Projektdokumentation

KamAnalyser

Einfache Erstellung digitaler Aufgabenblätter

Dokumentation zur schulischen Projektarbeit von:

Markus Bullmann
Jeremy Gauchel
Martin Mielert
Dominik Kilian
Jochen Scheller

Auftraggeber:	***Klara Oppenheimer Schule***
Projektverantwortlicher:	*Karl Steinam*
Durchführungszeitraum	*01. September – 1.Mai*

Inhaltsverzeichnis

Schnellübersicht

Ansprechpartner

Name	Haupt - Funktion	E-Mail
Scheller Jochen	Teamleiter	mail@scheller-jochen.de
Mielert Martin	Server, GUI	mmielert1302@aol.com
Gauchel Jeremy	Datenbank, Statistik	jakry@web.de
Bullmann Markus	Wordwrapper, TFS	markusbullmann@gmail.com
Kilian Dominik	System-Architektur	dok@salt-solutions.de

Verantwortlicher

Name	Funktion	E-Mail
Steinam Karl	Lehrer, Auftraggeber	steinam@klara-oppenheimer-schule.de

Dokumentenhistorie

Version	Bemerkung
0.1	Grobstruktur
0.2	Verzeichnisstruktur
0.3	Textfassung
0.4	Bilder anpassen
0.5	Layout
1.0	Final

Dokumentation

1. Einleitung

Die Gruppe Plus49 wurde für das Projekt KamAnalyser im Rahmen des Berufsschulunterrichts im Fach AWP ins Leben gerufen. Oben genannte Gruppe besteht aus fünf Auszubildenden/Schülern der Klasse 12Fi2. Im Gründungsgespräch wurde jedem Mitglied ein Aufgabenschwerpunkt zugewiesen, für den er verantwortlich handelt. Ziel ist es neben der gemeinsamen Projektbearbeitung, zielführende Absprachen untereinander zu treffen, an die sich die Gruppenmitglieder zu halten haben. Ebenso soll eine firmennahe Umgebung und Vorgehensweise geschaffen werden.

2. Grundlagen

Den Usern des KamAnalysers soll es möglich sein, Aufgaben (vorwiegend Zwischen- und Kammerprüfungen) in eine Datenbank einlesen zu lassen. Alle Aufgaben sollen mit Meta-Tags versehen werden, um sie später einfacher selektieren und gruppieren zu können. Mittels eines „Stegreifaufgaben-Konverters“ können die Lehrkräfte sich später Aufgabenblätter erstellen lassen.

2.1 Projektziele (Auftraggeber)

Vordergründiges Ziel des Projekts ist nicht die Fertigstellung des Projekts, sondern die Verwendung geeigneter Hilfsmittel während der Erstellung des Programms, sowie Unit-Tests und zugehörige Dokumentationen.
Die Mittel sollen so gewählt werden, dass die Vorgehensweise anhand von Modellen, Beschreibungen und Strukturen nachvollziehbar ist.
Mit Hilfe der Metatags, die auch das Thema beinhalten, sollen Statistiken mit der Themenhäufigkeit erstellt werden.

2.2 Projektziele (Gruppenintern)

Das Programm soll Worddateien einlesen können, in Bytes umwandeln und in einer Microsoft SQL-Datenbank speichern. Die Metatags zu den Aufgaben sollen so gewählt werden, dass man Art der Aufgabe, Jahr, Thema auslesen kann. Weitere Infos können ebenso hinterlegt werden.
Wenn die Aufgaben in der Datenbank gesucht werden, kann man sich seine Treffer mittels einer Word-Preview anzeigen lassen (read-only), oder auch bearbeiten. Hierzu öffnet sich das Dokument als Word-Dokument.
Die Treffauswahl kann man sich zu einer Stegreifaufgabe zusammenführen und ggf. drucken.

2.3 Meetings

In geeigneten zeitlichen Abständen sollen Gruppen-Meetings stattfinden. Hier sollen größere Probleme aufgegriffen und diskutiert werden. Sollten sich daraus Fragen an den Auftraggeber ergeben, wird dieser ggf. hinzugezogen. Außerdem wird der Stand älterer Aufgaben geprüft, um die selbst gesetzten Zeitpläne einzuhalten. Zum Abschluss werden die anstehenden Aufgaben zugewiesen, eine Zeitschätzung erstellt und Prioritäten festgelegt.
Diese To-Do's werden zusammen mit einem Protokoll im Team Foundation Server (TFS) angelegt.

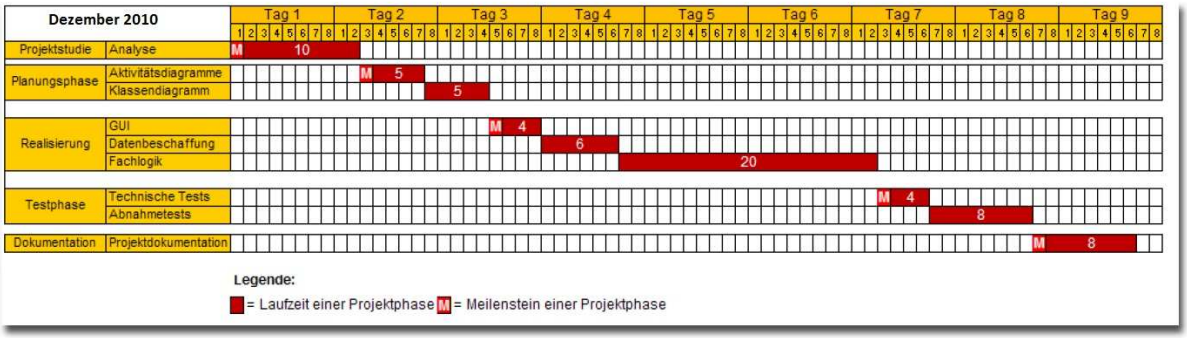
3. Projektanalyse

3.1 Ist-Analyse

Aktuell hat die Klara-Oppenheimer-Schule kein digitales System zur Speicherung, Auswertung und Verwaltung von Prüfungsaufgaben. Dies musste bisher von Hand erfolgen, was dementsprechend viel Zeit in Anspruch nahm. Aufgrund spezieller Wünsche wurde der Kauf entsprechender Software nicht in Betracht gezogen.

3.2 Sollanalyse

Aufgrund der aufgezeigten Nachteile der aktuellen Situation, soll im Zuge dieser Projektarbeit ein System programmiert werden, welches obige Nachteile egalisiert, genannte Ziele realisiert und trotzdem - auch für nicht IT-Lehrer – problemlos anwendbar ist.
Ein wichtiger Teil soll auch eine Projektdokumentation, mit genauer Weg- und Problembeschreibung sein.



