).

2.4 Gegenüberstellung der Systemarchitekturen

Die Architektur der ADT unterscheidet sich somit fundamental von der SAP GUI basierten Entwicklung. Eclipse fungiert als vollwertige Integrierte Entwicklungsumgebung (IDE). Ein Vorteil der Client-basierten Architektur liegt in der Unterstützung moderner Entwicklungsparadigmen. Komplexere Artefakte wie Core Data Services (CDS) lassen sich aufgrund ihrer Syntax-Anforderungen nur in einer solchen modernen IDE effizient handhaben (vgl. Hardy, 2015, Kap. 15.2.1).

Zusammenfassend stellt die folgende Tabelle die architektonischen und funktionalen Unterschiede gegenüber:

Merkmal	**Objekt Navigator (SE80)**	**Eclipse ADT**
Architektur	Server-basiert / Terminal-Emulation	Client-basiert / Lokale IDE (vgl. Lordieck/Sprenger, 2024, Kap. 1)
Kommunikation	DIAG-Protokoll (Synchron / Hohe Latenz)	RFC / REST (Ressourcen-Ladung bei Bedarf)
Benutzeroberfläche	Monolithisches MDI (Ein Fenster)	Flexibel (Views, Perspektiven, Multi-Monitor) (vgl. Pęgiel, 2021, Kap. 4)

Merkmal	Objekt Navigator (SE80)	Eclipse ADT
Rendering	Server-Side Rendering (Pixel/Layout vom Server)	Client-Side Rendering (Nur Datenübertragung)
Offline-Lesbarkeit	Nein (Verbindungsabbruch = Datenverlust)	Code im Cache lesbar

3 Methodik der Evaluation

3.1 Vorgehensweise und Kriterienauswahl

Die vorliegende Arbeit verfolgt einen qualitativen, analytischen Ansatz. Zur objektiven Gegenüberstellung der Entwicklungsumgebungen SE80 und Eclipse ADT wird eine kriteriengeleitete Untersuchung angewandt. Die Methodik gliedert sich in zwei Phasen:

1. **Literaturanalyse:** Aufarbeitung der theoretischen Grundlagen (siehe Kapitel 2).

2. **Empirischer Funktionsvergleich:** Gegenüberstellung beider Werkzeuge anhand konkreter Anwendungsfälle ("Use Cases") in Kapitel 4.

3.2 Definition der Evaluationskriterien

Kriterium	Definition und Fragestellung
Entwicklungseffizienz	Wie zeiteffizient lassen sich Standardaufgaben erledigen? Bewertet werden Features wie Code-Vervollständigung und Templates.
Usability (Gebrauchstauglichkeit)	Wie intuitiv gestaltet sich die Benutzeroberfläche? Betrachtet werden Navigation und Fenster-Management.
Funktionalität & Refactoring	Welche Werkzeuge stehen zur Qualitätssteigerung bereit? Fokus auf Refactoring und Debugging.
Zukunftsfähigkeit (Modern ABAP)	Inwieweit unterstützt das Werkzeug moderne Paradigmen wie S/4HANA und CDS?

4 Vergleichende Analyse der Entwicklungsumgebungen

4.1 Code-Erstellung und Editor-Funktionen

Besonders evident wird die Diskrepanz im täglichen Arbeiten bei der Erstellung von Quellcode. In der SE80 ist der Entwicklungsprozess oft durch assistentengestützte Dialoge ("Wizards") geprägt, da der Editor selbst nur geringe kontextsensitive Hilfe bietet (vgl. Lordieck/Sprenger, 2024, Kap. 4.2).

Bei der Implementierung eines Funktionsbausteins erfolgt der Aufruf in der SE80 entweder durch manuelle Eingabe der benötigten Parameter oder mithilfe der Funktion „Muster einfügen“ (Strg+F6). Wie Abbildung 3 zeigt, öffnet sich dabei ein separates Dialogfenster, das den Aufruf des Funktionsbausteins als Code-Block generiert. Erst

nach Bestätigung generiert das System den Code-Block. Der Nachteil: Nachträgliche Änderungen (z. B. ein neuer Parameter) werden nicht automatisch erkannt und das Muster muss oft neu eingefügt werden. Zwar erfordern auch in den ADT nachträgliche Schnittstellenänderungen eine Anpassung der Aufrufstellen, diese wird dort jedoch durch kontextsensitive Hinweise und automatische Quick Fixes wesentlich unterstützt.

