IHK Berlin

FACHINFORMATIKER/IN FÜR ANWENDUNGSENTWICKLUNG

Protokoll zur betrieblichen Projektarbeit

Diese Erklärung ist der Dokumentation als erste Seite vor dem Deckblatt beizufügen.

Hiermit versichere ich, Frank Loleit

Vor- und Zuname des Auszubildenden

die betriebliche Projektarbeit

Entwicklung eines Moduls in einer Desktopanwendung zur Umstellung von direkten Datenbankzugriffen zu einer gekapselten Datenbankkommunikation mit optionaler Simulationsmöglichkeit

Genaue Bezeichnung der Projektarbeit

sowie die eingereichte Dokumentation unter Betreuung von

Arthur Kirchner / Argus Data Insights

Vor- und Zuname des Ausbilders/ Name des Ausbildungsbetriebes

selbstständig und **ohne fremde Hilfe** konzipiert, verfasst und durchgeführt zu haben. Teile der Dokumentation, die ich **nicht selbstständig** erstellt habe, sind von mir entsprechend **gekennzeichnet** worden. Die von der Verordnung vorgesehene Richtzeit wurde eingehalten. Die Arbeit hat in dieser Form keiner anderen Prüfungsinstitution vorgelegen.

Ich bin darüber aufgeklärt worden, dass meine betriebliche Projektarbeit bei **Täuschungshandlungen, bzw. Ordnungsverstößen mit „Null" Punkten bewertet** wird und als nicht bestanden gilt.

Ich bin weiter darüber aufgeklärt worden, dass dies auch dann gilt, wenn festgestellt wird, dass meine Projektdokumentation im Ganzen oder zu Teilen mit der eines anderen Prüfungsteilnehmers übereinstimmt. Ich nehme zur Kenntnis, dass ggf. stichprobenartige Kontrollen durchgeführt werden können.

Berlin, 26.11.2021

Ort und Datum

Frank Loleit

Unterschrift des Prüfungsteilnehmers

Unterschrift Projektverantwortliche/-r des Auftraggebers

Abschlussprüfung Winter 2021

Fachinformatiker (Anwendungsentwicklung)
Dokumentation der betrieblichen Projektarbeit

Entwicklung eines Moduls in einer Desktopanwendung zur Umstellung von direkten Datenbankzugriffen zu einer gekapselten Datenbankkommunikation mit optionaler Simulationsmöglichkeit.

Abgabetermin: 09.12.2021

Auszubildender:
Frank Loleit
Wildenbruchstr. 43
12435 Berlin

Ausbildungsbetrieb:
ARGUS DATA INSIGHTS® Deutschland GmbH
Gneisenaustr. 66
10961 Berlin

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

GUI Graphical User Interface

DRY Don’t Repeat Yourself

SQL Structured Query Language

SSMS SQL Server Management Studio

PO Product Owner

UML Unified Modeling Language

IDE Integrated Development Environment

Glossar

Abstrakte Klasse Klasse aus der kein Objekt instanziert werden kann. Abstrakte Klassen dienen als Blaupause für weitere Klassen, welche von ihr erben.

Access-Modifier Schlüsselwort aus der Objektorientierten Programmierung, das Sichtbarkeit von Daten außerhalb der eigenen Klasse regelt. Die wichtigsten beiden sind „private“ und „public “.

Assembly Logische Organisationseinheit in .NET, welche einen Baustein einer größeren Anwendung darstellt. Auch „Komponente“ oder „Modul“ genannt.

Boolean Standard-Datentyp aus allen bekannten Programmiersprachen, der entweder den Wert „wahr“ oder „falsch“ annehmen kann.

Breakpoint Markierung am Rand der Zeile im Programmcode, an der die Ausführung des Programms an eben dieser Stelle angehalten wird. Nach dem Anhalten kann im Code-Editor bzw. der integrierten Entwicklungsumgebung der aktuelle Status von Variablen abgefragt werden und bei Bedarf Zeile für Zeile weiter ausgeführt werden.

Connection String „Ein Connection String beschreibt den Standort einer Datenbank (Rechnername oder IP-Adresse ggf. Port oder Dateisystempfad) und den Zugangsweg (Protokoll, Authentifizierung) sowie Verbindungseinstellungen zu dieser Datenbank“[1].

Customer Obsession Wirtschaftsphilosophie, die den Kunden als Maßstab für alle Überlegungen des ökonomischen Denken und Handels emporhebt. Gemäß dieses Ansatzes gibt es keine den Wünschen des Kunden übergeordneten Maxime.

Datenbank Management System Komplexe Applikation mit ausführlicher graphischer Benutzeroberfläche, das für die Arbeit mit Datenbanken und Datenbankanfragen konzipiert ist. Obwohl diese Programme keine Voraussetzung dafür sind, mit Datenbanken zu arbeiten (denn erfahrene Informatikerinnen können dies auch manuell), arbeitet nahezu jedes Unternehmen mit einem solchen System.

Dictionary „Ein Dictionary ist mit einem Wörterburch zu vergleichen. In einem Wörterbuch finden sie unter einem Schlüsselbegriff die zugeordnete Information. So steht etwa in einem englisch-deutschen Wörterbuch unter dem Eintrag *house* der zugeordnete deutsche Begriff *Haus*“ [2].

Dynamisch Von dynamisch geschriebenen Programmiersprachen spricht man, wenn der Datentyp erst in der Runtime durch einen Interpreter entschieden wird und sich jederzeit wieder ändern kann. In einer statisch geschriebene Programmiersprache hingegen, ist er Datentyp unveränderlich und kann sich in der Runtime nicht mehr ändern..

[1] Entnommen aus Schwichtenberg 2021
[2] Entnommen aus Theis 2014 S. 120

IntelliSense Von Microsoft entwickeltes Werkzeug für die integrierten Entwicklungsumgebungen Visual Studio und Visual Studio Code, welches der Programmiererin während beim Entwickeln eine Übersicht darüber gibt, welche Variablen, Klassen und Methoden an der gerade zu bearbeiteten Stelle im Code zur Verfügung stehen.

Kanban Prinzip für das Abarbeiten von Aufgaben im Team, bei dem alle anfallenden Aufgaben mit einer Karte visualisiert werden, die in Säulen von festgelegter maximaler Stückzahl abgelegt sind. Steigt eine Aufgabe im Bearbeitungsstadium auf, wird sie von links nach rechts „gepullt“. Die rechteste Säule trägt immer den Namen „Done“.

Legacy Code Code, der von mittlerweile nicht mehr kontaktierbaren Entwicklern „geerbt“ wurde, nicht mehr in allen Aspekten dem aktuellen Stand der Technik entspricht und darüber hinaus nicht umfassend dokumentiert ist.

List .NET-Klasse, die klassische Arrays aus C# ersetzt. Lists werden zwar wie Arrays eingesetzt, sind aber deutlich dynamischer. So ist deren Datentyp statisch, jedoch nicht deren Größe. D.h. ihre Größe nimmt zu, wenn ein Element hinzukommt und nimmt ab, wenn ein Element gelöscht wird. Zudem besitzen sie mehrere Sortier- und Filterfunktionen.

Open Source Software, die von den Entwickler öffentlich zur Verfügung gestellt wird, ohne dass dabei Kosten für Interessenten anfallen. Die entsprechende Software kann für jeden erdenklichen Zweck verwendet, also auch erweitert oder modifiziert werden. Ebenso kann Open-Source-Software für kommerzielle Zwecke verwendet werden.

Pain Driven Development „Pain Driven Development, or PDD, is the practice of writing software in such a way that you only ‚fix‘ problems when they are causing pain, rather than trying to preempt every possible issue“[3].

Persistenz Die theoretische Möglichkeit eines Objekts unendlich und unabhängig von der Software, in der es erzeugt wurde, zu existieren. In der Praxis kann keine hundertprozentige Persistenz gewährleistet werden.

Product Owner „Der:die Product Owner:in ist ergebnisverantwortlich für die Maximierung des Wertes des Produkts, der sich aus der Arbeit des Scrum Teams ergibt. Wie dies geschieht, kann je nach Organisation, Scrum Team und Individuum sehr unterschiedlich sein“[4].

Pull Request „A pull request is a method of submitting contributions to an open development project. It is often the preferred way of submitting contributions to a project using a distributed version control system (DVCS) such as Git“[5].

Query Datenbankanweisung, welche an einen Datenbankserver verschickt wird und dazu dient, Daten abzufragen, zu erstellen, zu verändern oder zu löschen.

[3] Entnommen aus Smith 2017
[4] Entnommen aus Schwaber und Sutherland 2020
[5] Entnommen aus Johnson 2013

Refaktorierung „Code-Refactoring ist der Prozess der Umstrukturierung bestehenden Quellcodes ohne dessen externes Verhalten zu ändern. Refactoring verbessert keine funktionalen Attribute der Software. Die Vorteile von Refactoring umfassen verbesserte Lesbarkeit des Codes und reduzierte Komplexität“[6].

Single Responsibility Principle Das erste Prinzip aus den fünf vom Softwareentwickler Robert C. Martin formulierten ‚SOLID‘-Prinzipien, welches besagt, das ein Modul nur einen einzigen Grund haben sollte, sich zu ändern. Beispielsweise kann sich ein Modul entweder in Hinblick auf die GUI ändern oder in Hinblick auf die Datenbankkommunikation, aber niemals mal das eine, mal das andere.

SqlCommand Nicht mit „Query“ zu verwechseln. Es handelt sich um eine Klasse aus .NET. Diese speichert Parameter, die für Tabellenspalten stehen, die dazugehörigen Datentypen und eine schematische Query mit Platzhaltern für Werte, die in den Spalten gespeichert werden sollen.

[6] Entnommen aus Kranz 2021

1 Einleitung

In diesem Projekt wird das Ziel verfolgt, die Datenbankkommunikation einer Desktopanwendung zu vereinheitlichen und die Arbeit der Entwickler von der Arbeit der Nutzer abzugrenzen. Obgleich es sich bei dem entwickelten Modul um ein Werkzeug handelt, das nur von Entwicklern geöffnet und verwendet werden kann, steht der Kunde im Vordergrund aller Bemühungen. Denn sowohl das Auffinden und Lösen von Programmfehlern, wie auch das Implementieren von neuen Komponenten kann durch die Vereinheitlichung und Separierung der Datenbankoperationen deutlich effizienter werden. Der Kunde erhält dadurch nicht nur ein tendenziell stabileres System, sondern kann auch damit rechnen, dass neue gewünschte Funktionalitäten schneller umgesetzt werden. In diesem Sinne steht bei allen Bemühumgen dieses Projektes stets die Customer Obsession im Vordergrund.

1.1 Projektbeschreibung

Ausgangspunkt der Arbeit ist die Desktopapplikation „Arche“, entwickelt im .NET-Framework[7] 3.5 mit WinForms-GUI. Das Debugging gestaltet sich teils als recht aufwändig, da alle Funktionen umgangen werden müssen, die Kundendaten generieren, verändern oder löschen. Grund dafür ist das Fehlen von Mechanismen, durch die sich Datenbankoperationen, die von Kunden und Entwicklern vorgenommen werden, jeweils voneinander abgrenzen lassen.
Auch das Implementieren von neuen datenbanksensiblen Features gestaltet sich als umständlich, da es kein einheitliches Prinzip für die Datenbankkommunikation gibt. Viel mehr hat jeder einzelne Workflow jeweils eine eigene Klasse für die Datenbankkommunikation, die sich inhaltlich aber kaum unterscheiden, sondern im Wesentlichen voneinander kopiert wurden und das DRY-Prinzip wenig Berücksichtigung findet. Daher soll ein Modul entwickelt werden, dass die Kommunikation mit der Datenbank kapselt.

Zunächst soll es möglich sein, dass eine Query, die ein beliebiger Workflow an den Datenbankserver übergibt, zurückgehalten werden kann und somit nicht Gefahr zu laufen, unerwünschte Änderungen an Datenbankeinträgen vorzunehmen. Dabei soll in einer für Entwickler bestimmten GUI die entsprechende Query mit sinnvoller Hervorhebung entsprechender Keywords erfolgen. Zudem soll der Entwickler frei entscheiden können, ob die Query verarbeitet, zurückgehalten oder in ein SQL-Management-System übertragen werden soll. Im Anschluss soll die uneinheitliche Datenbankkommuniktation durch eine globale Logik mit Hilfe von Datenbankmodellen ersetzt werden. Aus diesen Modellen soll schlussendlich ein einheitliches Verfahren abgeleitet werden können, das die Datenbankanbindung auf ein einziges Assembly des Projektes konzentriert und somit eine langfristige Arbeitserleichterung schafft.

1.2 Projektziel

Nach Abschluss des Projektes muss es möglich sein, in Arche Insert-, Update- und Delete-Funktionen auszulösen, ohne dabei Kundendaten zu beeinträchtigen. Ist das zu entwickelnde Modul aktiv und es wird in der GUI von Arche ein Speichern- oder Löschen-Button geklickt, soll Arche angehalten und

[7] Microsoft-Corporation 2020

stattdessen ein Fenster öffnen, das nur für Entwickler bestimmt ist. Dieses gibt nun die Query aus, welche an den Datenbankserver geschickt worden wäre. Die Möglichkeit, die Query mit einem Klick umgehend nach SSMS zu versenden muss ebenfalls gegeben sein, wie die Funktion, frei entscheiden zu können, ob die Query von Arche ausgeführt wird oder nicht. Die Entwickler müssen die Möglichkeit erhalten die Datenbank lokal zu simulieren und die entsprechenden Queries lokal auszuführen. Zudem muss ein einheitliches auf Datenbankmodellen basierendes Prinzip implementiert werden, über welches jedwede Insert-, Update- und Delete-Query mit entsprechenden Parametern generiert werden kann.