(Soll-Zeitplan, mit Bearbeitungszeitrum)

4. Planungsphase

4.1 Speichertechnologien

Die Planungsphase für den generellen Aufbau des Programmes bzw. der Programmstruktur wurde in den ersten Gruppenmeetings durchgeführt. Dabei galt es zunächst ein geeignetes Dateiformat zu finden, welches im SQL-Server abgelegt wird. In die engere Auswahl kamen dabei folgende Formate:

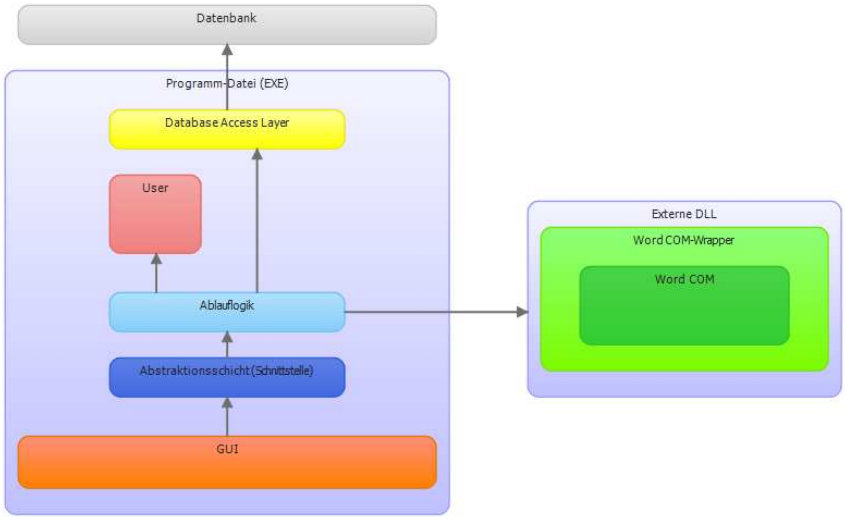
- Bilder (PNG, JPG)
- HTML
- XML
- Word Dateien

Es wurden Gruppen gebildet, die diese Speichermöglichkeiten auf Durchführbarkeit, Ressourcenverbrauch, Geschwindigkeit, Vor- und Nachteile prüfen sollten. Nach Vorstellung aller Möglichkeiten, wurde die Speicherung der Aufgaben als Word Dokument gewählt. Die genaue Abwägung der Varianten befindet sich im Anhang unter Speichertechnologien.

4.2 Windows Installer XML

Auf der Suche nach einem geeigneten Tool, welches in der Lage ist ohne größere Vorkenntnisse eine Installationsdatei zu erzeugen, haben wir uns für WIX entschieden.
Da es als freie Software veröffentlicht wurde, mussten wir uns nicht um entsprechende Lizenzen kümmern. Der große Vorteil gegenüber anderen Setup-Systemen ist, das WIX den Endzustand des Programms beschreibt und nicht das Framework der Installation. Somit können alle Details des Installationsvorgangs genau auf das Zielsystem abgestimmt werden.
Vor allem aber auch die Integration in Visual Studio, welches als Haupt-Programmiertool eingesetzt wird, sowie die Möglichkeit der Anbindung an den bereits bestehenden TFS-Server sprechen zusätzlich für die Verwendung von WIX. Zusätzlich kann es einfach in das Microsoft-Bundle integriert werden und die Kompatibilität mit unserem Buildprocess (MS Build) kann problemlos konfiguriert werden.
Für die Oberfläche des Installations-Routine haben wir uns für „Mondo“ entschieden, da hier alle typischen Installationsdialoge vorhanden sind.

5. Projektrealsierung



(Layer-Diagramm, große Ansicht im Anhang)

5.1 Server

Eine virtuelle Maschine, 64bit-fähig, geeignet für Windows-/ Linux-Server wurde vom Auftraggeber zur Verfügung gestellt. Die Admin-Berechtigungen wurden ebenfalls bereits vor Projektbeginn an Plus49 übergeben. Die Hardware-Ressourcen belaufen sich auf einen Dual-Core E5405 mit 2*2 GHz CPU der Intel Xenon-Klasse, sowie zugesicherte 4 GB Arbeitsspeicher.

5.1.1 Anfängliche Implementierung

Zum Projektstart wurde ein Windows 2008 Server mit MySQL-Server gewählt, da uns entsprechende Lizenzen für ein hausinternes Projekt zur Verfügung standen und die notwendigen Kenntnisse sowohl im Windows-, als auch im Datenbankbereich vorhanden waren. Zusätzlich kamen Mantis Bug Tracker und Mantis Graph zum Einsatz. Somit sollten Problemdokumentation, Arbeits- und Aufgabenzuweisung, sowie graphische Darstellung von Problemverläufen ermöglicht werden. Abgerundet wurde das Softwarepaket durch die freie Softwareverwaltung Apache-Subversion, welches die Aufgabe der Versionsverwaltungen für Dateien und Verzeichnisse übernehmen sollte.

5.1.2 Änderung der Server-Umgebung

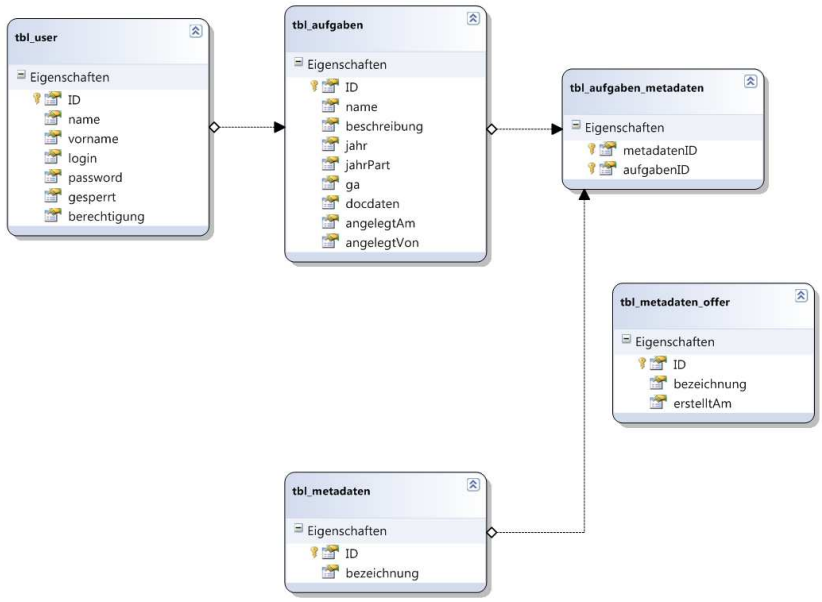
Auf nachträglichen Wunsch des Auftraggebers wurde ein Team Foundation Server installiert und eingerichtet. Die entsprechende Microsoft-Lizenz wurde zur Verfügung gestellt. Mit dieser Maßnahme konnte das gemischte Software-Bundle ersetzen werden. Zusätzlich bietet diese vereinheitlichte Lösung auch Vorteile in der Nutzung und Handhabung. So brauchen die Team-Mitglieder nur noch einen einzigen Login. Alle eingetragen Probleme und Lösungen können mit der entsprechenden Code-Datei der Versionsverwaltung versehen werden und der Administrator muss sich nur noch um eine Software kümmern (z. B. Updates, etc.). Trotzdem können die Accounts – je nach Aufgabengebiet - in den Teilgebieten mit verschiedenen Rechten versehen werden.
Durch den späteren Umstieg auf den TFS bot sich auch der Wechsel auf eine Microsoft Datenbank an. Somit mussten bereits angefertigte Lösungsskizzen und Notizen nochmals überarbeitet werden.
Neben dem zeitlichen Mehraufwand der Installation, traten auch usertechnische Probleme durch das Klonen der virtuellen Maschine auf. In der User-Datenbank des TFS standen auch die User der „Original-VM“. Desweiteren gab es anfänglich Probleme mit dem Zugriff der Clients.
Aus lizenztechnischen Gründen ist der Server nun nur noch im Hausnetz der Klara-Oppenheimer-Schule verfügbar.