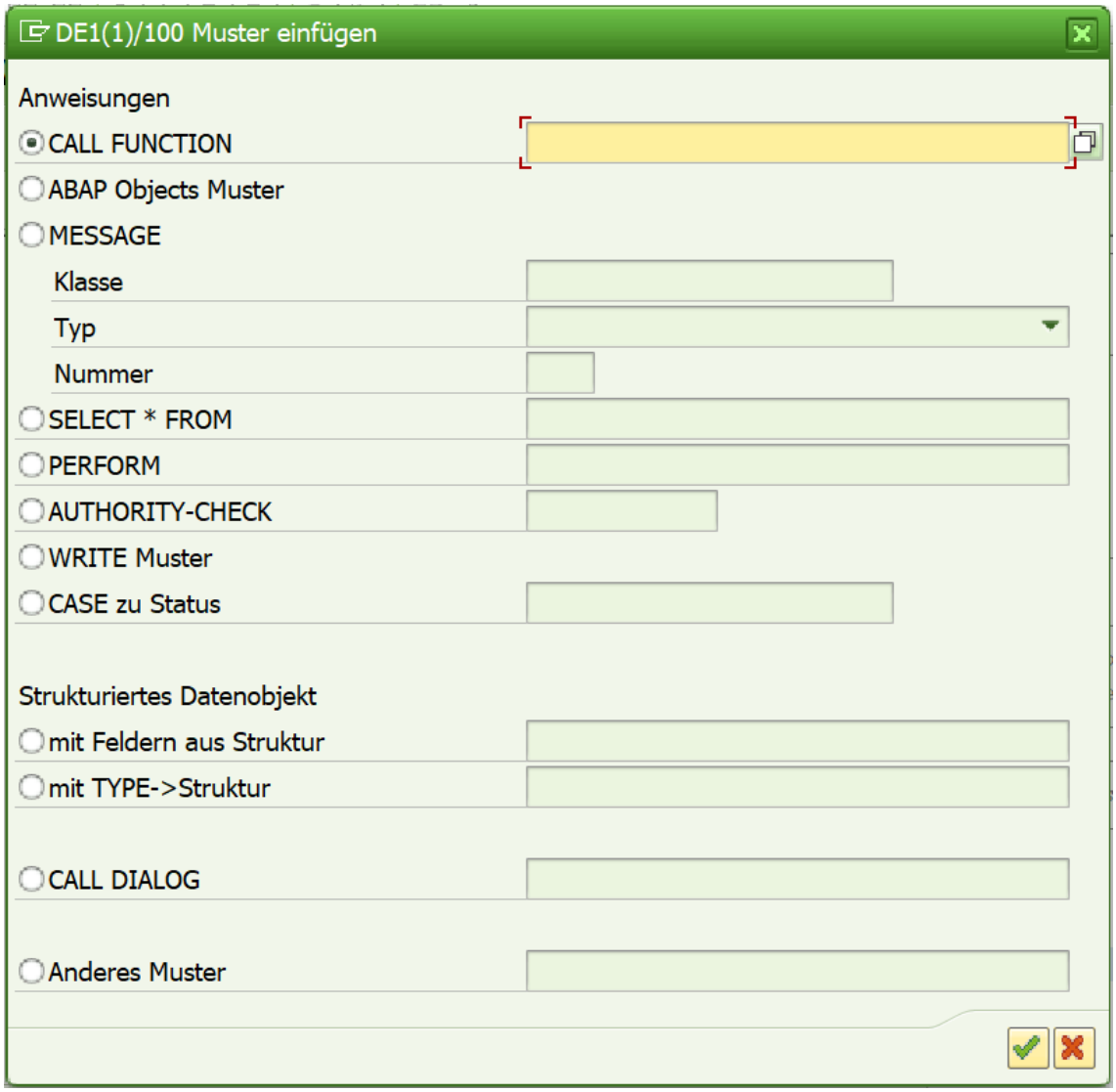

Abbildung 3: Der statische Dialog zur Muster-Einfügung (Pattern Wizard) für Funktionsbausteine in der SE80.

Im Gegensatz dazu bieten die ADT in Eclipse eine intelligente Code-Vervollständigung (Content Assist), aufrufbar über Strg+Leertaste. Abbildung 4 demonstriert dies am Beispiel der Klasse cl_salv_table. Eclipse analysiert den Kontext direkt an der Cursor-Position und schlägt nicht nur den Methodennamen vor, sondern visualisiert in einem Overlay sofort die komplette Signatur inklusive aller Import- und Export-Parameter (wie „r_salv_table“).

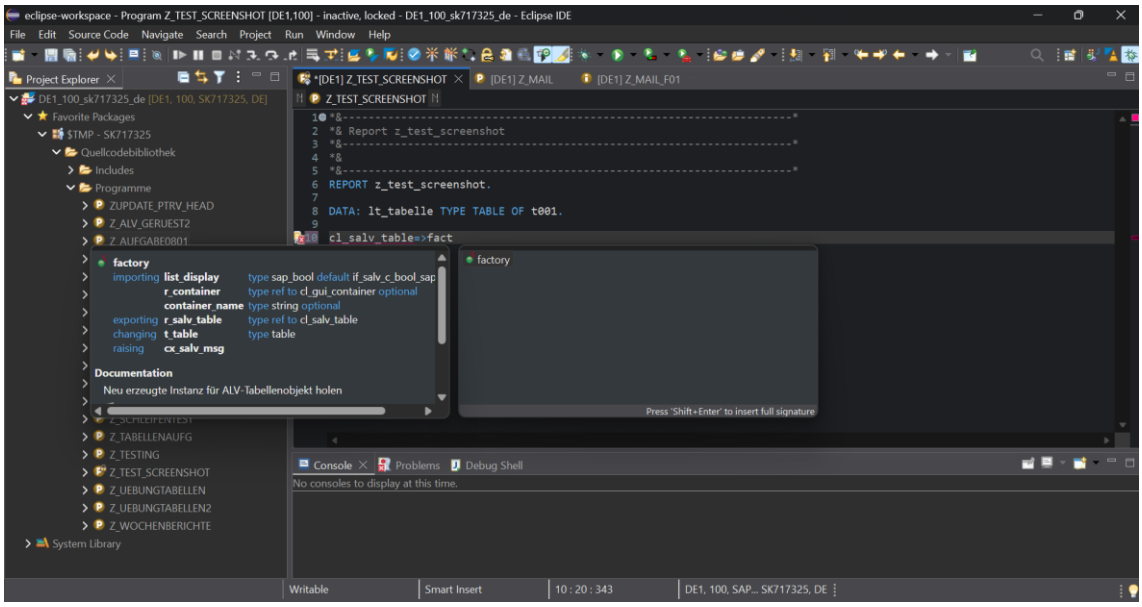

Abbildung 4: Die kontextsensitive Code-Vervollständigung (Content Assist) in den ADT mit Anzeige von Methodensignaturen.