1.3 Projektumfeld

Argus Data Insights (im Folgenden „Argus“ genannt) ist darauf spezialisiert, Print- , Video- und Onlinemedien - insbesondere Zeitungen, Fernsehen und soziale Netzwerke - für angefragte Zeiträume und Themen zu durchsuchen und dem Kunden skalierbare Dossiers mit Informationen zur eigenen Person, Firma oder Einrichtung zu geben. Die Kunden erlangen auf diese Weise einen detaillierten Einblick darüber, wie viel Aufmerksamkeit der Kunde in den Medien allgemein erhält, welche Themen mit dem Kunden assoziiert werden und und die Berichterstattung tendenziell positiv, negativ oder neutral ausfällt. Die Redaktionssoftware Arche wird sowohl zum Digitalisieren, Annotieren, Aufbereiten und Versenden der für den Kunden angefertigten Dossiers verwendet.
Dabei werden vor allem Print- und Onlineinhalte unterschieden. Die verschiedenen Medieninhalte werden in Fileservern abgelegt, deren Pfade sowie zahlreiche Medienparameter in Datenbanktabellen gespeichert. Darüber hinaus gibt es eine umfangreiche Kundendatenbank. Datenbankzugriffe erfolgen nicht einheitlich, sondern von verschiedenen Persistenz-Klassen aus, deren interne Logik von jeweils der zuletzt angelegten Persistenzklasse kopiert ist (nicht vererbt).

1.4 Projektbegründung

Die Gründe für das Projekt liegen in der erschwerten Wartung der Software, sowie dem erhöhten Aufwand, neue datenbanksensible Komponenten hinzuzufügen. Muss ein Ticket[8] bearbeitet werden, bei dem ein Problem mit Query und/oder Datenbank vermutet wird, müssen beim Debugging sehr früh Breakpoints gesetzt werden, um nicht Gefahr zu laufen, Kundendaten irreversibel zu verändern. Es fehlt die Möglichkeit, global die Ausführung von Datenbankprozessen zurückzuhalten. Darüber hinaus ist es äußerst kompliziert, die entsprechende Query zu erhalten, welche es für das jeweilige Ticket zu analysieren gilt. Daran anschließend ist kein einheitlicher Prozess vorhanden, der die Kommunikation von .NET und MSSQL steuert. Um sowohl Wartbarkeit, als auch einfache Erweiterbarkeit zu gewährleisten, ist dieses Projekt in Absprache mit den Product Ownern entstanden.

1.5 Projektabgrenzung

Was die technische Abgrenzung anbelangt, ist durch die komponentenorientierte Arbeitsweise in .NET das zu entwickelnde Modul gut von bereits fertigen Projekten innerhalb der Kernapplikation getrennt. Konkret bedeutet das, dass das neue Modul „DBEncapsulation“ und dessen Testmodul „DBEncapsulati-

[8] Bei Argus wird das Kanban-Board Atlassian Jira verwendet.

onTests“[9] per se isoliert und interagieren nur dann mit der Hauptapplikation Arche, wenn eine Referenz zu einem anderen Modul hergestellt wird. Wird eine Variable oder Methode aus DBEncapsulation als öffentlich deklariert, ist sie dennoch vorerst in keinem der anderen Module verfügbar und trägt nicht zur Überlastung der IntelliSense bei. Soll das Modul aus einem bereits entwickelten Modul verfügbar sein, muss dessen Projektdatei zunächst eine Referenz auf DBEncapsulation hinzugefügt werden. Im nächsten Schritt kann denjenigen Klassen, welche datenbankrelevante Prozesse implementieren ein „using“-Statement hinzugefügt werden. Erst unter diesen Voraussetzungen, sind in der entsprechenden Klasse Daten verfügbar, die als öffentlich deklariert sind.
Bezüglich der inhaltlichen Abgrenzung ist festzuhalten, dass aufgrund der hohen Vielfalt an Datenbankoperationen zunächst Insert- Update- und Delete im Vordergrund stehen werden. Da Select-Operationen keine Beeinträchtigungen von Kundendaten zur Folge haben können, wird für diese Fälle vor allem im Vordergrund steht den Entwicklern ein korrektes Feedback darüber zu geben, welche Tabellen und Spalten nach welcher Abfragelogik an die Datenbank übermittelt wurden. Da Select-Operationen stark heterogen sind, wird hier kein einheitlicher Prozess für angestrebt. Jedoch soll es für die Fälle Update, Insert und Delete grundsätzlich nicht mehr nötig sein, neue Logiken für neue Module zu entwerfen und lediglich durch Referenzierung auf DBEncapsulation entsprechende Funktionalitäten bereit gestellt werden.

2 Projektplanung

Für die Planung ist wesentlich, dass das Tagesgeschäft der Entwicklerinnen bei Argus nicht unterbrochen wird, sondern parallel zum hier dokumentierten Projekt mit allen zeitlichen und finanziellen Aufwendungen weiterläuft. Das heißt, bei der Projektplanung ist ein sinnvolles Verhältnis von Recherche, selbstständigem Kodieren und eigenverantwortlichen Entscheidungen einerseits, sowie zielgerichtete Unterstützung durch Ausbilder und Kolleginnen andererseits. Um den stetigen Austausch zu gewährleisten, wird der Fortschritt dieses Projektes im täglichen Meeting mit Vorgesetzten und Teammitgliedern geteilt. Zudem werden regelmäßige sowie spontane 1:1-Gespräche mit dem Ausbilder durchgeführt.

2.1 Projektphasen

Das Projekt wurde in einem Zeitraum von 70 Stunden umgesetzt. Dabei wurde die gesamte Stundenanzahl wie folgt aufgeteilt[10]:

Projektphase	**Geplante Zeit**
Analysephase	7 h
Entwurfsphase	11 h
Implementierungsphase	40 h
Abnahme und Einführung	2 h
Erstellen der Dokumentation	10 h
Gesamt	**70 h**

Den Einstieg in das Projekts betreffend ist zu erwähnen, dass es sich bei dem Projekt Arche um Legacy Code handelt, dessen Funktionsweise nur begrenzt objektorientiert aufgebaut ist. Sowohl bei Analyse, Entwurf und Implementierung ist daher immer das Nachvollziehen des Codes durch Recherche, Debugging und Rücksprache mit erfahrenen Mitarbeitern entscheidend.

[9] Siehe A.13
[10] Siehe A.1

2.2 Ressourcenplanung

Bei der Planung der Ressourcen[11] kann zwischen Hardware- und Softwareressourcen unterschieden werden. Bei der Auswahl wurde darauf geachtet, keine neuen Kosten zu generieren. Bei der Hardware kamen daher die von Unternehmen und Bildungseinrichtung bereitgestellten Computer zum Einsatz. Die eingesetzte Software ist entweder Open Source oder es existieren bereits entsprechende Lizenzen bei Argus. Das heißt, es wurden keine Ressourcen angeschafft, die vor Beginn des Projektes nicht vorhanden gewesen wären. Bezüglich des Personals ist ebenfalls festzuhalten, dass sowohl Ausbilder als auch Kolleginnen täglich kontaktierbar oder vor Ort waren. Code-Reviews und Einarbeitungen konnten in den Arbeitsalltag so integriert werden, dass keine neuen personellen Ressourcen notwendig wurden.

2.3 Entwicklungsprozess

Bevor der Entwicklungsprozess starten konnte, war es nötig, den Quellcode von Arche in ausgewählten Stellen zu debuggen. Dabei ging es nicht darum, Fehler zu beheben, sondern Schritt für Schritt den Code nachzuvollziehen. Dies war aus zweierlei Hinsicht sinnvoll: Zunächst hatte ich so die Möglichkeit, Wissen über C# und die Arbeit mit Visual Studio[12] zu sammeln und andererseits die zu bearbeitende Software Arche besser kennenzulernen. Hier wurde zunächst so vorgegangen, dass in der GUI von Arche der Text von Buttons und Labels in die Suchmaske des Visual-Studio-Explorers eingegeben wurden. Auf diese Weise konnten diejenigen Komponenten und Klassen identifiziert werden, welche für das Frontend an der entsprechenden Stelle genutzt wurden. Anschließend wurde weit oben im Code ein Breakpoint gesetzt und dann im Debug-Modus einen Button aus der GUI geklickt.

Da es bei diesem Projekt um Datenbankprozesse geht, wurden vor allem Buttons mit der Aufschrift „Speichern“, „Aktualisieren“ oder „Hinzufügen“ gewählt. Nun wurde in Absprache mit erfahrenen Kollegen der Code immer soweit weiter ausgeführt, bis die datenbankrelevante Funktion erreicht wurde, die Ausgangspunkt für dieses Projekt sein soll. Auf diese Weise konnte nicht nur die Funktionsweise von Arche besser verstanden, sondern auch jene Stellen bestimmt werden, an denen die hier zu entwickelnde Komponente andocken soll. An dieser Stelle muss erwähnt werden, dass es vor allem zu Beginn des Projektes nicht immer gelungen ist, Veränderungen an Datenbankeinträgen rechtzeitig zu unterbinden. Ungewollte Veränderungen von Daten kamen vor allem dadurch zustande, dass nicht immer klar war, ob eine Funktion die Beeinflussung von Daten vorbereitet oder bereits ausführt. Gelegentlich mussten daher einzelne Zeilen in der Datenbank manuell wieder eingefügt oder Rücksprache mit den Kunden gehalten werden. Diese Erfahrung war prägend für die generelle Herangehensweise an das Projekt, welche als Pain Driven Development bezeichnet werden kann.

3 Analysephase

Hier macht es Sinn in eine technische, sowie eine wirtschaftliche Analyse einzuteilen. In technischer Hinsicht wurde bereits angemerkt, dass das Verständnis des Codes nicht über Dokumentationen, sondern

[11] Siehe A.2
[12] In die Auswahl geeigneter IDEs kamen Visual Studio, JetBrains Rider und Monodevelop. Nähere Informationen Siehe A.3

über das Debugging und das Gespräch mit Kollegen funktioniert. Häufig viel dabei auf, dass Funktionen in Arche bei niedriger Schriftgröße deutlich über eine Bildschirmseite hinausreichen können und verschiedene Softwarebereiche wie die Logik der GUI nur schwach von der Datenverwaltung, zu der auch die Datenbanklogik gehört, getrennt wurden[13]. Da auch generell wenig Signaturen und keine Kommentare im Code vorhanden sind, bestand die Aufgabe darin, zunächst zu mutmaßen, welche Teile des Codes mit dem Speichern von Daten zusammenhängen könnten und welche jedwede anderen Funktionen erfüllen, die mit diesem Projekt nicht zusammenhängen.
Was die wirtschaftliche Analyse betrifft, ist nicht zuletzt der Zeitaufwand zu beachten, der notwendig wird, wenn wie im vorliegenden Fall das Single Responsibility Principle nicht eingehalten wird.

3.1 IST-Zustand

Die Datenbankkommunikation von Arche ist so organisiert, dass verschiedene Medienklassen (z.B. Printmedien, Onlinemedien, Socialmedia-Posts usw.) jeweils eine Persistence-Klasse zur Verfügung haben. Benannt sind diese ungefähr nach Mediumname, z.B. „PrintmedienTreffer.cs“ und „PrintmedienTrefferPersistence.cs“. Erstere enthält diverse Variablen und Funktionen, die mit dem entsprechenden Medium zusammenhängen[14]. Es gibt zwar auch hier keine Dokumentation, jedoch finden sich hier und da Kommentare der Entwicklerinnen, die Aufschluss über die entsprechende Geschäftslogik geben können. Es muss jedoch davon ausgegangen werden, dann nicht alle Variablen noch immer benötigt werden und womöglich datenbankrelevante und nicht datenbankrelevante Variablen vermischt sind. Die Abstrakte Klasse „AbstractTreffer“[15] wiederum bildet die Basis für alle Medienklassen und enthält verschiedene Parameter, wie z.B mehrere Identifikationsnummern.
Was die boolischen Werte anbelangt, ist zum Verständnis zu erwähnen, dass in Arche Benennungen wir „deleted“ oder „loaded“ nicht den aktuellen Zustand eines Objektes beschreiben, sondern ausdrücken sollen, dass ein Objekt geladen oder gelöscht werden soll. Würde man also eine Neubenennung dieser boolischen Werte anstreben, wäre eine Formulierung wie „toBeDeleted/toBeLoaded“ sinnvoll. Für das vorliegende Projekt steht jedoch zunächst die Refaktorierung der Datenbanklogik, weniger der äußeren Form, im Vordergrund.