5.2 Arbeitsumgebung

Während der Projektphase stehen neben der Unterrichtszeit auch die Client-PCs im Raum A108 für das Projekt zur Verfügung. Diese wurden mit aktuellen Office-Lizenzen, sowie Visual Studio 2010 ausgestattet. Diese Rechner dienen nur zur Programmierung und Anfertigung von Skizzen und Dokumentationen. Diese sollen aber nicht dort oder im Netzwerk gespeichert werden, sondern verpflichtend im TFS gepflegt werden.

5.3 Datenbank

5.3.1 Datenbankmodel



(Datenbankmodel, große Ansicht im Anhang)

Die Datenbankstruktur im SQL-Server besteht aus fünf Tabellen, die u.a. über Foreign Keys miteinander verbunden sind. Die oben dargestellten Tabellenanzahl und –struktur wurde von uns gewählt, da wir somit in der Lage waren alle anfallenden Daten speichern zu können, die Datenbank dennoch übersichtlich bleibt und spätere Erweiterungen einfacher implementiert werden können.
So wird es zum Roll-out-Termin in der „tbl_user“ nur die Benutzertypen admin und Gast geben, für spätere Erweiterungen ist aber eine User-Tabelle dennoch sinnvoll. Ein Gast kann nur Aufgaben hinzufügen, bearbeiten und lesen, während ein Admin zusätzlich noch die Metadaten verwalten kann.
Zusätzlich ist die Bennenung der Felder und Tabellen sprechend gewählt worden. In der „tbl_aufgaben“ werden die einzelnen Aufgaben vorgehalten. Für jede Aufgabe wird ein Word-Dokument (in der Spalte „docdaten“) binär gespeichert, um Inhalt und Formatierung nicht trennen zu müssen, sowie eine Anzeige ohne größeren Aufwand zu ermöglichen.
Aus dem Abbild des Datenbankmodels ist der Verweis der „tbl_user“ auf die „tbl_aufgaben“ ersichtlich, d.h. zu jeder Aufgabe wird der Nutzer gespeichert, der sie angelegat hat („angelegtVon“).
In der „tbl_metadaten“ werden alle vorhanden Metadaten aufgeführt und mit einer ID versehen. Da es zu erwarten ist, dass Metadaten häufiger verwendet werden, kann man mit dieser Extra-Tabelle den Speicheraufwand reduzieren, da man bei einer Aufgabe nur noch die ID der Metadaten halten muss. Außerdem gilt: Eine Aufgabe kann zu beliebig vielen Metadaten zugeordnet werden und ein Metabegriff kann bei mehreren Aufgaben zutreffen. Um diese n:m-Verbindung realisieren zu können, wurde die Tabelle „tbl_aufgaben_metadaten“ erstellt.
Da die Anwendung KamAnalyser eine interaktives Programm werden soll, bei dem alle Benutzer nicht nur profitieren, sondern auch mitarbeiten sollen, haben wir mit der Tabelle „tbl_metadaten_offer“ für den Typen Gast die Möglichkeiten geschaffen, Metadaten vorzuschlagen.

```
CREATE TABLE [dbo].[tbl_aufgaben]
(
    [ID] int NOT NULL,
    [name] varchar(50) NOT NULL,
    [beschreibung] text NULL,
    [jahr] int NOT NULL,
    [jahrPart] varchar(10) NOT NULL,
    [ga] int NOT NULL,
    [docdaten] varbinary (max) NOT NULL,
    [angelegtAm] datetime NOT NULL,
    [angelegtVon] int NULL,
    CONSTRAINT [PK_aufgaben] PRIMARY KEY ([ID]),
    CONSTRAINT [FK_aufgaben_user] FOREIGN KEY ([angelegtVon]) references [dbo].[tbl_user] ([ID])
)
```

(Bsp. Create Table)

5.3.2 Datenebenenanwendung

Eine Datenebenenanwendung (Date-Tier Application, DAC) ist einer neuer Projekttyp in Visual Studio 2010, mit der eine Datenbankstruktur erstellt und gepflegt werden kann. Beim Kompilieren einer DAC wird automatisch eine auf XML-basierende DACPAC-Datei erzeugt. Dort werden alle Informationen für eine Datenbank gespeichert. Damit kann dann in einem beliebigen SQL Server 2008 eine Datenbank erstellt werden.
Der durch die Verwendung des DAC entstandene einmalige Mehraufwand, wird durch die entstehenden Vorteile nicht nur kompensiert, sondern bietet auch deutlich mehr Vorteile für die spätere Programmierung und Usability. So können durch die lebende Datenbankstruktur Erweiterungen und Anpassungen schneller umgesetzt werden. Außerdem kann man die komplette Struktur via SQL-Scripting gepflegt werden und Visual Studio kann die DAC via Mausklick bereitstellen, die anschließend dem ausgewählten SQL Server 2008 automatisch zur Verfügung steht. (Rechtsklick-> Bereitstellen).
Zuletzt sind noch die zusätzlichen Features für eine erleichterte Bedienung einer DAC zu erwähnen. So stehen neben dem bereits erwähnten SQL-Scripting u.a. auch Data Generation Plans, Pre- und Post-Deployment SQL-Scripts zur Verfügung.

Aufgrund der aufgezeigten Vorteile, fiel die Entscheidung zur Verwendung einer Datenbankebenanwendung, obwohl der Mehraufwand durch geringe Vorkenntnisse noch erhöht wurde.

5.3.3 LINQ to SQL

Eine kurze Einleitung und Definition zu LINQ to SQL soll hier ein Zitat aus der MSDN von Microsoft liefern:
(http://msdn.microsoft.com/de-de/library/bb386976.aspx)
„In LINQ to SQL wird das Datenmodell einer relationalen Datenbank einem Objektmodell zugeordnet, das in der Programmiersprache des Entwicklers ausgedrückt ist. Wenn Sie die Anwendung ausführen, übersetzt LINQ to SQL die sprachintegrierten Abfragen im Objektmodell in SQL und sendet sie zur Ausführung an die Datenbank. Wenn die Datenbank die Ergebnisse zurückgibt, übersetzt LINQ to SQL diese zurück in Objekte, die Sie mit Ihrer eigenen Programmiersprache bearbeiten können.“

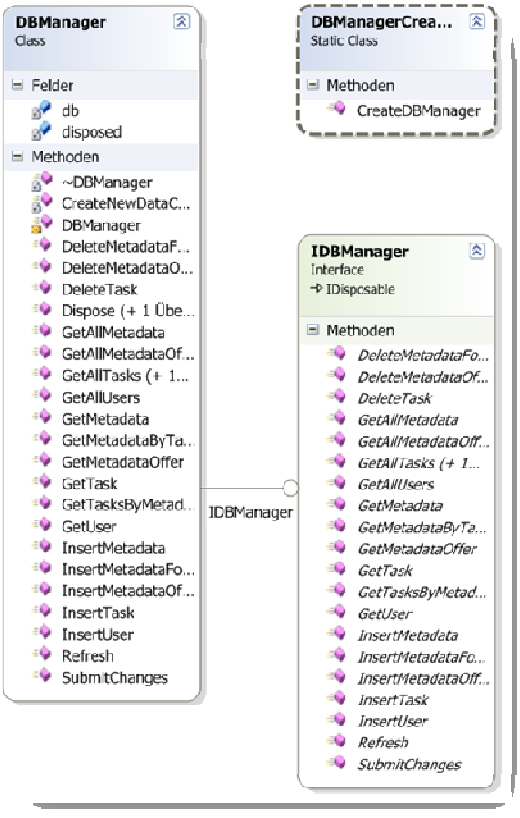
Mit der Verwendung von LINQ wird das Handling der Daten aus der Datenbank einfacher und übersichtlicher, da diese als fertige Objekte bereitgestellt werden. Außerdem wird die Erstellung der Abfragen durch die Verwendung der LINQ-Syntax in C# erheblich erleichtert.