Wie im Screenshot ersichtlich, weist die IDE kontextsensitiv auf Funktionen wie die vollständige Signatur-Einfügung (Shift + Enter) hin. Im Unterschied zur SE80, deren Code-Vervollständigung (Strg + Leertaste) überwiegend statisch bleibt oder über separate Dialoge („Muster einfügen“) erfolgt, ist diese Funktionalität in den ADT direkt in den Editor-Kontext integriert. Dadurch wird der Schreibfluss weniger unterbrochen und der Bedarf an expliziten Kontextwechseln reduziert (vgl. Lordieck/Sprenger, 2024, Kap. 4.2).

4.2 Navigation und Suche im Repository

Die Navigation innerhalb der SE80 ist primär hierarchisch organisiert und stark an die Paketstruktur gebunden. Zwar stehen auch Suchfunktionen zur Verfügung, diese sind jedoch häufig auf bestimmte Objekttypen beschränkt oder erfordern den Wechsel in separate Dialoge. Eclipse verfolgt demgegenüber einen konsequent suchbasierten Ansatz: Mit der Funktion „Open ABAP Development Object“ (Strg + Shift + A), dargestellt in Abbildung 5, können Entwickler systemweit und objekttypübergreifend nach Entwicklungsartefakten suchen.

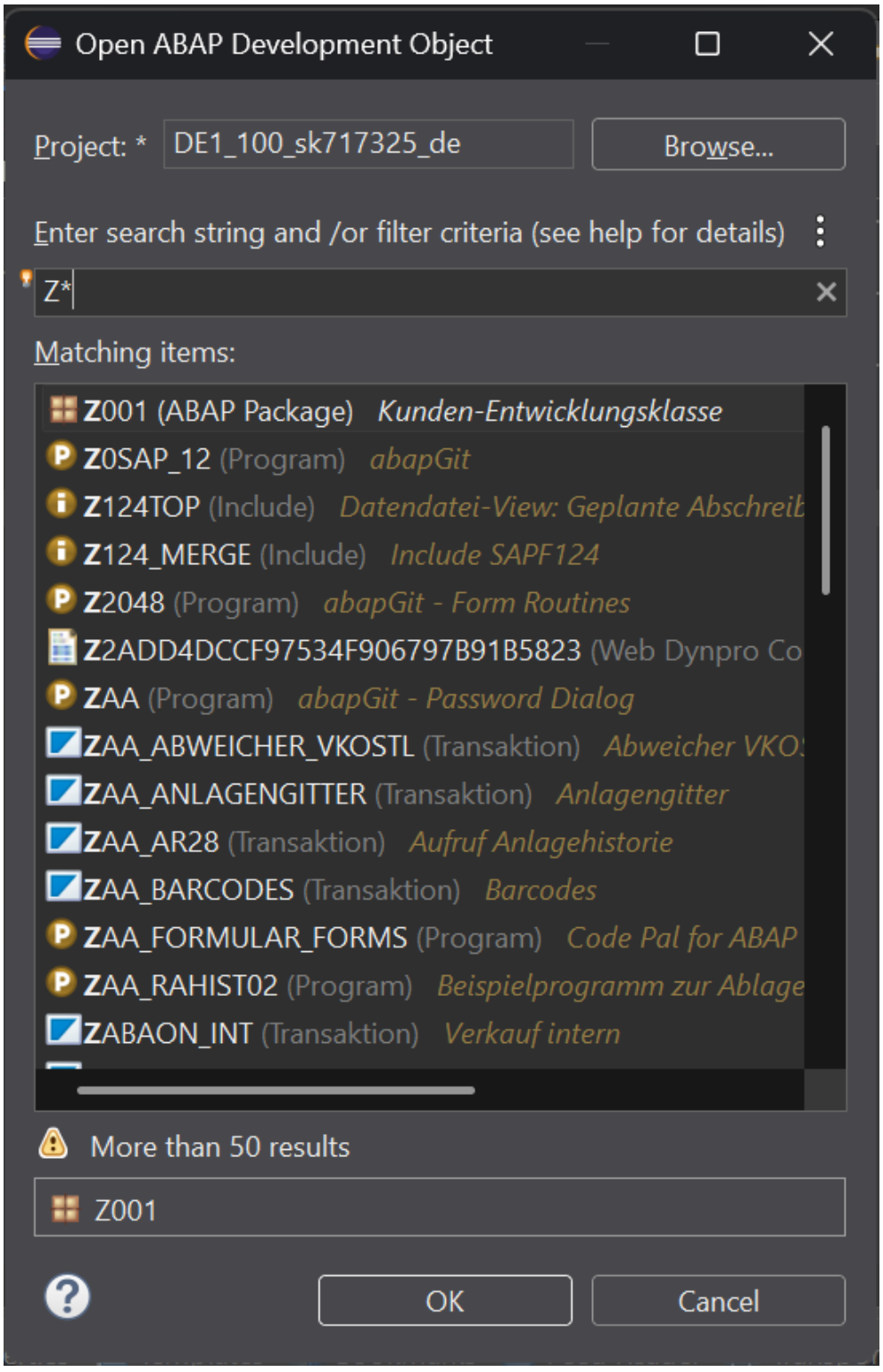

Abbildung 5: Der zentrale Suchdialog "Open ABAP Development Object" (Strg+Shift+A) zur systemweiten Suche in Eclipse.