Nutzt man in der Klasse „AbstractTreffer“ die IntelliSense-Funktion für die Referenzabfrage der Funktion „abstract int savePersistent()“, werden 799 Implementierung in zahlreichen Klassen ausgegeben. Es ist also davon auszugehen, dass diese Funktion eine zentrale Rolle für das Ablegen von Daten in der Datenbank spielt. Ebenso wird hier der Arbeitsaufwand deutlich: Da die Funktion abstrakt ist, wird nur die Signatur übergeben. Für jede Klasse, die Daten mithilfe dieser Funktion abspeichern möchte, muss eine eigene Implementierung geschrieben werden. Dies ist zwar sicherlich dem Umstand geschuldet, für unterschiedliche Medien unterschiedliche Speichermechanismen zu benötigen, jedoch wird ist auch klar, dass Arbeitsaufwand zur Weiterentwicklung der Software unverhältnismäßig hoch ist. Obgleich Argus anstrebt, langfristig auf modernere Lösungen wie ASP.NET[16] oder Angular[17] zu setzen, ist das Unter-

[13] Siehe A.18
[14] Siehe A.14
[15] Siehe A.19
[16] https://dotnet.microsoft.com/apps/aspnet
[17] https://angular.io/

nehmen in der Situation, an jedem Wochentag Mediendossiers an Kunden ausliefern zu müssen. Der IST-Zustand der Entwicklung von datenbankrelevanten Komponenten kann also am treffendsten mit zu hoher finanzieller und zeitlicher Belastung beschrieben werden.

3.2 Wirtschaftlichkeitsanalyse

Um das vorliegende Projekt als wirtschaftlich betrachten zu können, muss also eine konkrete Reduzierung der oben genannten Zeit und Kosten erreicht werden. Zentrale Fragen dabei sind: *Wie lange dauert es durchschnittlich der Anwendung eine neue Persistence-Klasse hinzuzufügen? Wie viel Kosten fallen bei der Entwicklung einer solchen Klasse an? Wie viel Zeit und Kosten können durch das Einführen einer gekapselten Datenbankkommunikation gespart werden?*
Hier ergibt sich die Schwierigkeit, dass unterschiedliche Persistence-Klassen natürlich auch unterschiedlich viel Zeit für die Entwicklung und somit unterschiedlich viel Entwicklungskosten mit sich bringen. Dennoch wird hier so akkurat wie möglich versuchst werden, Entlastungen für das Unternehmen mit konkreten Zeit- und Investitionseinheiten anzugeben, um einen Überblick über die Wirtschaftlichkeit des Projektes zu schaffen.

3.2.1 Zusammenstellen des Kostenplans

Der Kostenplan[18] teilt sich ein in die personellen (oben) und nicht personelle Aufwendungen (unten). Es wird davon ausgegangen, dass bei einem Stundenlohn von ca. 40€ immer mindestens ein Kollege für die fachliche Unterstützung zur Verfügung stehen. Die Unterstützung erfolgte dabei in offiziellen Meetings, aber auch spontan im Büro, wenn Gesprächsbedarf bestand. Ebenso verhält es sich mit dem Ausbilder, der für die Betreuung während des Projektes zuständig war und einen etwas höheren Stundenlohn erhält. Dabei muss erwähnt werden, dass ich es für das Betriebsklima nicht als sinnvoll erachtet habe, jede Beteiligte nach ihrem exakten Stundenlohn zu fragen. Die Lösung war daher, den PO nach gewöhnlichen Stundenlöhnen zu fragen, die im Unternehmen üblich sind. Obgleich ich meine Ausbildung vom Staat finanziert wird, ist es hier zielführend, einen durchschnittlichen Stundenlohn eines Praktikanten anzugeben, um zu gewährleisten, dass die Beträge auch für durchschnittliche zukünftige Projekte dieser Größenordnung aussagekräftig sind.

Im Kostenplan wird davon ausgegangen, dass Personal nicht nur über das Gehalt Kosten generiert, sondern dass auch Betriebskosten, wie Strom, Wartung der verwendeten Technik, Heizung im Büro u. Ä. Kosten verursachen. Daher wird ein pauschaler Handlungszuschlag von 30% auf den Lohn angenommen. Natürlich könnte man an dieser Stelle einwenden, dass der Handlungszuschlag in Wirklichkeit stark schwanken kann, dennoch macht es Sinn im Hinterkopf zu behalten, dass neben dem Lohn auch weitere Kosten für den laufenden Betrieb in der Firma anfallen.

Ebenfalls ist anzuführen, dass es sich bei dem Bei der notwendigen Hardware und Software ist zu bemerken, dass es sich bei den ermittelten Beträgen nicht um Anschaffungskosten handelt, sondern jene Kosten, die Hard- oder Software für gewöhnlich in der Zeitspanne des Projektes in Anspruch nehmen.

[18] Siehe A.4

Außerdem handelte es sich bei den unterstützenden Entwicklern um Senior-Entwickler, deren Stundenlohn verhältnismäßig hoch ist. Der Kostenplan gibt also nicht wieder, was zusätzlich für Kosten angefallen sind, sondern wie kostenintensiv das Projekt innerhalb des laufenden Betriebs ist.

3.2.2 Zeitersparnis

Zunächst musste ermittelt werden, wie viel Zeit die derzeitige Entwicklung von Arche ohne einheitliche Datenbankkommunikation angefallen ist[19]. Hier hat sich gezeigt, dass Tickets, bei denen datenbanksensible Prozesse gewartet werden müssen, relativ häufig vorkommen. Hier kann es z.B. vorkommen, dass ein Kunde glaubt, Daten abgespeichert zu haben, die Änderungen der jene Daten jedoch nicht im System sichtbar zu sein scheinen. Für diesen recht frequenten Fall muss die entsprechende Query oft langwierig durch im Debugging-Modus isoliert werden. Es konnte herausgefunden werden, dass das Identifizieren der Queries im Debug-Modus pro Ticket immer mindestens 10 Minuten vergehen und dies etwas weniger als alle zwei Arbeitstage vorkommt.

Zudem wird davon ausgegangen, dass monatlich 2 neue Persistence-Klassen hinzukommen bzw., da es sich um Legacy-Software handelt, gewartet werden müssen. Hierfür werden nach der bisherigen manuellen Vorgehensweise Spaltennamen per Copy-Paste von SSMS nach Visual Studio übertragen, was für jede der beiden monatlich bearbeiteten Klassen einmal geschieht. Folgende in der Tabelle aufgeführten Arbeitsschritte müssen jeweils für Insert-, Update- und Delete-Queries ausgeführt durchgeführt werden, wodurch diese jeweils 6 mal pro Monat durchgeführt werden. Durch das Automatisieren der genannten Prozesse konnte der meiste zeitliche Aufwand reduziert werden und insgesamt 778 Minuten pro Monat an Einsparungen errechnet werden.

3.2.3 Amortisation

Mit der ermittelten zeitlichen Einsparung konnte nun eine Rechnung darüber erstellt werden, wann die Schuld der Projektkosten voraussichtlich getilgt ist. Beim Stundenlohn wird nun wieder das ermittelte Seniorentwicklergehalt von 40 € pro Stunde angenommen, sowie ein Handlungszuschlag von 30%. Auch hier kann man davon ausgehen, dass für zukünftige Projekte auch Entwickler mit moderateren Gehältern beteiligt werden können. Der zufällige Vorteil dieser Konstellation ist jedoch, dass vergleichbare Projekte tendenziell günstiger werden als teurer. Wenn nun also davon ausgegangen wird, dass das Projekt insgesamt 2443,00 € betragen, die reguläre Entwicklung jedoch 8091,20 €/Jahr kostet, dann ist davon auszugehen, dass die Projektkosten nach etwa 3 1/2 Monaten wieder wett sind:

$$778\frac{Minuten}{Monat} \cdot 12\frac{Monate}{Jahr} = 9336\frac{Minuten}{Jahr} = 155,6\frac{Stunden}{Jahr} \tag{1}$$

$$155,6\frac{Stunden}{Jahr} \cdot (40€ + 12€) = 8091,20\frac{€}{Jahr} \tag{2}$$

[19] Siehe A.5

$$\frac{2443,00\,€}{8091,20\frac{€}{Jahr}} = 0,30 \text{ Jahre} \approx 3 \text{ Monate}, 18 \text{ Tage} \tag{3}$$

Auch hier muss realistischer Weise eingeschränkt werden: Natürlich ist dieses IHK-Projekt eines von mehreren Optimierungsvorhaben, die an Arche vorgenommen werden. Wie bereits erwähnt, wird ein langfristiges Vorhaben sein, die App in eine ASP.NET-Architektur zu überführen, was ebenfalls mit Änderungen Zeit und Kosten einhergeht. Die Amortisationsrechnung stellt also eine Annäherung an die wirtschaftlichen Verhältnisse dar, die durch das Projekt beeinflusst werden.

4 Entwurfsphase

Für den Programmentwurf muss zunächst zwischen dem Auffinden von Queries und dem automatisierten Generieren von SqlCommand-Objekten unterschieden werden. Beim Query-Tracing geht es darum, datenbanksensible Module schneller warten zu können und eine akkurate Rückmeldung über entsprechende Datenbankabfragen zu erhalten. Liegt hier ein Entwurf vor, wird dazu übergegangen, generell die Datenbankschnittstelle zu vereinheitlichen und mit Hilfe von Datenbankmodellen, automatisch SqlCommand-Objekte zu erzeugen. Die so entworfene Logik, soll dann global für alle Datenbankprozesse gelten.

4.1 Query-Tracing

Das Auffinden von Queries, die zu einem bestimmten Zeitpunkt von Arche an den SQL-Server geschickt werden, soll so organisiert werden, dass die Entwicklerin nicht mehr den Debug-Modus starten muss, um eine Query zu identifizieren. Allein durch einen einzigen Boolean soll sich das Query-Tracing ein- und ausschalten lassen. Sobald das Query-Tracing aktiviert ist, kann in Arche jedwede datenbanksensible Funktion ausgelöst werden, ohne dass Daten verändert werden.

4.1.1 Anwendungsfall

Der konkrete Anwendungsfall[20] sieht folgendermaßen aus: Ein dafür vorgesehener Boolean wird vom Entwickler auf „true“ gesetzt und damit das Query-Tracing aktiviert. Nun kann es sein, dass die entsprechende Persistence-Klasse, die die Datenbankkommunikation steuert, bereits ein kleines Code-Fragment enthält, durch das das Query-Tracing-Modul zum richtigen Zeitpunkt aktiviert wird. Oder es handelt sich um eine bisher noch nicht refaktorierte Klasse, und der Entwickler muss das in der Dokumentation hinterlegte Code-Fragment noch in die entsprechende Klasse einfügen. Oder es liegt der Fall vor, dass eine andere Entwicklerin bereits mit dieser Klasse gearbeitet hat und daher keine Code-Fragmente eingefügt werden müssen. Nun kann die Entwicklerin Arche in Visual Studio starten und die im zu bearbeitenden datenbanksensible Funktion[21] auslösen wodurch in jedem Fall ein Tracing-Fenster geöffnet wird. Dieses soll nun detaillierte Informationen über die momentane Query ausgeben. Zusätzlich stehen diverse Funktionen zur Verfügung. Obgleich das Modul sicherstellen soll, dass nicht ungewollt Queries an den Datenbankserver geschickt werden, kann man sich dennoch dafür entscheiden, eine Query ausführen zu lassen. Select-Queries verursachen keine Änderungen an Daten und es kann sein, dass sich der Entwick-

[20] Siehe A.6
[21] Dabei handelt es sich v. a. um „Suchen“-, „Speichern“- und „Löschen“-Buttons.

ler bei Insert-, Update- oder Delete-Queries sicher ist, dass diese keine unerwünschten Folgen für die Kundin haben.

Für den ersteren Fall soll das Modul den Select-Befehl automatisch erkennen und die keine Möglichkeit zum Unterdrücken der Query geben. Für die übrigen Fälle, soll man frei entscheiden können. Weiterhin kann der Entwickler sich die Query automatisch bei jeder Nutzung des Query-Trackers in die Zwischenablage kopieren lassen. Ebenfalls wichtig wird die Funktion sein, die Query nach SSMS zu kopieren, um von dort aus mit ihr weiterarbeiten zu können[22].
Möchte der Entwickler auf seinem lokalen Rechner die in Arche genutzte Datenbankstruktur nutzen, besteht die Möglichkeit durch Klick auf den entsprechenden Button, eine Query nach SSMS zu übertragen, die die Struktur des Datenbank gefiltert nach noch in Verwendung befindlichen Tabellen.

4.1.2 Architekturdesign

Um die Entwicklerinnen sinnvoll zu unterstützen, soll die Architektur[23] so gestaltet werden, dass eine sinnvolle Hervorhebung von datenbankrelevanten Keywords eingebaut wird. Zudem sollen sinnvolle Zeilenumbrüche bei entsprechenden Keywords dafür sorgen, dass für die Query wesentliche logische Einheiten schnell sichtbar werden. Hierfür soll in QueryTracing.cs eine einfache List mit Namen „sql-Keywords “ die wichtigsten SQL Keywords speichern. Diese sollen dann in der „QueryInfoforms.cs“ von „SetColorForKeywords“ erkannt und markiert werden. Dabei sollen nur zwei Elemente öffentlich sein: Der Boolean „isActive“ und die Funktion „SuppressQuery“. Der Boolean aktiviert und deaktiviert das Modul, die Funktion setzt alle übrigen Prozesse in Gang, die GUI auf der rechten Seite eingeschlossen. Die anderen Daten oder Methoden sind entweder „private“ (nur innerhalb der Klasse verfügbar) oder „internal“ (nur innerhalb des Moduls DBEncapsulation verfügbar).

4.1.3 Lokale Datenbank für Simulationsmodus

Mit Klick auf den Button „Create local database...“ soll eine spezifische Query zum Erzeugen der lokalen Datenbank nach SSMS übertragen. Beim Erzeugen dieser Datenbank wird nur die Struktur der Arche-Datenbank übertragen, nicht deren Inhalt[24]. Angedacht ist, dass die Entwickler die entsprechende Query nicht mehr zu bearbeiten brauchen, sondern lediglich einen Namen und einen Speicherort vergeben müssen, um hinterher gezielt einzelne Datensätze von der Live-Datenbank auf die lokale Datenbank zu übertragen. Hinterher können die vom Query-Tracer abgefangenen Queries einfach nach SSMS kopiert werden, um diese dann auf der lokalen Umgebung auszuführen.