Zusätzlich lassen sich mit dem O/R-Designer von Visual Studio gewünschte Strukturänderungen an der Datenbank leichter in die generierte Objektstruktur von LINQ umsetzen.
Zusätzlich bietet LINQ aber nicht nur Vorteile während der Programmierung und Fertigung der Anwendung, sondern auch zur Laufzeit. In LINQ to SQL ist auch eine Caching-Funktion für die Daten integriert, die die Performance deutlich steigert.

5.3.4 DB-Engine

```
public tbl_metadaten[] GetMetadataByTaskID(int aufgabenID)
{
    lock (this)
    {
        IQueryable<tbl_metadaten> query = from m in db.tbl_metadaten
                                          join am in db.tbl_aufgaben_metadaten on m.ID equals am.metadatenID
                                          where am.aufgabenID == aufgabenID
                                          select m;

        return query.ToArray<tbl_metadaten>();
    }
}
public tbl_metadaten_offer GetMetadataOffer(int id)
{
    lock (this)
    {
        IQueryable<tbl_metadaten_offer> query = from mo in db.tbl_metadaten_offer
                                                where mo.ID == id
                                                select mo;

        return query.First<tbl_metadaten_offer>();
    }
}
public tbl_metadaten_offer[] GetAllMetadataOffer()
{
    lock (this)
```

(DBManager.cs)



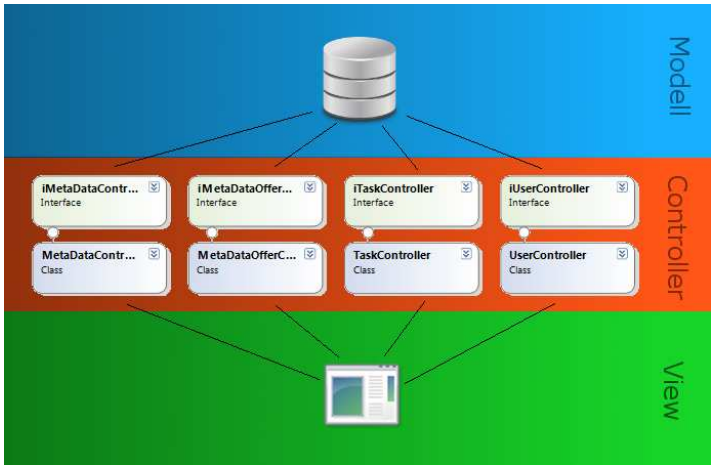
Die Datenbank-Engine ist eine Bibliothek in Visual Studio. Sie ist die Schnittstelle zwischen Datenbank und Applikation.
Dabei sind nur die statische Klasse „DBManagerCreator“ und das Interface „IDBManager“ öffentlich. Die Methode „CreateDBManager im „DBManagerCreator“ gibt eine neue Instanz vom „DBManager“ als „IDBManager“ zurück.
Im „DBManager“ sind alle Methoden, die gebraucht werden, implementiert.

(vollständige Objektstruktur der DBEngine)

5.4 *Frontend*

5.4.1 GUI – MVC

Bei der Implementierung der grafischen Oberfläche (GUI) haben wir uns für das Architektur-Pattern Model View Controller (MVC) entschieden.



Model:
Enthält die Daten aus der Datenbank und stellt diese für den Controller bereit (DBManager). Das Model ist komplett unabhängig vom View und Controller.

Controller:
Greift auf die Daten des Model zu und bereitet sie für den View auf. Der Controller ist abhängig vom Model jedoch nicht vom View.

(Model View Controller Architektur, großes Bild im Anhang)

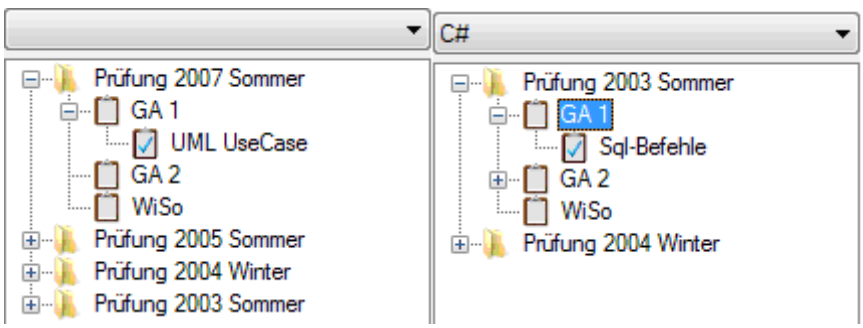
View:
Enthält das komplette GUI Handling / Forms und ruft seine Daten aus dem Controller ab und stellt diese dem Benutzer grafisch da. Der View baut auf dem Controller / Model auf und ist von diesen abhängig.

5.4.2 Startbildschirm

Um eine schnelle und einfache Eingewöhnung für die User zu ermöglichen, wurde eine Windows-nahe Ansicht gewählt. Der TreeView im linken Bereich stellt die Aufgaben – aufgeteilt in die verschiedenen Bereiche dar. Ebenso wurden verschiedene Icons gewählt, um eine deutliche Hervorhebung und Unterscheidung zu ermöglichen.
Der Baumstruktur ist folgendermaßen aufgebaut:

- „Prüfung Jahr Jahresteil“
  - GA1 / GA 2 / WiSo
    - Die Aufgabe

Eine Beispielskizze des Startbildschirms ist im Anhang hinterlegt.



(ohne Filter) *(mit Filter)*

In der Filter/ Combo-Box über dem TreeView kann man mit Hilfe der hinterlegten Meta-Tags alle vorhandenen Aufgaben mit nur zwei Klicks filtern.

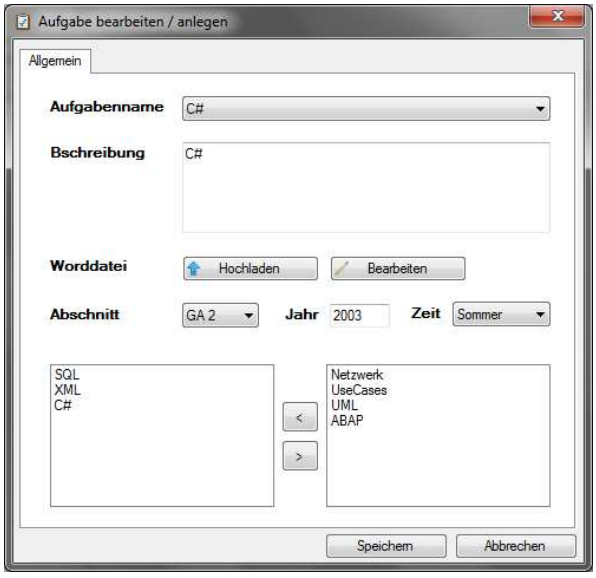
Außerdem kann man sich auch alle vorhanden Metadaten anzeigen lassen, die bereits von einem Administrator freigeschaltet wurden. (Die Bearbeitung von Metadaten findet sich im „MetaDaten Verwalten“-Dialog).
Bei jeder Änderung in der Combo-Box wird der TreeView neu aufgebaut.