Ein entscheidender Vorteil liegt in der Unterstützung von Wildcards und der CamelCase-Suche. Dies beschleunigt den Zugriff auf Entwicklungsobjekte signifikant (vgl. Lordieck/Sprenger, 2024, Kap. 4.1).

Um die Diskrepanz in der Bedienungsgeschwindigkeit zu quantifizieren, lohnt sich ein Blick auf die Tastenkürzel (Shortcuts). Während die SE80 stark mausorientiert ist, erlauben die ADT eine fast vollständige Steuerung über die Tastatur.

Funktion	**SAP GUI / SE80**	**Eclipse ADT**	**Kommentar**
Objekt öffnen	Manuelle Baum-Navigation oder SE24/SE38	Strg + Shift + A	Eclipse-Suche ist objektübergreifend
Aktivieren	Strg + F3	Strg + F3	Identisch (Standard SAP)
Code-Vervollständigung	Strg + F6 (Muster einfügen) oder Strg + Leertaste	Strg + Leertaste, anschließend Shift + Enter	Eclipse ist kontextsensitiv
Element umbenennen	Nicht vorhanden (Manuell)	Alt + Shift + R	Refactoring über alle Verwendungsstellen
Methode extrahieren	Nicht vorhanden	Alt + Shift + M	Automatische Code-Kapselung

Funktion	SAP GUI / SE80	Eclipse ADT	Kommentar
Definition anzeigen	Doppelklick (Navigation)	F3 oder F2	F2 zeigt Details im Overlay ohne Navigation
Fenster maximieren	Nicht möglich (MDI)	Strg + M	Fokus auf den Code-Editor
Zeile duplizieren	Strg + D	Strg + Alt + Unten	Standard-IDE-Verhalten

Diese Tabelle verdeutlicht, dass Eclipse ADT etablierte Standards moderner IDEs (wie Visual Studio oder IntelliJ) adaptiert, was insbesondere jüngeren Entwicklern den Einstieg erleichtert.

4.3 Unterstützung von Refactoring-Maßnahmen

Defizite der SE80 offenbaren sich insbesondere im Bereich des Refactorings. In der klassischen Umgebung müssen Aufgaben wie das Umbenennen von Methoden manuell durchgeführt werden, was fehleranfällig ist.

Die ADT stellen hierfür automatisierte Werkzeuge bereit. Wie das Kontextmenü in Abbildung 6 demonstriert, bietet Eclipse Funktionen wie "Rename" (Alt+Shift+R) oder "Extract Method" (Alt+Shift+M) an (vgl. Hardy, 2021, Kap. 1.2.4).

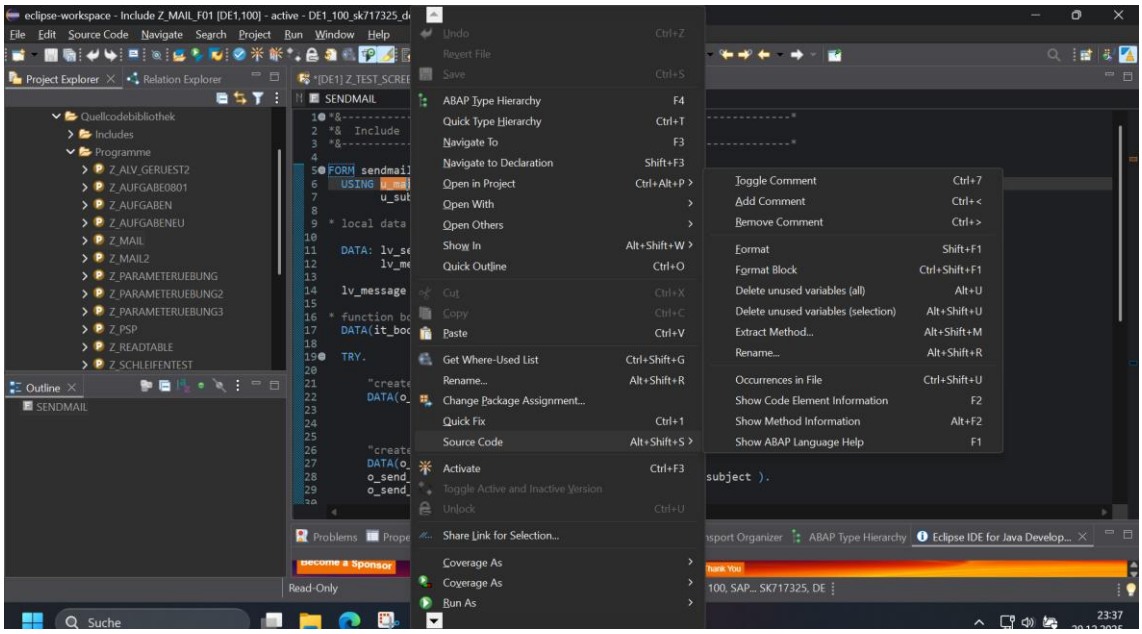

Abbildung 6: Das Kontextmenü für Source-Code-Operationen in Eclipse, inklusive automatisierter Refactoring-Funktionen wie "Extract Method".