[22] De facto soll in diesem Fall in Windows immer das Programm die Query öffnen, das in den Systemeinstellungen als Standardprogramm für .sql-Dateien vorgesehen ist. Je nach Konfiguration kann es sich also auch um ein anderes Datenbank Management System handeln.
[23] Siehe A.7
[24] Der Fachbereich Systemintegration von Argus geht davon aus, dass der Inhalt der Arche-Datenbank mehrere Petabytes umfasst und darüber hinaus als Verbund mehrerer Datenbanken zu verstehen ist. Daher ist nur das lokale Simulieren der Struktur sinnvoll.

4.2 Command-Generator

Nach dem Entwurf des obigen Knotenpunktes zur Überwachung der Queries kann nun die Architektur für den Command-Generator entworfen werden. Dieser soll Objekte der Klasse SqlCommand erzeugen. Dabei handelt es sich um eine .NET-Klasse, die in der Lage ist, diverse datenbankrelevante Parameter[25] zu speichern, die für die Datenbankkommunikation erforderlich sind. Es soll ab sofort nicht mehr notwendig sein, Update-, Insert- oder Delete-Queries manuell zu schreiben. Ebenso wenig soll es notwendig sein, die Spaltennamen und deren Datentyp zu kennen oder gar manuell von SSMS nach Visual Studio zu übertragen. Für jede noch in Benutzung befindliche Tabelle soll ein Modell erzeugt werden, dessen Klassenname dem Tabellennamen entspricht. Der SQL-Datentyp soll automatisch in einen .NET-Datentyp überführt werden und für jede Tabelle eine .cs-Datei abgespeichert werden, auf dessen Basis dann Queries und Parameter für die SqlCommand-Klasse generiert werden können.

4.2.1 Entwurf der Konvertierungslogik

Das Generieren von Models bedeutet für dieses Projekt, dass auf Basis des Datenbankschemas von Arche pro Tabelle eine .cs Datei erzeugt wird mit einer Klasse und mindestens eine Property. Diese besteht wiederum dem Access-Modifier „public“, aus einem .NET-Datentyp, eventuell einer nullable-Information, einem Bezeichner und jeweils Getter und Setter[26]. Im folgenden drei Beispiele für Column-Properties mit jeweils unterschiedlichem Datentyp:

```
public Guid? OID { get; set; }
public string Status { get; set; }
public DateTime sys_created { get; set; }
```

Wie erwähnt sind Getter und Setter immer gleich, selbiges gilt für den Access-Modifier. Für jede Property anders sind Datentyp, nullable (das Fragezeichen) und der Bezeichner. Die obige Übersicht soll aus den folgenden Sql-Daten konvertiert werden können:

TABLE_NAME	DATA_TYPE	IS_NULLABLE	COLUMN_NAME
AuftragDimension	uniqueidentifier	YES	OID
AuftragDimension	varchar	YES	Status
AuftragDimension	datetime	NO	sys_created

Dabei muss angemerkt werden, dass .NET Strings immer als nullable ansieht und hier das Fragezeichen wegfällt. Für die Konvertierung des Datenbankschemas zu .NET-Models könnte selbstverständlich auch ein in C# geschriebenes Programm dienen. Da es sich hier jedoch nicht um eine Funktionalität handelt, die Arche später ausführen können soll (denn Arche wird nur mit den Models als .cs-Dateien arbeiten), kann auch eine andere Programmiersprache gewählt werden. Da es dabei nicht unbedingt notwendig ist, komponentenorientiert zu arbeiten und der Datentyp innerhalb des Konvertierungsskripts nicht festge-

[25] U. a. Connection String, Servername, Datenbankname, Tabellennamen, Spaltennamen, Query.
[26] In .NET werden Getter und Setter nicht unabhängig von den entsprechenden Daten deklariert, sondern sind die Architektur von .NET integriert.

legt sein muss (der Inhalt der obigen Tabellen wird immer als string behandelt), bietet sich Python an. Der Vorteil dieser Programmiersprache liegt darin, dass kein gesondertes Projekt angelegt werden muss. Auch ist keine strikte Datenkapselung vonnöten. Viel mehr fördert der kurze lesbare Stil von Python in diesem Fall die schnelle Umsetzung des Entwurfs.

4.2.2 Entwurf für das Erstellen von Models

Für das Struktogramm[27], durch das .NET-Models erzeugt werden sollen, ist es notwendig, mit einer Liste von .NET-Properties zu arbeiten. Denn jede Klasse, die ein Model repräsentiert, enthält mindestens eine Property (analog hat jede Tabelle mindestens eine Spalte). Das Ergebnis des Datenbankschemas ist, wie oben zu sehen, so organisiert dass als erste Position immer aufs neue der Tabellenname erscheint, solange bis jeder in dieser Tabelle vorhandene Spaltenname auf der rechten Seite erschienen ist. Das Skript muss also in der Lage sein zu erkennen, welche Zeilen aus dem Ergebnis zu einer .cs-Datei gehören. Das geschieht, in dem der Tabellenname (oben „AuftragDimension“) als Variable zwischengespeichert werden soll. Sobald eine Zeile abgearbeitet ist, wird geprüft, ob der Name der temporären Tabelle mit dem Namen der folgenden Tabelle übereinstimmt.

Ist dies der Fall, wird die Property nur zur Liste hinzugefügt und die nächste Zeile bearbeitet. Sind die Namen unterschiedlich oder die letzte Zeile ist erreicht, bedeutet dies, dass die Tabelle (und somit der Inhalt der zu erstellenden .cs-Datei) fertig bearbeitet ist. Die Liste der Properties und der Tabellenname wird nun einer weiteren Liste .NET-Models hinzugefügt. Dies soll solange geschehen, bis alle Tabellen abgearbeitet sind.

4.2.3 Entwurf für den Query-Generator

Jedes Objekt der .NET-Klasse SqlCommand enthält eine Query. Dabei kann es sich um eine Query handeln, die ohne weiteres vom Datenbankserver verarbeitet werden kann oder um eine schematische Query, die nur durch Hilfsmittel, wie dem in diesem Projekt entwickelten Query-Tracers, gelesen werden kann. Die schematische Query sieht dabei so aus, dass bei der Zuweisung von Werten zu Spalten auf der linken Seite wie gewohnt die Spalte aufgeführt ist, auf der rechten Seite links vom Gleichheitszeichen jedoch kein Wert, sondern ebenfalls der Spaltenname mit einem @-Symbol davor (z.B. „sys_created=@sys_created“). D.h. die Query in SqlCommand-Objekt dient noch nicht zur Ausführung. Sie soll lediglich bereit stehen, um zum gegebenen Zeitpunkt mit Werten gefüllt zu werden, die der Nutzer in der GUI des entsprechenden Workflows eingibt. Durch die spezifische Schreibweise kann .NET dann das @-Zeichen und den folgenden Spaltennamen durch den gewünschten Wert ersetzen.

Der Query-Generator braucht nun zwei Informationen: *Welche Art von Query soll erzeugt werden (Insert, Update oder Delete)?* und *Wie lautet der Tabellenname?* Der Tabellenname entspricht jeweils dem Klassennamen der entsprechenden .cs-Datei, die nach erfolgreichem Erstellen der oben beschriebenen Models vorliegen soll. Dabei ist in der Regel davon auszugehen, dass alle drei Typen erzeugt werden sollen. Es ist wichtig darauf hinzuweisen, dass es sich bei den Informationen aus den Models nicht um

[27] Siehe A.8

Werte handelt, die aus den entsprechenden Properties abgespeichert sind. Es muss für das Erstellen von SqlCommand-Objekten auch kein Objekt dieser Models erzeugt werden. Lediglich die Bezeichnung von Klasse und Properties auf String-Ebene ist hier entscheidend. Anhand dieser Informationen kann der Query-Generator entsprechende Queries erstellen. Die Klasse[28] ist dabei so organisiert, dass mehrere Hilfsklassen als Datentypen innerhalb von QueryRequest deklariert werden. Diese werden in Lists abgelegt, die der Query-Generator iterieren kann.

So können beispielsweise alle Spalten für eine Insert-Query iteriert werden, indem der Query-Generator zwei strings, „columns“ und „values“ initialisiert, die beide mit einer öffnenden Klammer beginnen. Nun sollen alle Spalten jeweils für beide Variablen iteriert werden, wobei die values Variable, die oben aufgeführte Schreibweise mit „[spalte]=@[spalte]“ verwendet. Anschließend sollen beide Strings eine schließende Klammer erhalten. Die schematische Query kann dann an das SqlCommand-Objekt übergeben werden.

5 Implementierung

Für die Implementierung wurde so vorgegangen, dass zunächst das Query-Tracing, dann die Query-Request-Klasse und zuletzt der Command-Generator mit dem darin enthaltenen Query-Generator implementiert wurde. Diese Reihenfolge wurde gewählt, um den Query-Tracer nach Fertigstellung sogleich auch als Testumgebung für die vom Query-Generator erzeugten Queries zu verwenden.

5.1 Implementierung des Query-Tracers

Die GUI[29] wurde so umgesetzt, dass die Textbox, welche die Query anzeigt, den meisten Raum einnimmt. Um jedoch nicht allzu sehr andere möglicherweise noch geöffneten Fenster zu überlagern, wird die Query ab einer maximalen Länge gescrollt. Datenbankname und Serverame sind immer ersichtlich, ebenso wie der Connection String. Innerhalb der Query werden alle SQL-Keywords erkannt und farblich markiert. Die Farben wurden dabei an diejenigen aus SSMS angelehnt. Das Fenster öffnet sich immer dann, wenn eine Query von Arche an den Datenbankserver verschickt werden würde. Damit der Query-Tracer verwendet werden kann, ist ein kleines Code-Fragment[30] in die Persistence-Klasse einzufügen. Im Falle von Select-Queries wird die Query immer an den Datenbankservers geschickt. Insert, Update und Delete können unterdrückt werden und führen bei Klick auf den entsprechenden dazu, dass „SuppressQuery“den Wert „true “ annimmt und die datenbanksensible Funktion der Persistence-Klasse abgebrochen wird. Arche erhält den Wert „0“ zurück als Anzahl der veränderten Zeilen.

Für die Entwicklerinnen ist es von Vorteil, wenn in SSMS nicht erst die entsprechende Datenbank ausgewählt werden werden muss, um die Query in SSMS erfolgreich auszuführen[31]. Daher wird der Datenbankname (hier: „ArcheDB“) an der entsprechenden Stelle eingefügt. Der Query-Tracer sucht dabei

[28] Siehe A.9
[29] Siehe A.10
[30] Siehe A.17
[31] .NET übergibt den Datenbanknamen nicht in der Query, sondern in einem gesonderten Parameter.

schlicht nach dem Schlüsselwort „from“ und fügt den Datenbanknamen gemäß den Konventionen von SSMS mit zwei Punkten zur Query hinzu.

5.2 Kodierung der Methode zum Erzeugen von Table-Models

Die Methode zum Erzeugen von Table-Models wurde nun in Python kodiert[32]. Dabei hat sich gezeigt, dass nicht alle in klassischen Struktogrammen bekannten Konventionen in einer Dynamisch geschriebenen Programmiersprache wie Python zur Geltung kommen. So gibt es keine Unterscheidung von Deklaration und der Zuweisung von Werten. Ebenso war es nicht notwendig, jede einzelne Kontrollstruktur explizit mit „if“ oder „else“zu kodieren. Bei booleschen Werten konnte die im Struktogramm vorliegende Verzweigung oft mit nur einer Zeile umgesetzt werden. Darüber hinaus muss im Gegensatz zum Struktogramm auch „try “ und „catch “ unterschieden werden[33].
Sobald in der linken Spalte eine neue Table erscheint oder die letzte Zeile erreicht ist, werden alle gesammelten Properties in einer Liste als Parameter für die Klasse „DotNetModel[34]“ übergeben, dass aus den Models den C#-Code erzeugt. Ein ursprünglich angedachtes Dictionary, das auf der einen Seite .NET-Datentypen und auf der anderen Seite SQL-Datentypen enthalten sollte, musste hier nicht umgesetzt werden. Aufgrund der zeichenarmen Syntax von Python war es ausreichend, die Datentypen mit simplem „if “ und „elif “ zu assoziieren[35].

5.3 Implementierung des Query-Generators

Nach der erfolgreichen Implementierung der oben beschriebenen Model-Generators konnte nun der Query-Tracer kodiert werden. Dieser nimmt ein Objekt der in A.9 dargestellten Query-Request-Klasse. Queries sollen mit Hilfe der nun erstellten Models für die Typen „insert[36]“, „update “ und „delete “ erzeugt werden. Bei der Implementierung zeigt sich ein entscheidender Vorteil der automatischen Generierung gegenüber dem manuellen Verfassen einer Query:
Es ist nun sowohl unmöglich geworden, falsche Bezeichnungen für Tabelle oder Spalte zu wählen. ist keine entsprechende .cs-Datei vorhanden, die die Tabelle repräsentiert, wird der entsprechende Versuch unterringelt. Spaltennamen brauchen nun ebenfalls nicht mehr von der Programmierin eingegeben werden und es kann zudem nicht vorkommen, dass die Reihenfolge Spalte und zuzuordnendem Wert nicht übereinstimmt. Es muss lediglich bekannt sein, um welche Tabelle es sich handelt und sie kann nach Eingeben der ersten Buchstaben durch die IntelliSense ausgewählt werden.