Der rechte Bereich (Preview) nimmt den größten Bereich der Oberfläche in Anspruch. In diesem wird die Word-Datei aus der Datenbank als Preview angezeigt. Da die Anwendung auch auf kleineren Bildschirmen/ Netbooks laufen soll, aber längere Word-Dokumente nicht ausgeschlossen werden können, wurde extra eine Scrollbar in die Preview-Funktion integriert.

Für die Ansicht und Darstellung des Kontext-Menüs wurde die Vereinfachte Darstellung gewählt –angelehnt an das Standarddesign des Windows alter View. Auf eine neuartige Ribbon-Oberfläche wurde bewusst verzichtet, da der Aufwand den Nutzen deutlich übersteigt. Der genaue Aufbau des Menüs ist anhand von Screenshots im Anhang aufgezeigt.
Die Status-Bar am unteren Rand soll dem User eine schnelle Übersicht über wichtige Informationen zum Anmelde- und Berechtigungsstatus geben.
Somit wird jeder freie Platz sinnvoll genutzt und alle wichtigen Grundfunktionen sind bereits am Startbildschirm aufrufbar.

Den größten Aufwand bei der Realisierung der Oberfläche war die Umsetzung des asynchronen Ladens der Word-Datei. In der ursprünglichen Version hätte man, aufgrund der langen Ladezeit, einen Ladebalken integrieren müssen.

5.4.3 Aufgabe bearbeiten/ anlegen



Der Aufbau des Dialogs wurde im Windows-Standard Stil aufgebaut (incl Reiter). Außerdem soll der Benutzer den Dialog von oben nach unten abarbeiten können, so dass nichts vergessen wird und kein Überfluss an Informationen ensteht.
Der Dialog öffnet sich stets im Vordergrund und lässt –so lange er aktiv ist- keine Eingabe in der Hauptanwendung zu.
Da bei der Bearbeitung einer bereits bestehenden Aufgabe deren Name nicht mehr verändert werden darf, haben wir uns entschieden dort eine Combo-Box zu verwenden, damit mit problemlos auf andere Aufgaben switchen kann, ohne den Umweg über die TreeView wählen zu müssen.

(Dialog: Aufgabe bearbeiten)

Beim Hochladen einer Aufgabe von der Festplatte in unser System gab es anfangs massive Probleme bei der Kommunikation mit dem Word-Wrapper-Modul. Hier war es zunächst nicht möglich zu prüfen, wie lange der Wrapper braucht um die Datei einzulesen. Da in dieser Zeit keine weitere Eingaben erfolgen dürfen, mussten in diesem Zeitraum alle Dialoge deaktiviert werden.
Die Lösung wird in der Dokumentation des Word-Wrappers veranschaulicht.

Folgende Felder müssen zwingend ausgefüllt werden:

- Aufgabenname (nur im Neu-Anlegen-Modus möglich)
- Word-Datei muss vorhanden sein. Hierbei ist die unterschiedliche Vorgehensweise beim bearbeiten und anlegen zu beachte:
  - Im Neu-Dialog muss genau eine Word-Datei hochgeladen werden
  - Beim Bearbeiten kann eine Neue hochgeladen werden oder eine vorhandene bearbeitet werden
- Abschnitt: GA1 / GA2 / WiSo
- Jahr
- Zeit: Sommer- oder Winterprüfung
- MetaDaten (mindestens ein Tag muss gewählt werden)

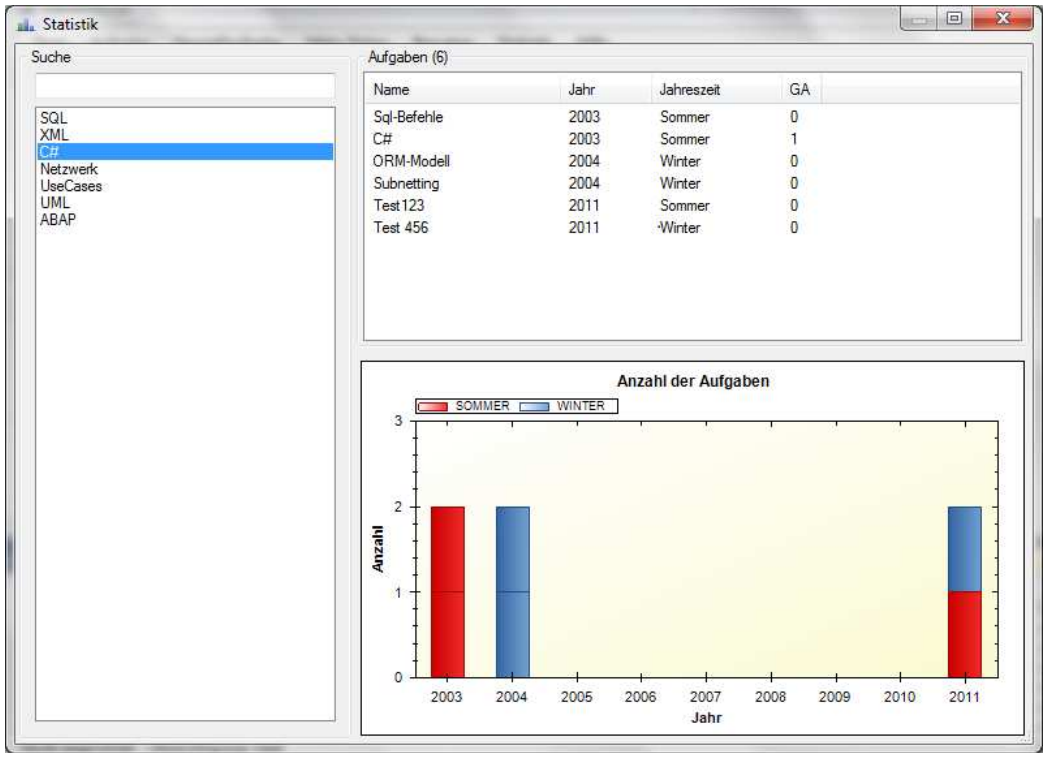
Das Hinzufügen einer Beschreibung ist optional.

5.4.4 Stegreifaufgabe erstellen

Der Aufbau des ExGenerator-Dialogs (Screenshot, siehe Anhang) wurde an das MainView angelehnt. Um alle Aufgaben auf einem Blick zu haben, wurde die reine TreeView durch eine Check-Box-List ausgetauscht, um einerseits alle Aufgaben auf einen Blick zu haben, aber andererseits diese auch sofort selektieren zu können. Somit kann man unnötige Informationen ausblenden.
Außerdem wurde auf eine sofortige Erstellung der Word-Preview verzichtet, da sonst der Dialog –aufgrund der Ladezeit des Worddokuments- ständig hängen und laden würde.

5.4.5 Statistik und Auswertung

Die statistische Auswertung aller vorhandenen Aufgaben erfolgt auf Basis der eingetragen Meta-Tags. Dies ist vor allem für Abschlussprüfungen interessant, um Schwerpunkte zu erkennen und deren Verschiebung.