Bei Selektion eines Code-Blocks und Ausführung von "Extract Method" analysiert Eclipse den Datenfluss und generiert automatisch eine neue Methode mit korrekten Parametern. Dies senkt die Hemmschwelle für qualitätssichernde Maßnahmen (vgl. Hardy, 2021, Kap. 1.2.4). Gleichzeitig erfordert der Einsatz automatisierter Refactoring-Werkzeuge ein hohes Vertrauen in die IDE, da insbesondere bei komplexen Legacy-Strukturen eine nachträgliche manuelle Überprüfung der Änderungen weiterhin notwendig bleibt.

Ein wesentliches Differenzierungsmerkmal zwischen der SE80 und den ABAP Development Tools (ADT) in Eclipse ist die Unterstützung durch sogenannte "Quick Fixes" (in Eclipse über das Tastenkürzel Strg + 1 erreichbar). Während Refactoring in der SE80 oft manuelle Eingriffe in verschiedenen Transaktionen erfordert, ermöglichen die ADT einen assistierten Ansatz direkt im Quellcode.

Ein exemplarisches Szenario für die Effizienzsteigerung ist die "Usage-First"-Entwicklungsmethodik. Hierbei schreibt der Entwickler zunächst den Aufruf einer Methode, die noch gar nicht existiert. In der SE80 würde dies zu einem Syntaxfehler führen, der den Entwickler zwingt, den Editor zu verlassen, in die Klassendefinition (SE24) zu wechseln, die Methode samt Signatur manuell anzulegen und anschließend zum Aufruf zurückzukehren. Hardy beschreibt diesen Prozess in Eclipse als signifikant effizienter: Durch die Anwendung eines Quick Fixes auf den fehlerhaften Methodenaufruf generiert die IDE automatisch die Methodendefinition und -

implementierung, wobei die Typisierung der Parameter direkt aus dem Kontext des Aufrufs abgeleitet wird (vgl. Hardy, 2021, Kap. 1.2.3). Dieser Automatismus reduziert nicht nur den manuellen Schreibaufwand, sondern minimiert auch den Kontextwechsel ("Context Switch"), was den kognitiven "Flow" des Entwicklers aufrechterhält.

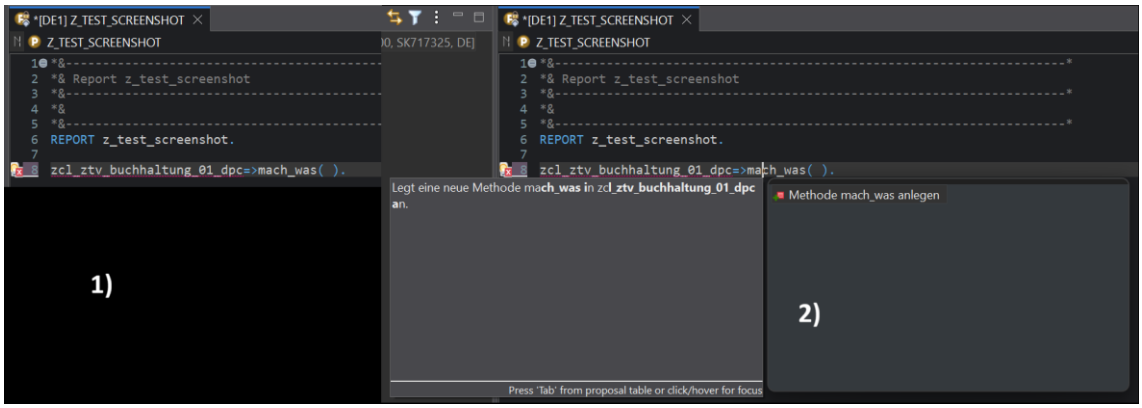

Abbildung 7: Automatisierte Methodengenerierung durch Quick Fixes in Eclipse ADT.

Die praktische Anwendung dieses Features wird in Abbildung 7 veranschaulicht. Das Beispiel zeigt einen Methodenaufruf, der im Code implementiert wurde, bevor die Methode technisch existierte (Usage-First-Ansatz). Während dies in der SE80 einen manuellen Abbruch des Arbeitsflusses erfordern würde, bietet Eclipse ADT über das Quick-Fix-Menü (Strg + 1) kontextsensitiv die automatische Generierung der fehlenden Methode an (siehe rechter Bildteil). Wie der Screenshot verdeutlicht, erkennt die IDE dabei automatisch die benötigten Parameter und Signaturen, wodurch fehleranfällige manuelle Eingaben entfallen.

4.4 Analyse- und Debugging- Werkzeuge

Ein kritischer Erfolgsfaktor in der Softwareentwicklung ist die effiziente Fehleranalyse. In der SE80 wird der klassische ABAP Debugger als eigenständiges Fenster (separater Modus) ausgeführt. Wie Abbildung 7 zeigt, überlagert dieses Fenster oft den eigentlichen Editor, was den visuellen Kontext des Entwicklers unterbricht.