5.4 Hinzufügen von Spalten und Datentyp zum SqlCommand-Objekt

Nachdem es nun möglich ist, Queries zu erzeugen, können jetzt die Spalten mit ihrem jeweiligen Datentyp dem Objekt der .NET-Klasse SqlCommand hinzugefügt werden. Dazu müssen die nun im .NET-

[32] Siehe A.15
[33] Hier beim Iterieren der Zeilen: *Ist in der Zeile darunter eine andere Tabelle als die aktuell bearbeitete Tabelle?* Wenn bereits die letzte Zeile bearbeitet wird, muss eine Index-Exception abgefangen werden.
[34] Siehe A.22
[35] *if SqlDataType == ‘varchar‘: DotNetDataType = ‘string‘* usw.
[36] Siehe A.21

Datentyp vorliegenden Properties wieder in den Sql-Datentyp überführt werden[37]. Dabei werden nun alle Properties iteriert, deren Bezeichnung als Parameter mit einem @-Symbol an das Objekt übergeben und der entsprechende Datentyp wieder zur Sql-Architektur hinzugefügt. Auch hier ist zu bemerken, dass es beim Iterieren der Models keinesfalls um den Inhalt irgendeiner Variable geht. Ausschließlich die Zeichenketten von Bezeichner und Datentyp sind hier entscheidend. Der Bezeichner der Property (z.B. „enthaeltUeberschrift“) entspricht also der gleichnamigen Tabelle in der Datenbank. Der .NET-Datentyp „Boolean“ wird durch ein entprechendes Dictionary in den Datentyp „bit“ überführt.

6 Qualitätssicherung und Abnahme

Um höchstmögliche Qualität bei der Datenbankkommunikation zu gewährleisten, war es sinnvoll, sowohl eine erfahrene Entwicklerin, als auch die in der Firma zuständige Datenbankexpertin, sowie den in den Prozess der Qualitätssicherung[38] mit einzubeziehen. Grundsätzlich wird davon ausgegangen, dass die erfahrene Entwicklerin besonders im Bereich .NET unterstützend eingreifen kann, während die Datenbankexpertin natürlich die Qualität des entwickelten Moduls im Bereich SQL prüfen kann. Zunächst wird so vorgegangen, dass der Entwickler (der Praktikant) eigenständig die in dieser Dokumentation dargestellten Features entwickelt. In unregelmäßigen Abständen werden die Ergebnisse der erfahrenen Entwicklerin präsentiert, die die Funktionalität innerhalb von .NET bewerten kann und Rückmeldung bezüglich Bugs und Kodierstil geben kann.

Dieser Vorgang wird so oft durchgeführt wie Entwicklungsbedarf vorhanden ist. Sobald das Modul innerhalb von .NET die gewünschte Qualität erreicht hat, kann der Entwickler Stichproben der mit dem Command-Generator erstellten Objekte von der Datenbankexpertin bewerten lassen. Diese kann die nun automatisch generierten Sql-Command-Objekte mit manuell kodierten Objekten vergleichen. Der QueryTracer[39] dient dabei als Hilfsmittel, um manuell automatisch generierte und und manuell geschriebene Queries zu vergleichen. Alle übrigen Attribute des Sql-Command-Objekts werden mit Hilfe der geprüft. Ist die Prüfung durch die Datenbankexpertin abgeschlossen, wird vom Entwickler ein Pull Request erstellt, welcher von der erfahrenen Entwicklerin bearbeitet wird. Auch diese Arbeitsschritte werden so lange durchgeführt, bis sowohl die erfahrene Entwicklerin, als auch die Datenbankexpertin zufrieden sind. Sind beide Schleifen durchlaufen wird der PO über die Ergebnisse unterrichtet. Er hat nun auch die Möglichkeit mit der Datenbankexpertin im 4-Augen-Gespräch zu entscheiden, ob das neue Modul in den Mainbranch integriert wird.

7 Entwickler-Dokumentation

Für die Entwickler-Dokumentation[40] des Moduls wurde die Markiersprache Markdown gewählt. Diese ist leicht zu erlernen und kann zudem einfach in Web-Oberflächen wie Confluence oder Jira eingebunden werden. Zum Erstellen der Markdown-Datei wurde mit Visual Studio Code gearbeitet. Wird ein entspre-

[37] Für die Zuordnung von .NET-Datentyp zu SQL-Datentyp siehe Microsoft-Corporation 2021.
[38] Siehe A.11
[39] Siehe A.10
[40] Siehe A.12

chendes Mark-Down Plugin als Erweiterung installiert, kann auf der linken Seite das Markdown-Skript bearbeitet werden. Auf der rechten Seite kann die Entwickler-Dokumentation eingesehen werden, so wie sie die Entwickler sehen werden. Die Mitarbeiterinnen und Mitarbeiter von Argus bilden ein internationales Team, das auf Englisch kommuniziert (es sei denn, es sind gerade nur deutschsprachige Kommunikationsteilnehmer zugegen).
Die Entwickler-Dokumentation ist entsprechend ebenfalls auf Englisch. Sie unterscheidet, wie diese Projektdokumentation, zwischen dem Query-Tracer und dem Command-Generator. Im Falle des Query-Tracers sind neben dem Text auch einzelne Screenshots aus der GUI verfügbar. Beim Command-Generator sind zudem mehrere Code-Fragmente vorhanden, die für die Erstellung neuer Persistence-Klassen benötigt werden. Das in dieser Projektdokumentation dargestellte Verfahren, nach dem der Command-Generator funktioniert, ist in verkürzter, handlungsorientierter Form in der Entwickler-Dokumentation auf Englisch mit enthalten.

8 Fazit

Im Laufe des Projektes sind mal mehr technische, mal mehr organisatorische Herausforderungen in den Vordergrund gerückt. Im Folgenden soll ein Überblick über den Soll- und Ist-Zustand am Ende des Projektes, sowie mögliche Erweiterungen für die Zukunft gegeben werden.

8.1 Abschließende Bewertung des Projekts

Im Wesentlichen hat es sich als sehr nützlich herausgestellt, die Expertise von Kolleginnen und Kollegen in das Projekt mit einfließen zu lassen. Unterm Strich konnte so die in A.11 aufgeführte Vorgehensweise eine vollwertige Kapselung der Datenbankkommunikation gewährleisten. Dennoch muss bei den angestrebten Funktionalitäten differenziert werden. Es ist zwar möglich, einen Simulationsmodus mit Hilfe des Query-Tracers aufzubauen, jedoch setzt dieser mehr technisches Wissen vom Entwickler voraus, als zu Beginn des Projektes angedacht. Dies ist zwar grundsätzlich kein großes Problem, denn der Umgang mit SSMS und SQL im Allgemeinen ist den Mitarbeitern bekannt, doch konnte durch die teils manuelle Einrichtung des Simulationsmodus nicht der Komfort erreicht werden, der ursprünglich angedacht war.

8.2 Ausblick

Während die Datenbankkommunikation nun zu großen Teilen auf Models basiert, ist die generelle Architektur von Arche noch nicht einheitlich für die Verwendung von Models konzipiert. Es besteht weiterhin das in A.14 aufgeführte Problem der uneinheitlichen Speicherung verschiedener Daten an verschiedenen Stellen im Code, mit Bezeichnern, die nicht gänzlich mit Tabellen- oder Spaltennamen übereinstimmen. Eine wichtige Aufgabe für den geplanten Übergang von WinForms nach ASP.NET und Angular wird also sein, alle persistenten Daten, immer mit Hilfe von Models zu speichern, die vorher aus dem entsprechenden Datenbankschema generiert wurden. Darüber hinaus ist es sinnvoll, mittelfristig eine Logik zu erarbeiten, wie das Frontend über ein einheitliches Prinzip, die Models ansteuert, ohne sich allzuviel mit dem Backendcode beschäftigen zu müssen. Was die Unternehmenskultur anbelangt, wird es Sinn machen, den Fachbereich Datenbanken noch stärker mit dem Fachbereich Entwicklung zu verknüpfen.

Literatur

Johnson, Mark. *What Is A Pull Request?* 2013. URL: http://oss-watch.ac.uk/resources/pullrequest.

Kranz, Jan-Dirk. *Was ist Code Refactoring?* 2021. URL: https://it-talents.de/it-wissen/code-refactoring/.

Microsoft-Corporation. *SQL Server-Datentypzuordnungen*. 2021. URL: https://docs.microsoft.com/de-de/dotnet/framework/data/adonet/sql-server-data-type-mappings.

– *Übersicht über .NET Framework*. 2020. URL: https://docs.microsoft.com/de-de/dotnet/framework/get-started/overview.

Schwaber, Ken und Jeff Sutherland. *Der Scrum Guide - Der gültige Leitfaden für Scrum, Die Spielregeln*. 2020. URL: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-German.pdf.

Schwichtenberg, Dr. Holger. *Erklärung des Begriffs: Connection String*. 2021. URL: https://www.it-visions.de/glossar/alle/3510/Connection_String.aspx.

Smith, Steve. *Pain Driven Development*. 2017. URL: https://www.weeklydevtips.com/episodes/010.

Theis, Thomas. *Einführung in Python*. 2014. ISBN: 978-3-8362-2861-9.

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase	**7 h**
1. Ist-Analyse	5 h
2. Kostenanalyse	2 h
3. Kommunikation mit dem Fachbereich	1 h
Entwurfsphase	**11 h**
1. Datenbanken sichten	1 h
2. Programmiersprachen und Frameworks auswählen	2 h
3. Graphische Benutzeroberfläche skizzieren	2 h
4. Logik für Datenbankmodelle entwerfen	3 h
5. Konvertierungslogik der Datentypen erarbeiten	1.5 h
6. Logiken für SQL-Command-Generierung skizzieren	1.5 h
Implementierungsphase	**40 h**
1. GUI designen	3 h
2. Textformatierung einpflegen	5 h
3. Query-Verarbeitung programmieren	4 h
4. Andockstellen im bestehenden Code identifizieren	2 h
5. Query-Request-Klasse programmieren	6 h
6. Skripte für das Generieren von Datenbankmodellen schreiben	7 h
7. Query-Generator implementieren	5 h
8. Datenkonvertierung einbinden	4 h
9. Refactoring der bestehenden Datenbanklogik	2 h
10. Test der neuen Datenbankkommunikation	2 h
Abnahme und Schulung	**2 h**
1. Abnahme durch Fachabteilung	1 h
2. Installation	1 h
Erstellen der Dokumentation	**10 h**
1. Erstellen der Projektdokumentation	8 h
2. Erstellen der Entwicklerdokumentation	2 h
Gesamt	**70 h**

Tabelle 1: Detaillierte Zeitplanung

A.2 Ressourcenplan

Hardware
1. Rechner für die Softwareentwicklung (bereitgestellt durch Argus)
2. Rechner für die Dokumentation (bereitgestellt durch das bfw)

Software
1. Microsoft Visual Studio 2019 Professional
2. Microsoft Visual Studio Code
3. Microsoft SQL Server Management Studio 18
4. Pascal Brachet TexMaker 2021
5. Pluralsight Lernplattform (bereitgestellt durch Argus)
6. Atlassian Source Tree
7. Draw.io (UML-Designer)
8. Strutorizer (Struktogramm-Designer)

Programmiersprachen und Frameworks
1. .NET Framework 3.5
2. Python 3.9.7
3. pyodbc (Framework für die Anbindung von Python an MSSQL)
4. WinForms (Grafische Benutzeroberfläche)

Personal
1. Entwickler
2. Administratoren
3. Ausbilder
4. Praktikant

Tabelle 2: Ressourcenplan

A.3 Auswahl der Integrierten Entwicklungsumgebung

		Visual Studio 2019		JetBrains Rider		MonoDevelop	
Kriterium	Gewichtung	Punkte	Gewichtet	Punkte	Gewichtet	Punkte	Gewichtet
Preis	20	1	20	2	40	5	100
YouTube-Inhalte	10	1,75	17,5	0,25	2,5	0,1	1
Foreneinträge	5	1	5	1	5	0,1	0,5
Leichte Erlernbarkeit	15	0,25	3,75	1	15	3,7	55,5
Verbreitung	30	4	120	2	60	0,1	3
Mitgelieferte Features	20	2	40	3,75	75	1	20
Summe	100	10	**206,25**	10	197,5	10	180

Tabelle 3: Auswahl der Integrierten Entwicklungsumgebung

A.4 Projektkosten Details

Angefallen bei	Angefallen für	Std.	Lohn/Std.	Lohn	Zschlg. 30%	Summe
Entwickler	Fachliche Unterstützung	4	40,00 €	160,00 €	48,00 €	208,00 €
Ausbilder	Einarbeitung Betreuung	8	45,00 €	360,00 €	108,00 €	468,00 €
Praktikant	Durchführung Projekt	70	10,00 €	700,00 €	210,00 €	910,00 €
PO	Bereitstellung Kosteninfo	2	60,00 €	120,00 €	36,00 €	156,00 €
Personal		84	155,00 €	1.340,00 €	402,00 €	1.742,00 €
Notebook für die Dokumentation						72,00 €
Desktop-PC für die Entwicklung						69,00 €
Infrastruktur Hardware (Netzwerk, VPN, Server, Verkabelung)						170,00 €
Peripheriegeräte (Docking Station, Maus, Tastatur, Headset)						190,00 €
Infrastruktur Software (Scrum-Tools, Windowslizenz)						60,00 €
Visual Studio Professional Lizenz						140,00 €
Technik						701,00 €
Gesamt						**2.443,00 €**

Tabelle 4: Projektkosten Details

A.5 Zeitersparnis

Arbeitsschritt	Mntl.	vorher	nachher	Gespart
Identifizierung von Queries (Debug)	36	10 min	2 min	288 min
Übertragen von Spaltennamen von SSMS nach VS2019	2	20 min	0 min	40 min
Verfassen der Insert-, Update- und Delete-Query	6	10 min	0 min	60 min
Manuelles Übertragen der Datentypen von SQL nach .NET	6	10 min	0 min	60 min
Hinzufügen der SQL-Parameter zu SQL-Command	6	15 min	0 min	90 min
Prüfung der Daten durch Mitarbeiter	6	40 min	0 min	240 min
Gesamtersparnis pro Monat				**778 min**

Tabelle 5: Zeitersparnis

A.6 Anwendungsfalldiagramm

In diesem Diagramm wird „Query Tracer“ wie eine Person oder Firma behandelt. Dies ist insofern sinnvoll als dass die Entwicklerin nicht mit Argus, sondern mit firmeneigener Software interagiert. „Lokale Datenbank erstellen“ könnte auch von „Query Tracing aktivieren“ abhängen, ist andererseits aber auch ein eigenständiger Arbeitsschritt.