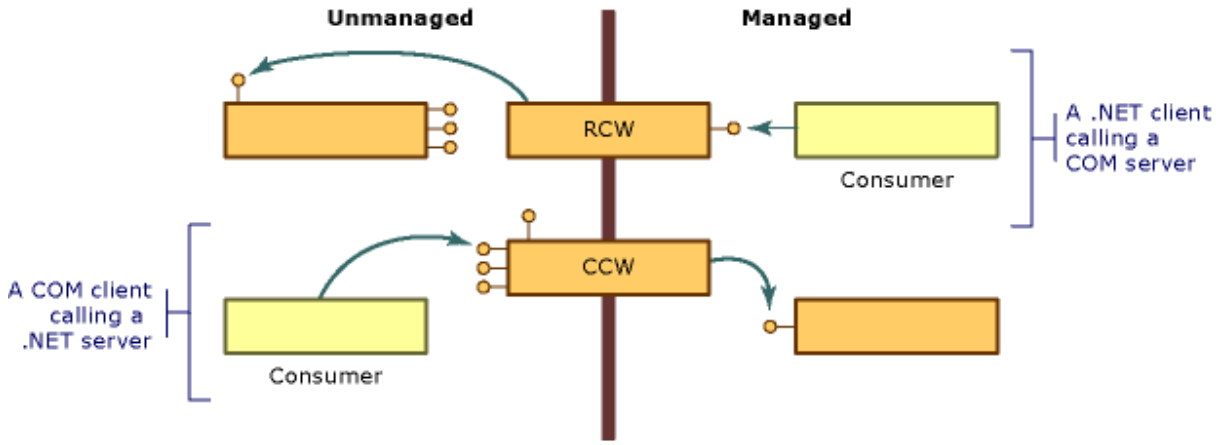
Die Statistiken sind im Hauptmenü der GUI implementiert. Nach dem Anklicken der Statistikfunktion öffnet sich ein neues Fenster, welches grob in vier Bereiche aufgeteilt ist.
Die linke Hälfte besteht oben aus einer TextBox und darunter einer ListBox. In diese ListBox werden alle bekannten Metatags aufgelistet, welche mit der Suchfunktion in der TextBox auch vorgefiltert werden können (Suchfunktion).
Mit einem Doppelklick auf das gewünschte Tag oder dem Button <<Statistik anzeigen>> wird automatisch eine aktuelle Statistik in der rechten Fensterhälfte neu generiert. Rechts oben findet sich eine Übersicht in einer ListView, wann und wo der gesuchte Tag in einer Prüfungsaufgabe verwendet wurde. Darunter wird mit Hilfe des Benutzersteuerelements ZedGraphControl ein Balkendiagramm erzeugt, in dem Sommer- und Winterprüfung, die Häufigkeit und die Jahreszahl berücksichtigt werden. ZedGraphControl ist ein in C# geschriebener Controler, aus dem man Methoden aufrufen kann um Statistiken anzuzeigen.

Die größte Herausforderung war, die statistischen Daten dynamisch für jede Metadaten zu generieren. Probleme traten hier vorwiegend bei der Bereinigung auf, bevor man neue Statistiken erstellen und anzeigen konnte. Ein Ausschnitt des Codes mit der (markierten) Bereinigungsfunktion befindet sich im Anhang.

6. Technische Umsetzung des „Word Wrapper“

5.1 .NET COM Interop

Da die Automation API von Office (und daher auch von Word) auf COM aufsetzt, muss zwischen der verwalteten Umgebung (.NET) und der unverwalteten Umgebung (COM) kommuniziert werden. .NET bietet hierzu mehr oder weniger komfortable Mechanismen an. Ein COM-Objekt implementiert unterschiedliche Interfaces. Jedes COM-Objekt implementiert mindestens IUnknown. Über diese Interfaces wird auf ein COM-Objekt zugegriffen, um nun mit C# auf solch ein Objekt zuzugreifen muss die Interfacedeklarationen bekannt sein. Dazu muss in C# das entsprechende managed Pedant zum Unmanaged deklariert werden. Dabei müssen die Signaturen der managed und unmanaged Schnittstellen genau übereinstimmen. Diese Schnittstellen legen fest wie mit einem Objekt kommuniziert werden kann. Nun muss nur noch dieses Objekt erzeugt werden um es zu benutzten. Die .NET Laufzeit stellt COM-Objekte über Proxy Objekte zur Verfügung. Es gibt zwei generelle Proxytypen einmal den sog. Runtime Callable Wrapper (RCW) und den COM Callable Wrapper. Ein RCW wirkt wie ein „normales“ .NET Objekt, übersetzt und leitet die Methode Aufrufe an das COM-Objekt weiter. Ein CCW wirkt genau umgekehrt und leitet daher die Methoden Aufrufe an die .Net Laufzeit weiter.



(Schematischer Aufbau von „COM Interop“)

5.2 *Genereller Aufbau des Word Wrapper Modules*

Das WordWrapper Modul ist eine Abstraktion der Word COM API. Es erleichtert die Benutzung der Word Funktionen und kapselt sämtliche Logik die das Anzeigen, Bearbeiten und Verwalten der Word Dateien ausführt in sich. Das Modul besteht aus wenigen öffentlichen Typen, die zentrale Rolle spielt die WordExchanger Klasse. Sie stellt eine Fassade da welche die weitern Subklassen versteckt. Die Dokumente liegen als Byte-Stream in der Datenbank ab und werden an den WordExchanger übergeben. Dieser ruft den TempController auf welcher den Byte-Stream als Datei in das temporäre Verzeichnis des KamAnalyser abspeichert. Der Pfad zu einem Dokument wird an den WordController übergeben, dieser ruft die entsprechenden API Aufrufe von Word auf und erstellt eine Wordinstanz mit diesem Dokument.

5.3 IDocument

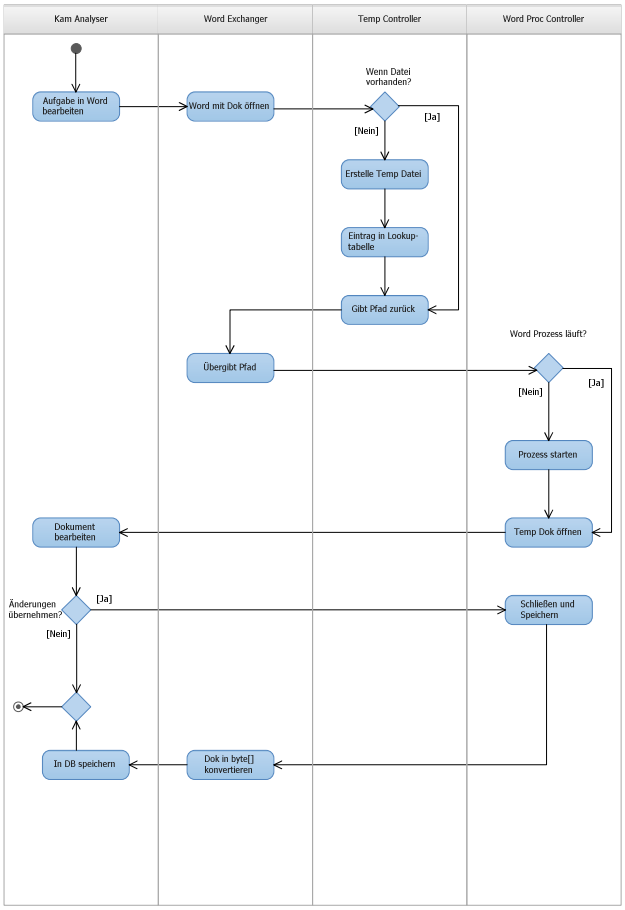
IDocument ist das öffentliche Bindeglied zwischen dem WordWrapper und der restlichen Software. Zum internen Datenaustausch wird der DocumentItem Typ verwendet, welcher eine einfach Implementierung des Interfaces darstellt. Über den WordExchanger werden diese Objekte erzeugt.

```
public interface IDocument
{
    System.Drawing.Image PreviewImage { get; }
    byte[] DocumentData { get; }
    string Path { get; }

    void ReloadData();
}
```