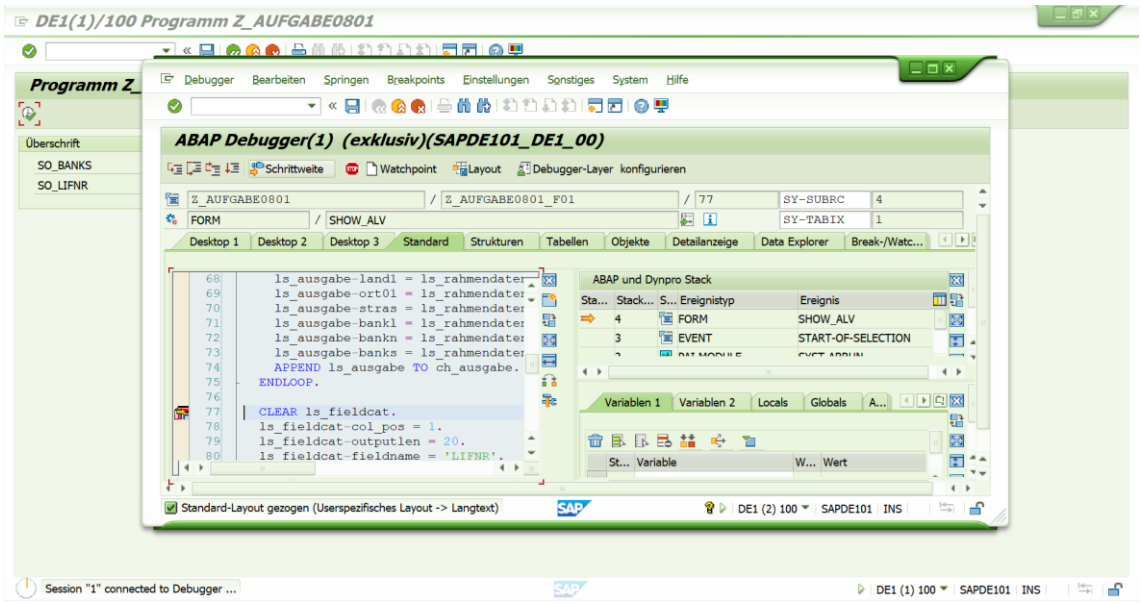

Abbildung 8: Der klassische ABAP Debugger in einem separaten Fenster, losgelöst von der Entwicklungsumgebung.

Die ADT integrieren den Debugging-Prozess hingegen nahtlos in die IDE. Durch die spezielle "Debug-Perspektive" bleibt der Quellcode im Editor sichtbar, während Variablen und Breakpoints in dedizierten Views angeordnet werden (vgl. Lordieck/Sprenger, 2024, Kap. 6.3).

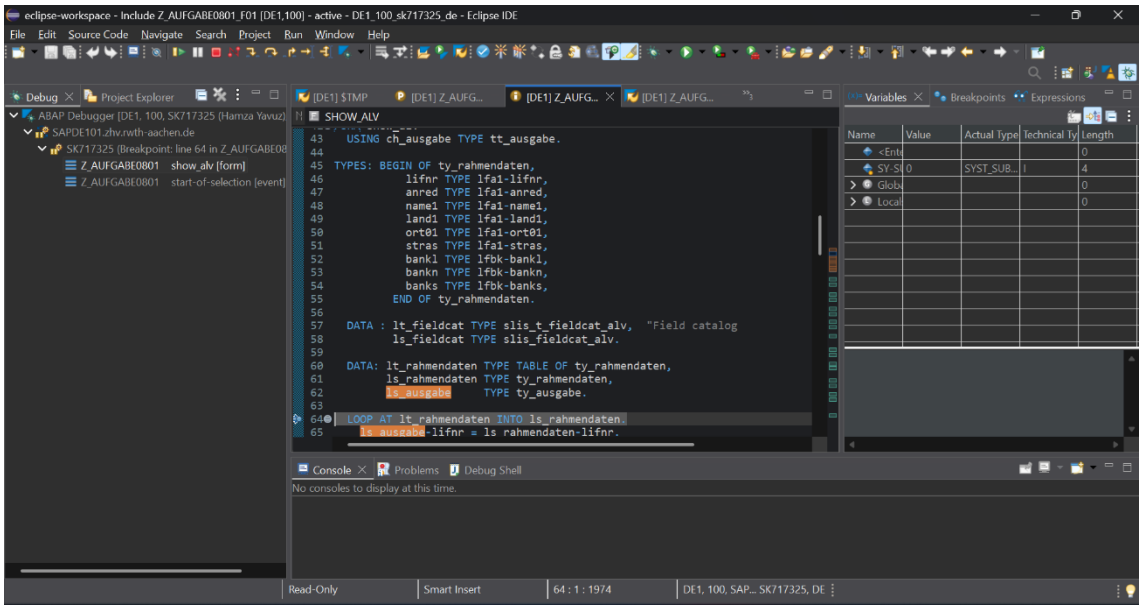

Abbildung 9: Die integrierte Debug-Perspektive in Eclipse: Quellcode und Variablen-Ansicht in einer gemeinsamen Entwicklungsumgebung.