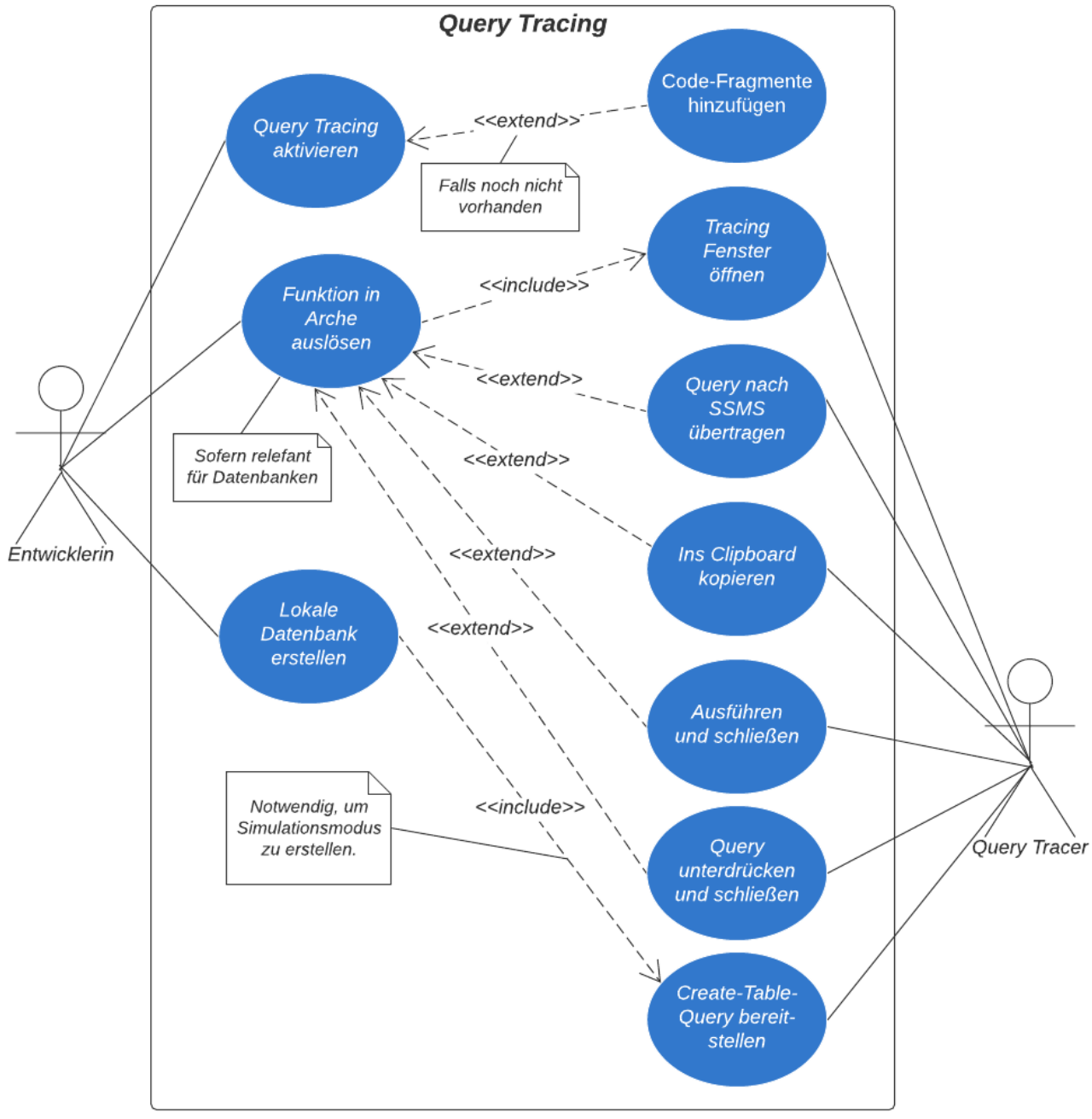

Abbildung 1: Anwendungsfalldiagramm

A.7 Query-Tracing-Klassen

Übersicht über Backend- und Frontend-Logik von Query-Tracing. Diese UML-artige Ansicht wurde mit Hilfe von Visual Studio erstellt. Befindet sich kein kleines graues Symbol rechts unten vom Hauptsymbol, ist der Inhalt „öffentlich“. Ein Herz bedeutet dem Modul „intern“, ein Schloss „privat“. Graue Daten stehen für GUI-Elemente.

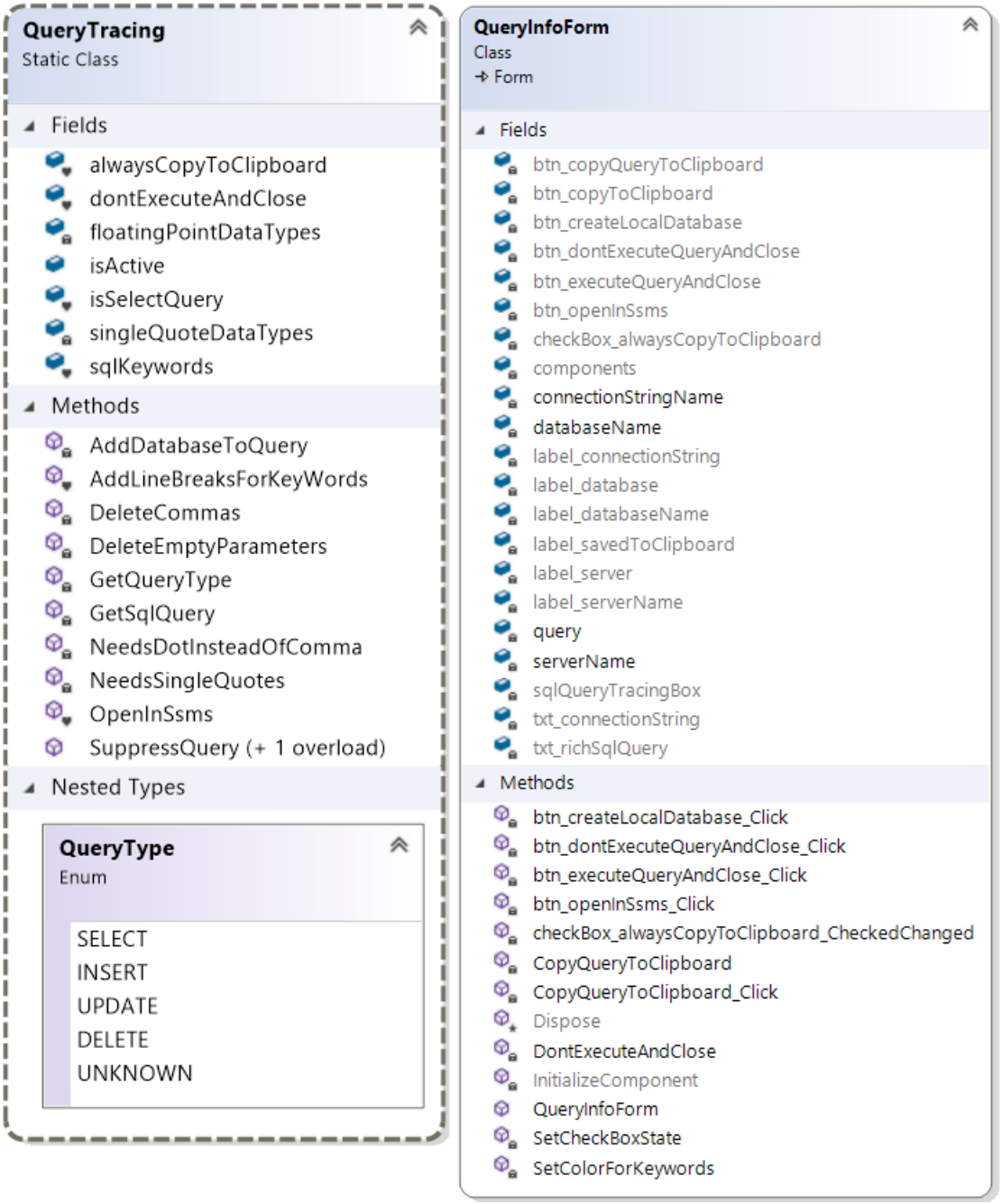

Abbildung 2: Query-Tracing-Klassen

A.8 Struktogramm .NET-Models-Generierung

Dieses Diagram stellt schematisch das Erzeugen von .NET-Models dar. Da die Models mit der dynamisch geschriebenen Programmiersprache Python erzeugt wurden, kann Deklaration und Zuweisung von Werten zu Variablen zusammenfallen. Die Kodierung des Struktogramms ist in A.15 aufgeführt.

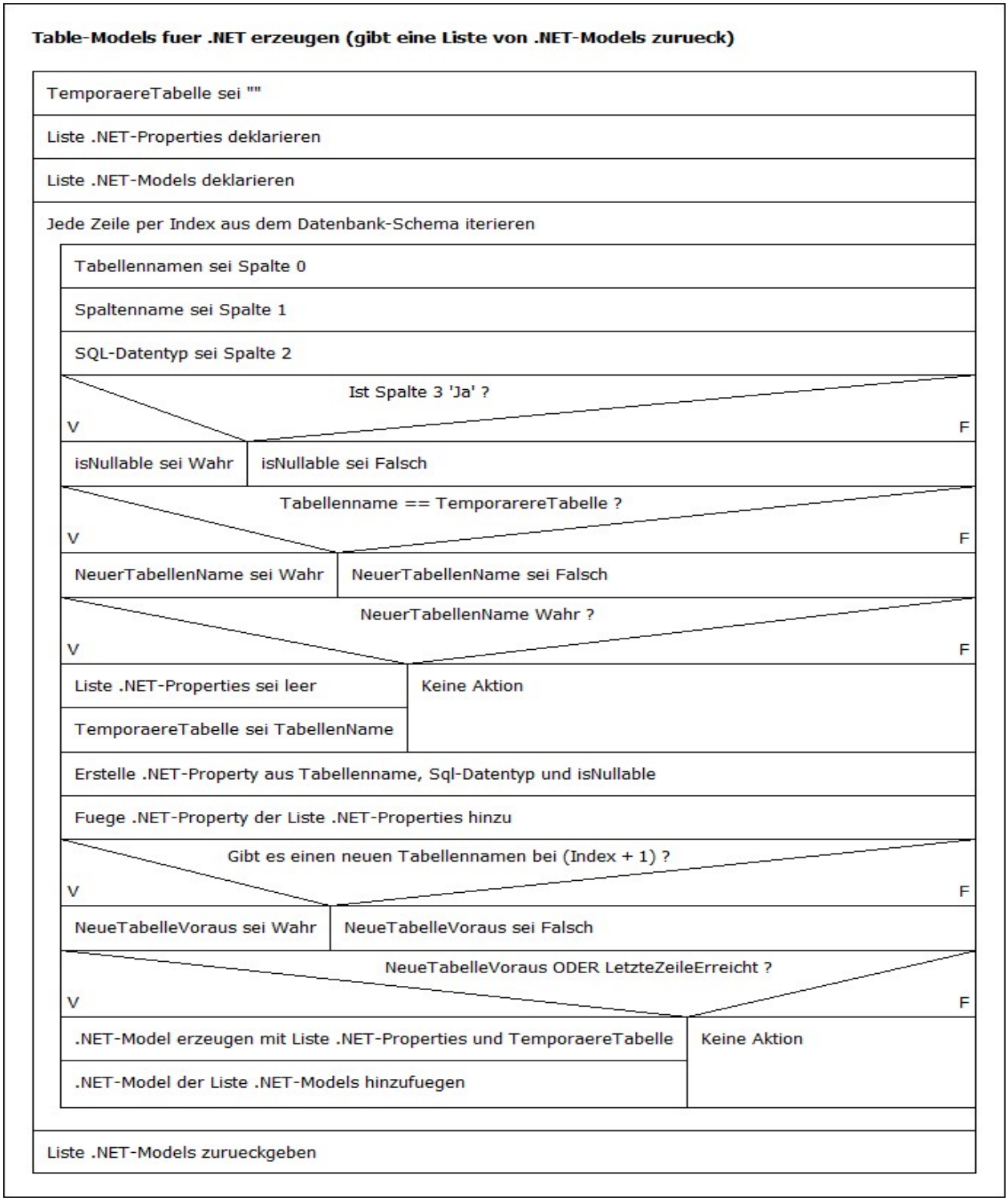

Abbildung 3: Struktogramm .NET-Models erzeugen

A.9 Klassendiagramm QueryRequest

Klassendiagramm der Klasse QueryRequest. Zu sehen sind Variablen und Methoden, sowie alle Klassen und Enumeratoren, die als Datentyp in QueryRequest verwendet werden.