(Deklaration „IDocument“)

5.4 *Überwachen der Dateiänderungen*

Ein erheblicher Aufwand bestand darin Dokumente automatisch mit Word zum Bearbeiten zu öffnen. Prinzipiell muss nur das Dokument aus der Datenbank auf die Festplatte abgelegt werden. Danach wird Word mit dem abgespeicherten Dokument gestartet. Wird das Dokument wieder geschlossen müssen die Änderungen wieder in die Datenbank eingelesen werden. Die ersten Ablaufpunkte stellen kein Problem dar, da die bereits bestehenden Funktionalitäten aufgegriffen werden konnten. Die Word API bietet jedoch nur Events an welche das Schließen der Datei signalisiert, jedoch nicht, wann das Dokument geschlossen wurde. Es ist wichtig, dass das Dokument geschlossen wurde, da sonst jeglicher Zugriff auf die Datei fehlschlagen würde, da Word einen exklusiven Filelock auf die momentan geöffneten Dokumente hält. Daher können die Änderungen nicht mehr in die Datenbank eingelesen werden, solang nicht bekannt ist, wann die Datei wieder freigegeben wurde.



Im ersten Schritt wurde die Überwachungslogik in eine eigene Klasse gekapselt: DocumentGuardObject. Ein DocumentGuardObject wird durch einen Aufruf von WinWordProcControler.ViewDoc() erzeugt. Dieses löst einen Event aus sobald das Dokument geschlossen wurde. Der KamAnalyser meldet sich auf diesen Event an und liest die Daten wieder in die Datenbank.

(Ablaufdiagramm: Daten Änderrung, große Ansicht im Anhang)

Um diesen Event auszulösen wurden einige Ansätze im Team besprochen und getestet, wobei nur ein Ansatz umsetzbar war.
Zuerst wurde versucht den Prozess der durch COM erzeugt wurde zu überwachen, dies war nicht möglich da der Prozess erst nach einem Timeout beendet wurde nachdem der COM-Server beendet wurde, des weiteren wird der COM-Server erst beendet wenn der managed Proxy vom Garbage Collector aufgesammelt wird, dies ist wieder nicht vorhersehbar und führt zu Zeitverzögerungen.
Der nächste Schritt bestand darin den Word Prozess manuelle zu starten, die .NET Klasse Process bietet einen ProcessClosed Event an, welcher wiederum den DocumentClosed Event des DocumentGuardObject auslösen könnte. Dies war jedoch nicht möglich da Word mehrere Word Instanzen in einen Prozess zusammen führt. Daher ist die Annahme, dass jedes Dokument einen eigen Prozess hat falsch und daher der Event auch nicht ausgelöst wird. Da der Word Prozess nicht beendet solang nicht alle Dokumente geschlossen wurden.

Die endgültige Lösung bestand darin kontinuierlich den Zugriff auf die Datei zu prüfen und sobald dieser erlaubt wurde, wird angenommen, dass die Datei auch geschlossen wurde. Die programmierte Routine erfüllt zwar die Anforderungen ist aber von der Code Qualität und Stabilität nicht perfekt. Solang die Datei nicht freigegeben wird läuft die Routine eine

Endlosschleife durch, da diese jedoch asynchron abgearbeitet wird, ist die Oberfläche nicht blockiert. Desweiteren können nicht vorhersehbare Nebeneffekte auftauchen wie z.B. COM-Timeout oder fehlgeschlagene RPC Aufrufe. In unseren Test traten diese Probleme jedoch nur sehr selten auf und stellen aus Anwendersicht kaum ein Problem dar.

5.5 *Word Preview Control*

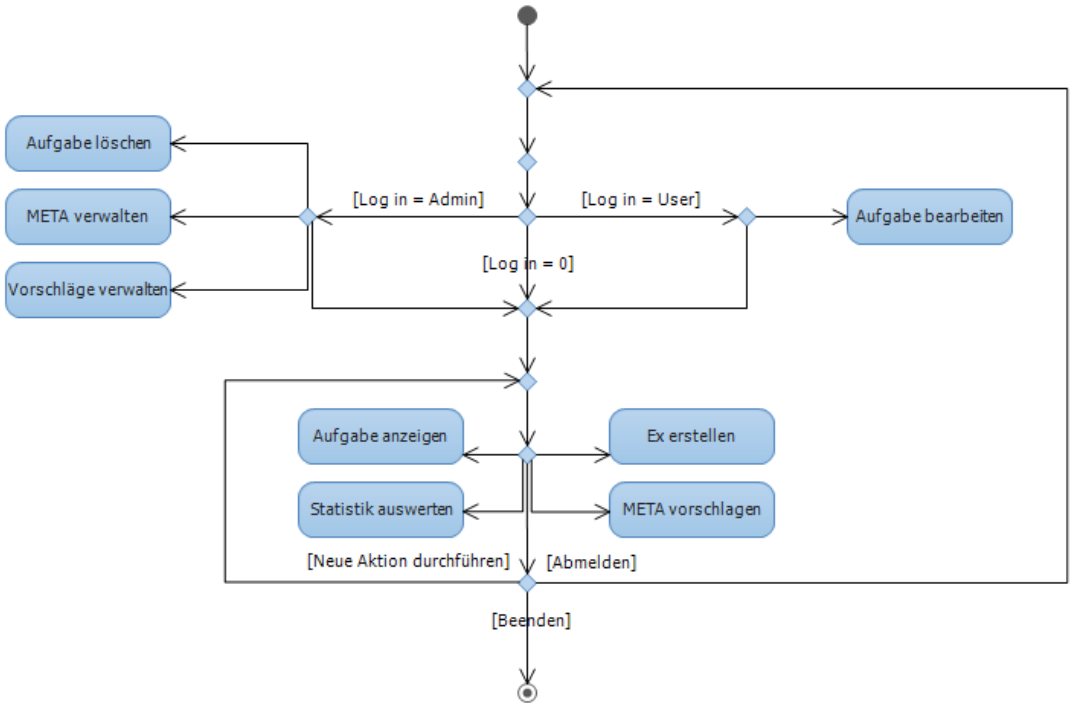
Der erste Ansatz das Preview Control zu Realisieren war eine Vorschau mit einem Enchaned Metafile das als Bitmap dargestellt wird. Dieses Enchaned Metafile wir dabei über die Word COM API geladen und dann erst in ein Bitmap umgewandelt und an das Frontend übergeben.
Allerdings wurde schnell klar, dass dieser Lösungsansatz viele Nachteile mit sich bringt!
Die Vorschau hatte eine sehr schlechte Qualität, und die Konvertierung war sehr Rechenaufwendig. Außerdem ist so kein Copy & Paste möglich da es sich bei der Darstellung nur um ein Bild handelt.