Es ist jedoch anzumerken, dass der klassische SAP-GUI-Debugger bei bestimmten Analyseaufgaben weiterhin klare Vorteile bietet. Dies zeigt sich insbesondere bei dialogorientierten Programmen sowie bei der Untersuchung komplexer Kontrollflüsse. Wie in Abbildung 8 dargestellt, stellt der SAP-GUI-Debugger den vollständigen ABAP- und Dynpro-Aufrufstack dauerhaft und übersichtlich dar. Ereignisse wie START-OF-SELECTION, FORM-Aufrufe oder Dynpro-Events sind klar voneinander getrennt sichtbar und erlauben eine schnelle zeitliche Einordnung des Programmablaufs.

Gerade bei tief verschachtelten Funktionsaufrufen, der Analyse von Includes oder bei der Fehlersuche in klassischen Dynpro-Anwendungen erleichtert diese permanente Stack-Darstellung das Verständnis des Kontrollflusses erheblich. Auch systemnahe Aspekte wie Update-Tasks, RFC-Kontexte oder Benutzerinteraktionen lassen sich im SAP-GUI-Debugger häufig transparenter nachvollziehen.

Die Debug-Perspektive der ABAP Development Tools integriert den Debugging-Prozess zwar nahtlos in die Entwicklungsumgebung und ist für den täglichen, codezentrierten Entwicklungsprozess komfortabler. Bei spezialisierten Analyseaufgaben erreicht sie jedoch nicht in allen Fällen die gleiche Übersichtlichkeit und Detailtiefe wie der klassische SAP-GUI-Debugger (vgl. Lordieck/Sprenger, 2024, Kap. 6.3).

Ein technologisches Alleinstellungsmerkmal von Eclipse ist zudem die ABAP Profiling Integration. Während Laufzeitanalysen (Transaktion SAT) in der SE80 oft mühsam konfiguriert werden müssen, erlaubt Eclipse den direkten Start eines Profiling-Laufs aus dem Editor heraus. Es werden hierbei die visuellen Auswertungsmöglichkeiten betont, die es erlauben, Performance-Flaschenhälse ("Hotspots") grafisch im Code-Fluss zu identifizieren, ohne tief in technische Trace-Dateien einsteigen zu müssen (vgl. Lordieck/Sprenger, 2024, Kap. 6.3).

4.5 Unterstützung moderner ABAP-Konzepte (S/4HANA)

Neben der Effizienz ist die Wahl der Entwicklungsumgebung heute eine Frage der technischen Machbarkeit im Kontext von S/4HANA. Mit der Einführung moderner Paradigmen verschiebt sich die Anwendungslogik zunehmend auf die Datenbank ("Code-to-Data").

Zentrales Element hierfür sind die Core Data Services (CDS). Hardy (2021) klassifiziert diese als „Datenbank-Views der nächsten Generation“, welche die funktionalen Möglichkeiten klassischer SE11-Views weit übertreffen. Da CDS-Views mittels einer eigenen DDL-Syntax definiert werden, ist deren Bearbeitung in der SE80 technisch nicht mehr möglich. Für die Umsetzung moderner Anforderungen ist der Einsatz der ABAP Development Tools in Eclipse somit faktisch alternativlos (vgl. Hardy, 2015, Kap. 15.2.1).

4.6 Automatisierte Qualitätssicherung und Test-Driven Development (TDD)

Die Qualitätssicherung in ABAP-Projekten umfasst heute weit mehr als die reine Fehlerbehebung im laufenden System. Neben statischer Codeanalyse (z. B. über das ABAP Test Cockpit) sind automatisierte Unit-Tests und ein strukturierter Refactoring-Prozess entscheidend, um Änderungen risikoarm und nachvollziehbar umzusetzen (vgl. Hardy, 2021, Kap. 10).

Test-Driven Development (TDD) in ABAP folgt dem Prinzip „Red-Green-Refactor“: Zunächst wird ein Test formuliert, der erwartetes Verhalten beschreibt und (noch) fehlschlägt; danach wird die minimal notwendige Implementierung erstellt, bis der Test grün ist; abschließend wird der Code refaktoriert, ohne die Fachlogik zu verändern. In ABAP wird dieses Vorgehen typischerweise mit ABAP Unit umgesetzt, wobei Testklassen als lokale Klassen mit dem Zusatz FOR TESTING definiert werden und Assertions über CL_ABAP_UNIT_ASSERT erfolgen (vgl. Hardy, 2015, Kap. 15.2.1).

```
CLASS ltc_calculator DEFINITION FOR TESTING
  DURATION SHORT
  RISK LEVEL HARMLESS.

  PRIVATE SECTION.
    DATA: mo_cut TYPE REF TO zcl_calculator. " Class Under Test

    METHODS: setup.
    METHODS: test_addition FOR TESTING.
ENDCLASS.

CLASS ltc_calculator IMPLEMENTATION.
  METHOD setup.
    " Wird vor jedem Test ausgeführt
    mo_cut = NEW #( ).
  ENDMETHOD.

  METHOD test_addition.
    " GIVEN: Zwei Zahlen
    DATA(lv_sum) = mo_cut->add( i_a = 5 i_b = 3 ).

    " THEN: Erwarte Ergebnis 8
    cl_abap_unit_assert=>assert_equals(
      act = lv_sum
      exp = 8
      msg = 'Addition fehlgeschlagen'
    ).
  ENDMETHOD.
ENDCLASS.
```

Listing 1: Beispiel einer ABAP-Unit-Testklasse (TDD) mit setup und assert_equals

Listing 1 zeigt eine kompakte ABAP-Unit-Testklasse für eine Rechner-Klasse („Class Under Test“). Der Aufbau mit setup (Initialisierung vor jedem Test) und einer Testmethode, die über assert_equals ein erwartetes Ergebnis prüft, entspricht dem üblichen Muster für Einstiegsbeispiele in ABAP Unit (vgl. Hardy, 2015, Kap. 15.2.1).