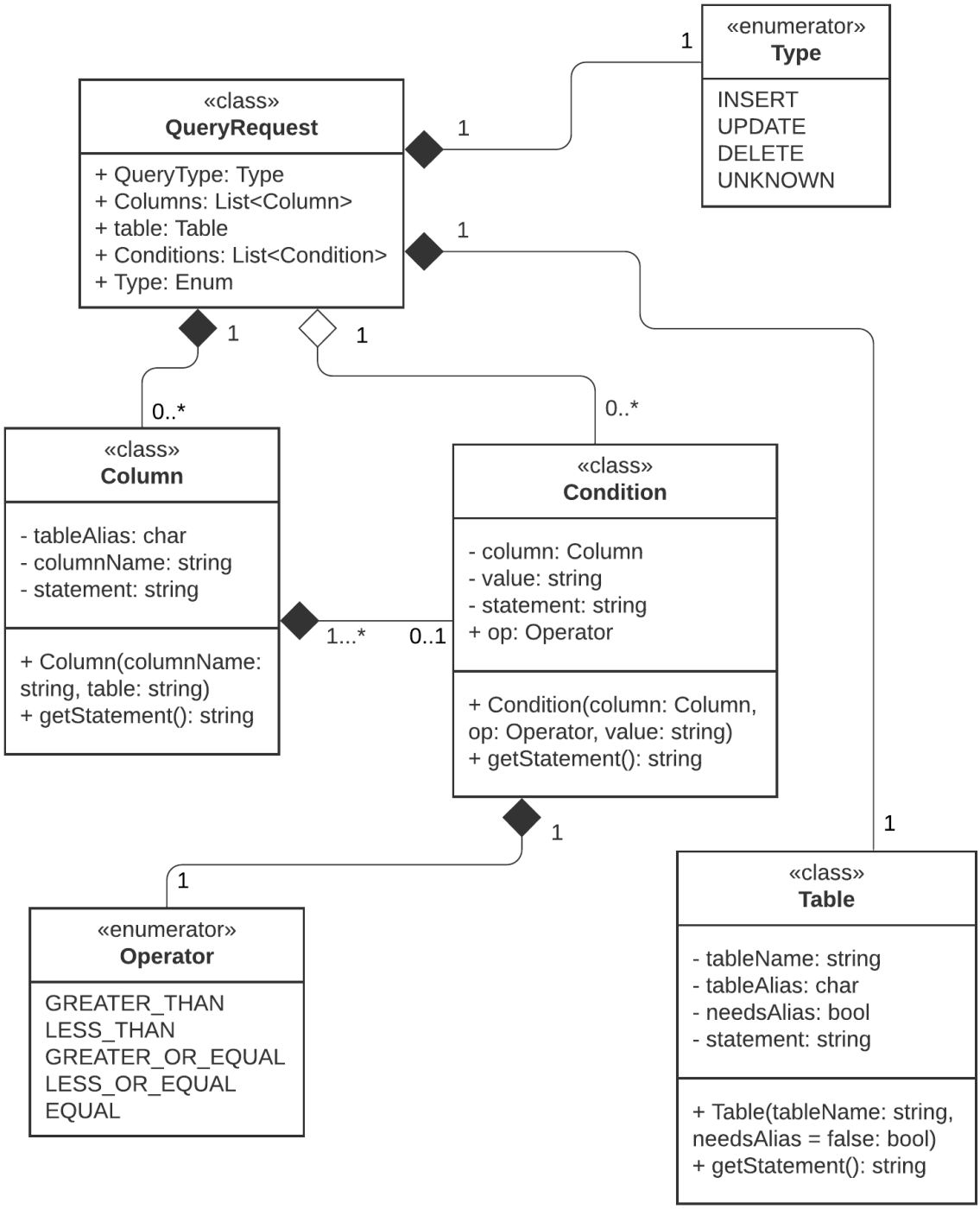

Abbildung 4: Klassendiagramm

A.10 GUI Query-Tracer

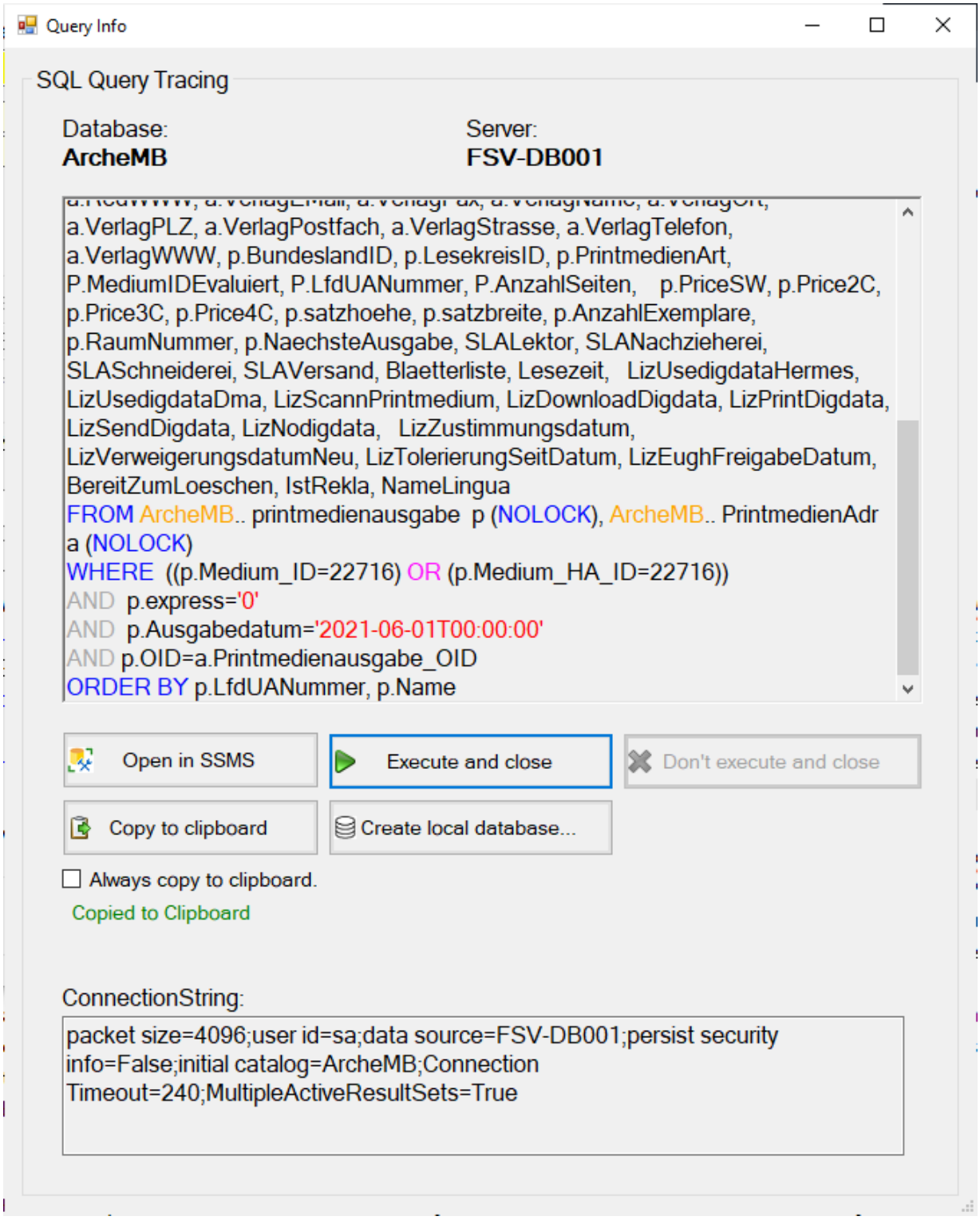

Abbildung 5: GUI Query-Tracer

A.11 Sequenzdiagramm Qualitätssicherung

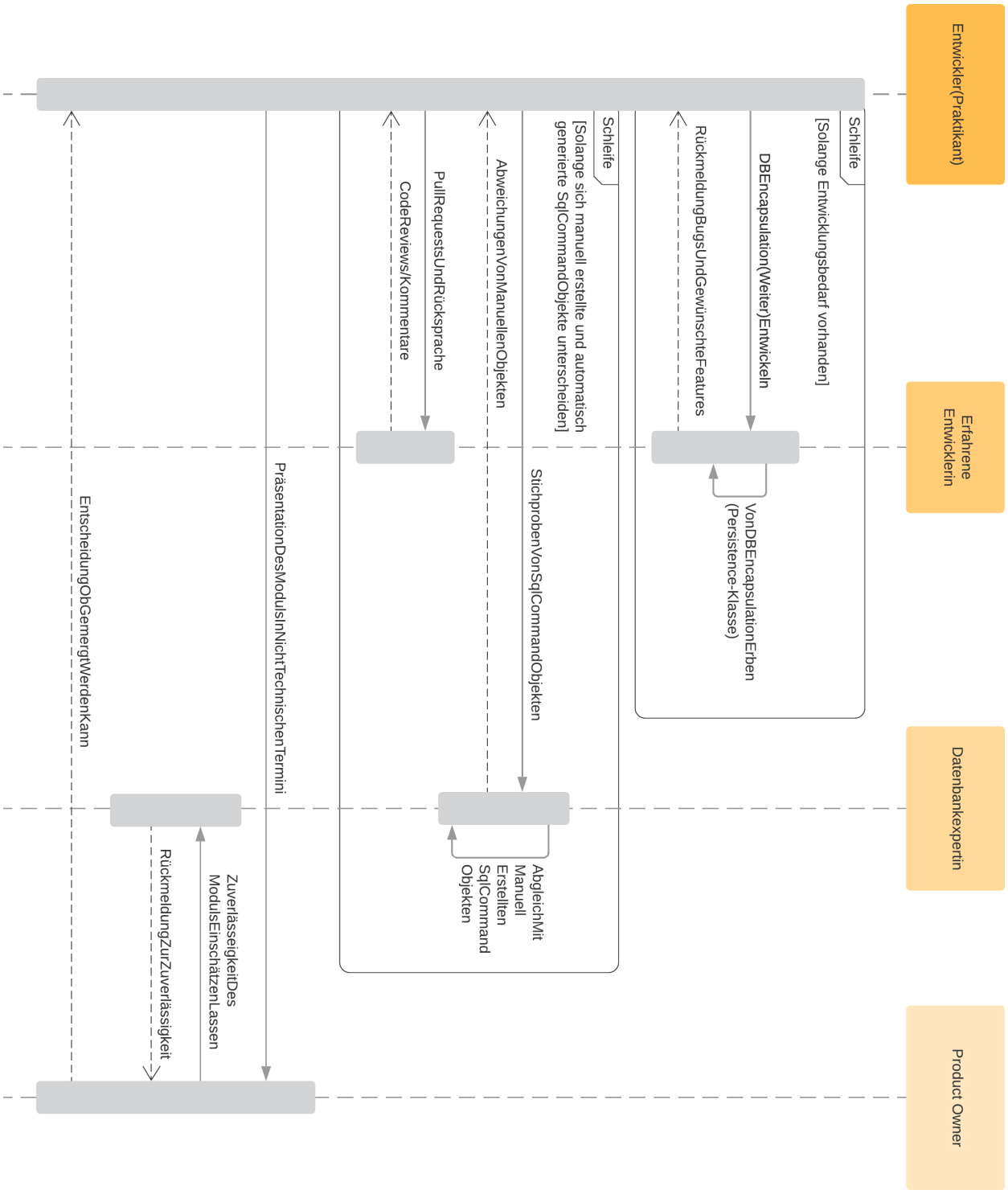

Abbildung 6: Sequenzdiagramm Qualitätssicherung

A.12 Auszug Entwickler-Dokumentation

Abbildung 7: Auszug Entwickler-Dokumentation

A.13 Auszug Projektabgrenzung

Einige Komponenten aus Arche mit den Komponenten DBEncapsulation und DBEncapsulationTests:

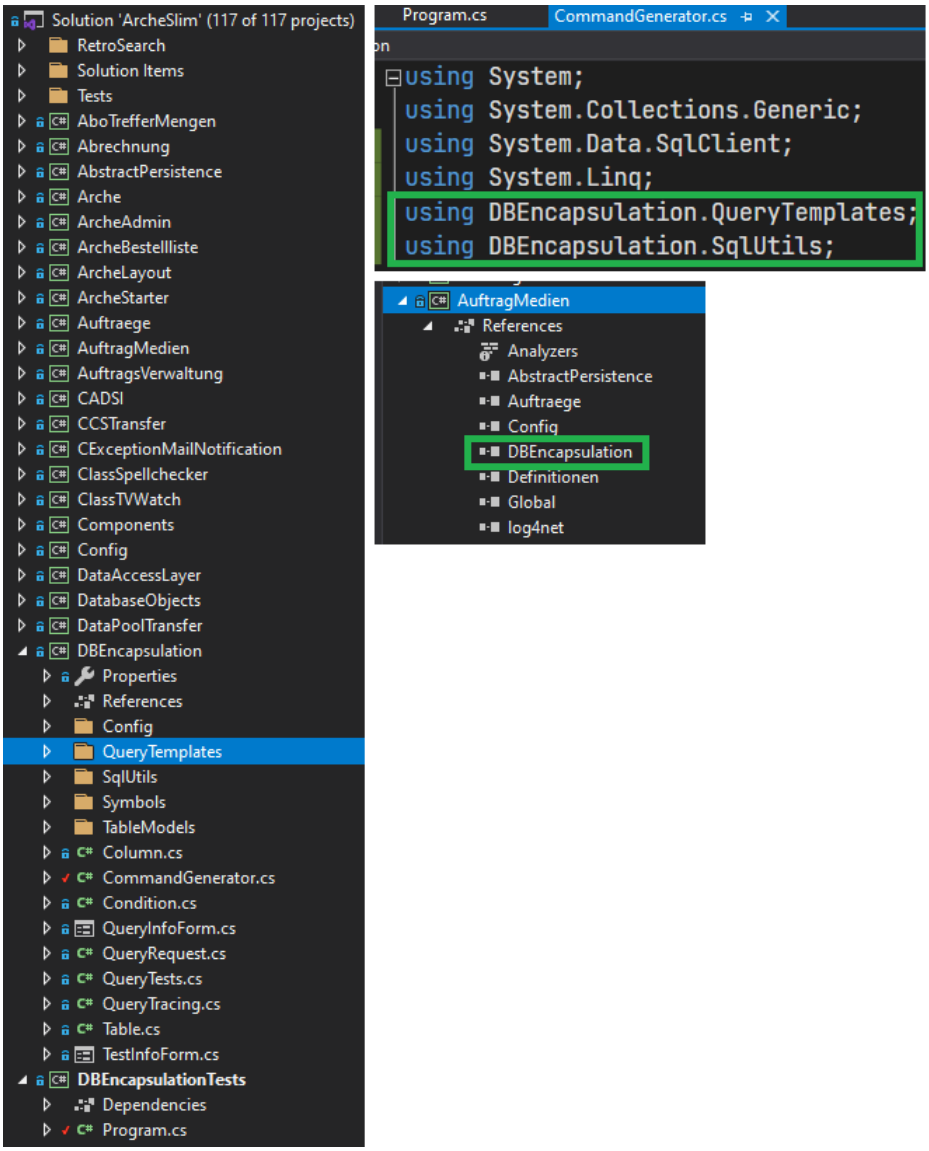

Abbildung 8: Assemblies und Referenzierung

A.14 Auszug PrintmedienTreffer.cs

Mehrere Variablen aus der bereits vorhandenen PrintmedienTreffer-Klasse. Manche sind noch in Benutzung, andere sind obsolet:

```
private Guid fAuftrag_OID;
private string fAuftragsNr;
private int fAboNummer;
private string fAboNummerExtern;
private Guid fOrgClipping_OID;
private bool fAK;
private int fLfdNummer;
private int fWorkflowID;
private int fStatus;
private DateTime fStatusDatum;
private bool fInDataPool = false;
private DateTime fDataPoolDatum;
private DateTime fGedruckt;
private DateTime fKalkVersand;
private string fKundenRubrik;
private int fStatusArchiv;
private int fAnzahlBilder;
private string fModifiedUserid;
private DateTime fTrefferDatum;
private TrefferTarifeListe myTarifeListe;
// NK 07.09.2005 Erweiterungen fuer den Versand MB
private DateTime fLieferscheinGeneriert;
private DateTime fLieferscheinGedruckt;
private DateTime fVersendetDatum;
private string fLieferschein_UserID;
private string fVersand_UserID;
private int fLieferscheinNr;
private int fAnzahlSeitenKA; // Anzahl der Seiten aus dem SJB-Tarif
private int fBerechnet; //Ob der versendete Treffer verrechnet werden soll 1 = ja
private int fTrefferTyp;
private int fHerkunftID; // LN 11.11.2005
private int fAnzahlSeitenKU;
// NK Ende Erweiterungen fuer den Versand MB
```

Abbildung 9: PrintmedienTreffer-Auszug

A.15 Methode zur Erzeugung von Table-Models

Kodierung des in A.8 erarbeitetenden Struktogramms zur Erzeugung von Table-Models.

```
def createModels(dataSet) -> List[DotNetModel]:
    tempTableName = ''
    dotNetProperties = []
    dotNetModels = []
    for index in range(len(dataSet)):
        tableName = dataSet[index][0]
        columnName = dataSet[index][1]
        sqlDataType = dataSet[index][2]
        isNullable: bool = True if dataSet[index][3] == 'YES' else False
        newTableName: bool = tableName != tempTableName
        if newTableName:
            dotNetProperties = []
            tempTableName = tableName
        dotNetProperty = DotNetProperty(columnName, sqlDataType, isNullable)
        dotNetProperties.append(dotNetProperty)
        try:
            newTableNameAhead: bool = dataSet[index + 1][0] != tableName
        except IndexError:
            newTableNameAhead = False
        lastRow: bool = index + 1 == len(dataSet)
        if lastRow or newTableNameAhead:
            dotNetModel = DotNetModel(dotNetProperties, tempTableName)
            dotNetModels.append(dotNetModel)
    return dotNetModels
```

Abbildung 10: Methode zur Erzeugung von Table-Models

A.16 Command-Generator Hauptmethoden

Wesentliche Methoden des Command-Generators. Es wird das gewünschte Table-Model übergeben, sowie der gewünschte Query-Typ aus einem Enumerator. Query und SqlParameter werden entsprechend hinzugefügt:

```
public static SqlCommand GetCommand<TableModel>
    (QueryRequest.Type type) where TableModel : new()
{
    SqlCommand sqlCommand = new SqlCommand
        (GetQuery<TableModel>(type));
    SqlParameterCollection sqlParam = sqlCommand.Parameters;
    AddParamAndDataType<TableModel>(ref sqlParam);
    return sqlCommand;
}
1 reference | FrankLoleit, 16 days ago | 1 author, 1 change
private static void AddParamAndDataType<TableModel>
    (ref SqlParameterCollection sqlParam) where TableModel : new()
{
    TableModel tableModel = new TableModel();
    SqlHelper sqlHelper = new SqlHelper();
    foreach (var column in tableModel.GetType().GetProperties())
    {
        sqlParam.Add($"@{column.Name}",
            sqlHelper.GetDbType(column.PropertyType));
    }
}
```

Abbildung 11: Command-Generator Hauptmethoden

A.17 Code zum Aktivieren des Query-Tracers

Dieses Code-Fragment wird in einer Persistence-Klasse eingefügt, an der Stelle bevor eine Query zum Datenbankserver geschickt wird. Die Variable „anzahl“ stammt aus dem Legacy-Code.

```
if (QueryTracing.isActive && QueryTracing.SuppressQuery(sqlDeleteCommand))
{
   return 0;
}
anzahl = sqlDeleteCommand.ExecuteNonQuery();
```

Abbildung 12: Code zum Aktivieren des Query-Tracers

A.18 Vermischung verschiedener Aufgabenbereiche

Anfang der Speichern-Funktion aus bereits vorhandenen GUI-Klasse. Diese erstreckt sich über mehrere Bildschirmseiten und vermischt Front- und Backend-Logik:

```
private void saveButtonClick(object p_sender, EventArgs p_e)
{

    bool fehlerObjekte = ((currentPrintmedienTreffer == null) ||
        (myClip == null) || (myAusgabe == null));

    if (fehlerObjekte)
    {
        ShowFehlerForm
            ("Eines der relevanten Werte (Treffer, Clip, Ausgabe) " +
            "ist nicht valide!", "Validierung Speicherung");
        return;
    }

    bool fehlerAaew = (currentPrintmedienTreffer.mitAAW() ||
        currentPrintmedienTreffer.mitAAEWGanzeSeite()) &&
        (textBox_AAW.Text.Trim().Length == 0);
    bool fehlerPmgOhneBildangabe = currentPrintmedienTreffer.mitDIG() &&
        myAusgabe.LizenzID.Equals(10) &&
        !(rd_mitbild.Checked || rd_ohnebild.Checked);
    bool fehlerPmgOhneHeadline = currentPrintmedienTreffer.mitDIG() &&
        myAusgabe.LizenzID.Equals(10) && (tbHeadline.Text.Trim().Length == 0);
    bool fehlerNurHeadline = (currentPrintmedienTreffer.mitHeadline() ||
        currentPrintmedienTreffer.mitErfassungInternational())
                            && !myAusgabe.LizenzID.Equals(10)
                            && (tbHeadline.Text.Trim().Length == 0);
    bool istPmgTreffer = currentPrintmedienTreffer.mitDIG() && myAusgabe.LizenzID.Equals(10);

    if (fehlerAaew)
    {
        ShowFehlerForm("Dieser Treffer hat die Dienstleistung AÄW " +
            "und verlangt zwingend einen gültigen Flächenwert!", "Validierung Anzeigenäquivalenzwert");
        if (!hatHeadline)
        {
            return;
        }
    }
```

Abbildung 13: Vermischung verschiedener Aufgabenbereiche

A.19 Auszug AbstractTreffer.cs

Mehrere Variablen aus der bereits vorhandenen PrintmedienTreffer-Klasse. Manche sind noch in Benutzung, andere sind obsolet:

```
public abstract class AbstractTreffer : IArchePersistenceObject
{

    protected Guid fOID;
    protected Guid fClipping_OID;
    protected string fTrefferNr;
    protected Int16 fProdID;
    protected bool fDeleted;
    protected bool fLoaded;
    protected DateTime fSysModified;

    Properties

    99+ references | Manuel Bülow, 210 days ago | 1 author, 1 change
    public abstract int savePersistent();
    10 references | Manuel Bülow, 210 days ago | 1 author, 1 change
    public abstract int savePersistent(System.Data.SqlClient.SqlTransaction transaction);
    11 references | Manuel Bülow, 210 days ago | 1 author, 1 change
    public abstract int savePersistent(System.Data.SqlClient.SqlConnection connection);
}
```

Abbildung 14: AbstractTreffer.cs

A.20 Klasse DotNetModel

Objekte der Klasse DotNetModels beinhalten den C#-Code für jeweils eine zu erzeugende Datei:

```
class DotNetModel:
    def __init__(self, dotNetProperties, tableName,
    namespace='DBEncapsulation.TableModels', frameworks=['System']):
        self.cSharpCode = ''
        self.tableName = f'{tableName}_Table'
        for framework in frameworks:
            self.cSharpCode += f'using {framework};\n'
        self.cSharpCode += '\n'
        self.cSharpCode += f'namespace {namespace}\n' + '{'
        self.cSharpCode += ('\n\t/// <summary>\n\t/// Represents one table,' +
         'its columns and datatypes from ArcheMB\n')
        self.cSharpCode += (f'\t/// Use to save objects of type {tableName}\n\t///' +
         '</summary>\n')
        self.cSharpCode += f'\tpublic class {self.tableName}\n' + '\t{\n'
        for dotNetProperty in dotNetProperties:
            self.cSharpCode += f'\t\t{dotNetProperty.statement}\n'
        self.cSharpCode += '\t}\n}'
```

Abbildung 15: Klasse DotNetModel

A.21 Insert-Methode

Ein Beispiel für das automatische Erzeugen einer Query aus der Klasse Query-Generator. Tabellen- und Spaltennamen werden beim Erzeugen eines Query-Generator-Objekts anhand der Klassen- und Property-Bezeichnung der des entsprechenden Table-Models ermittelt:

```
private string Insert()
{
    if (queryRequest.Columns is null || queryRequest.Columns.Count == 0)
    {
        throw new InvalidOperationException("No columns have been " +
            "transmitted for insert-query.");
    }
    string query = "INSERT INTO ";
    string columns = "(";
    string values = "VALUES (";
    query += $"{queryRequest.Tables[0].Statement} ";
    foreach (Column column in queryRequest.Columns)
    {
        columns += column.Statement;
        values += $"@{column.Statement}";
        if (column ≠ queryRequest.Columns.Last())
        {
            columns += ", ";
            values += ", ";
        }
        else
        {
            columns += ") ";
            values += ")";
        }
    }
    query += columns + values;
    return query;
}
```

Abbildung 16: Insert-Methode

A.22 Klasse DotNetModel

Objekte der Klasse DotNetModels beinhalten den C#-Code für jeweils eine zu erzeugende Datei:

```
class DotNetModel:
    def __init__(self, dotNetProperties, tableName,
    namespace='DBEncapsulation.TableModels', frameworks=['System']):
        self.cSharpCode = ''
        self.tableName = f'{tableName}_Table'
        for framework in frameworks:
            self.cSharpCode += f'using {framework};\n'
        self.cSharpCode += '\n'
        self.cSharpCode += f'namespace {namespace}\n' + '{'
        self.cSharpCode += ('\n\t/// <summary>\n\t/// Represents one table,' +
         'its columns and datatypes from ArcheMB\n')
        self.cSharpCode += (f'\t/// Use to save objects of type {tableName}\n\t///' +
         '</summary>\n')
        self.cSharpCode += f'\tpublic class {self.tableName}\n' + '\t{\n'
        for dotNetProperty in dotNetProperties:
            self.cSharpCode += f'\t\t{dotNetProperty.statement}\n'
        self.cSharpCode += '\t}\n}'
```

Abbildung 17: Klasse DotNetModel