Unterstützung durch ADT im Vergleich zur ABAP Workbench (SE80): Während sich Qualitätssicherungs-Transaktionen in der ABAP Workbench eher als getrennte Werkzeuge anfühlen, bindet ADT viele Funktionen direkt in den Editor-Kontext ein. Dazu zählen insbesondere Quick Fixes/Quick Assist (z. B. Strg+1) für Refactorings und Code-Erzeugung sowie die Möglichkeit, Prüfergebnisse und Fundstellen über Navigations- und Link-Funktionen schnell anzuspringen (vgl. Lordieck/Sprenger, 2024, Kap. 4.6.3; Lordieck/Sprenger, 2024, Kap. 4.10).

Einordnung für die Evaluation: Für die im Seminartitel geforderte Gegenüberstellung lässt sich festhalten, dass ADT eine stärkere „In-Editor“-Integration von Testen, Refactoring und Analyse bietet. Dadurch verkürzen sich Feedback-Zyklen, was insbesondere bei iterativen Vorgehensmodellen (wie TDD) die Entwicklungsproduktivität und die Codequalität unterstützen kann. Die ABAP Workbench bleibt in vielen Systemlandschaften weiterhin relevant, erreicht jedoch bei modernen Entwicklungspraktiken weniger Bedienkomfort und Kontextkohärenz als ADT (vgl. Hardy, 2015, Kap. 15.2.1).

5 Diskussion der Ergebnisse

5.1 Zusammenfassende Bewertung

Die Analyse offenbart eine signifikante technologische Divergenz. Hinsichtlich der Entwicklungseffizienz zeigen die ADT durch Features wie Content Assist eine deutliche Überlegenheit gegenüber den statischen Wizards der SE80. Das Kriterium der Funktionalität und Refactoring stellt den deutlichsten Unterschied dar: Was in der SE80 manuelle Arbeit ist, wird in Eclipse zum automatisierten Standard. Bezüglich der Zukunftsfähigkeit zeigt sich, dass die SE80 für klassische ABAP-Entwicklungsaufgaben weiterhin geeignet ist, bei der Umsetzung moderner S/4HANA-spezifischer Technologien jedoch funktionale Einschränkungen aufweist. Zentrale Konzepte wie Core Data Services (CDS) lassen sich ausschließlich in den ABAP Development Tools umsetzen, wodurch sich ADT insbesondere für Neuentwicklungen als notwendige Ergänzung zur klassischen ABAP Workbench etabliert.

Die Bewertung berücksichtigt dabei, dass eine technische Überlegenheit einzelner Werkzeuge nicht automatisch bedeutet, dass diese in jedem organisatorischen, historischen oder projektbezogenen Kontext uneingeschränkt überlegen sind.

5.2 Strategische Implikationen

Für IT-Organisationen empfiehlt sich ein prozessualer "Hybrid-Ansatz". Für reine Wartungsaufgaben im klassischen Report-Umfeld bietet die SE80 weiterhin einen weiterhin sinnvollen Einsatzrahmen. Für sämtliche Neuentwicklungen und Refactoring-Maßnahmen ist der Einsatz von Eclipse ADT jedoch aus Effizienz- und Qualitätsgründen als ratsam anzusehen.

6 Fazit und Ausblick

Die vorliegende Arbeit evaluierte die Entwicklungsumgebungen SE80 und Eclipse ADT. Die Untersuchung zeigt, dass der Wechsel von der serverzentrierten SE80 zur clientbasierten Eclipse-Plattform eine deutliche Veränderung der Entwicklungsweise darstellt. Es konnte nachgewiesen werden, dass die ADT durch asynchrone

Kommunikation, Multi-Tasking und tiefe Integration moderner Syntax-Konzepte (CDS) die Produktivität signifikant steigern. Die SE80 verbleibt als robustes, aber funktional limitiertes Werkzeug für den Bestandsschutz. Für ABAP-Entwickler gewinnt die Nutzung von Eclipse ADT zunehmend an Bedeutung, insbesondere im Kontext moderner ABAP- und S/4HANA-Entwicklungen.

Erklärung

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema
Evaluation der Entwicklung von Programmen in der SAP Programmiersprache ABAP in den Programmierumgebungen Objekt Navigator im SAP Graphical User Interface im Vergleich zu Eclipse mit ABAP Development Tools
selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.
Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Hamza Yavuz

Aachen, den 24.12.2025

Unterschrift der Studentin / des Studenten

Literaturverzeichnis

Hardy, Paul David (2015):
ABAP to the Future

Hardy, Paul David (2021):
Improving the Quality of ABAP Code

Lordieck, Thomas; Sprenger, Manfred (2024):
SAP-Schnelleinstieg: ABAP-Entwicklung in Eclipse - 2., erweiterte Auflage

---

Pęgiel, Łukasz (2021):
ABAP in Eclipse: Install, Configure, Use, and Enhance Your ADT