Im Team wurde dann besprochen das diese Lösung lediglich als „Fallback Lösung“ eingesetzt werden kann, und wir für den normalen gebrauch eine neuen Lösungsansatz benötigen.
Der nachfolgende Quellcode zeigt wie die „Fallback Lösung“ implementiert ist.

```
wDoc.Range(start, end).Select();

byte[] metafileBytes = wDoc.Application.Selection.EnhMetaFileBits;

using (System.IO.MemoryStream wmfStream = new System.IO.MemoryStream(metafileBytes))
{
    metafile = (Metafile)Metafile.FromStream(wmfStream);
}

bitmap = new Bitmap(metafile.Width, metafile.Height);
using (Graphics g = Graphics.FromImage(bitmap))
{
    g.DrawImage(metafile, new Point());
}
```

(Quellcodeauszug der „Fallback“ Lösung)

Als neuen Lösungsansatz haben wir uns dann auf eine Preview mittels COM Objekt von Word geeinigt. Dieser ist generell der gleiche Mechanismus wie es der Windows Explorer verwendet. Was die Realisierung angeht wird auf ein COM Objekt von Word zugegriffen welches das öffentliche IPreviewHandler Interface implementiert. Diese bindet nun eine MS Word Instanz ein und erzeugt auf diese Weise eine Vorschau mit viel besserer Qualität, weniger Rechenaufwand und außerdem wird so das Feature Copy & Paste ermöglicht. Für den Fall das kein Word auf dem Client Rechner Installiert ist wird automatisch unsere BITMAP Lösung verwendet.
Zur Veranschaulichung dieser Problematik wurden passende Screenshots im Anhang hinterlegt.



(Aktivitätsdiagramm: genereller Ablauf)

7. Installer

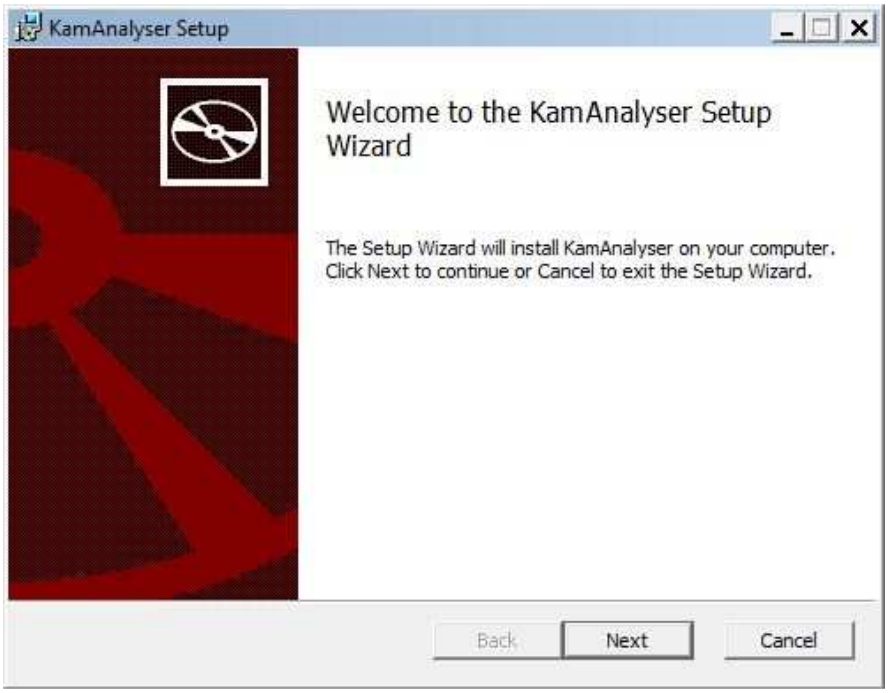
7.1 Server-Setup

Das Server-Setup beinhaltet eine eigenständigen C#-Applikation, welche ein aktuelles Abbild unserer Datenbank auf dem gewünschten Zielsystem installiert. Außerdem wird die Möglichkeit geschaffen, alle Daten aus vorhandenen Tabellen zu migrieren.

Die Applikation selbst wird mit Hilfe der „Setup-Language“ WIX in ein MSI gepackt, die auf das Zielsystem kopiert werden kann. Der Inhalt des MSI-Paketes enthält alle Daten der Applikation und das aktuelle Abbild der Datenbank. Das WIX-Script wurde so aufgebaut, dass direkt nach der Installation die Applikation ausgeführt und die Datenbank initialisiert werden kann. Der Installationsvorgang wird über eine Konsole gestartet, in welcher alle gewünschten Parameter mitgegeben werden können (z.B. SQL-Server).

Nach dem Fertigstellen ist die aktuellste Version der Datenbank auf dem Zielsystem funktionsbereit vorhanden und kann vom KamAnalyser verwendet werden.

7.2 Installer für Clients



(Installer-Skizze, weitere Skizzen im Anhang)

Bei der Realisierung dieses WIX-Skriptes musste darauf geachtet werden, dass alle DLL-Dateien und die entsprechenden Verweise unseres Projekts mit eingebunden werden.
Nachdem das WIX-Script incl. Mondo abgeschlossen war musste nur noch eine Build-Definition für unseren Setup definiert werden. Diese Definition konnte aber ganz einfach über Visual Studio Konfiguriert und getestet werden.
Als kleines Feature haben wir noch eine Build-Definition mit „Code Analyses“ angelegt um eine genau Auswertung des Build-Vorgangs zu erhalten. Weiterhin bietet sich der Vorteil das bei fehlerhaften Builds immer einen „Fehler Auftrag“ im TFS erstellt wird. Somit kann der Fehler schneller erkannt und behoben werden. Zusätzlich wird die Lösung des Problems dort direkt dokumentiert. So wird durch das ineinander greifen der einzelnen Komponenten ein perfekter Buildprocess und eine vollständige Dokumentation gewährleistet.

8. Projektabschluss

Abschließend kann ein positives Gesamtresümee gezogen werden, jedes Teammitglied konnte sich gut in die verwendete Technologie einarbeiten. Die aufgetretenen Probleme führten nicht dazu, dass das Projektziel gefährdet wurde. Jedes Teammitglied ist mit dem Endergebnis sehr zufrieden, im Nachhinein würde niemand großartige Unterschiede in einer Neuimplementierung machen.
Die Word API stellte sich im Nachhinein als nicht so flexible heraus wie erwartet, die aufgetauchten Probleme konnten zwar gelöst werden, rückblickend sollte jedoch eine andere Technologie in Betracht gezogen werden.
Nach einer gewissen Einarbeitungszeit in die s.g. Datenebenenanwendung (DAC) von Visual Studio vereinfacht LINQ to SQL (L2S) den Entwicklungsaufwand erheblich. Bei Änderungen an der Datenstruktur in der Datenbank müssen diese Änderungen ebenfalls in das L2S Objektmodell übernommen werden. Da Datenebenenanwendung erst in Visual Studio 2010 eingeführt wurden, sind diese nur für SQL Server 2008 verfügbar. Eine Abwärtskompatibilität auf z.B. SQL Server 2005 ist nicht vorhanden.

Glossar

Begriff	Beschreibung
Hier wird ein Glossar geführt…	

Anlagenverzeichnis

I. Speichertechnologien

Bilder

Die einzelnen Aufgaben sollten als Grafikdatei abgelegt werden. Die Erstellung der Grafiken können im Programmumfeld nur schwer realisiert werden. Daher müssten diese mit externen Tools erstellt werden. Ein weiterer Nachteil ist, dass Grafiken statisch sind. Das heißt sie können nachträglich nicht mehr geändert werden, sondern müsste dann jedes Mal neu erstellt werden. Somit würde hier ein hoher Wartungs- und Verwaltungsaufwand durch den Auftraggeber im laufenden Betrieb gewährleistet werden.

HTML

Hypertext Markup Language ist eine textbasierte Auszeichnungssprache zur strukturierten Darstellung von Inhalten wie Texten und Bildern in Dokumenten. HTML-Files sind die Grundlage des World-Wide-Webs und können von einem Webbrowser dargestellt werden. Dem Text wird darin durch Auszeichnungen (sog. Tags) von Textstellen eine Struktur hinterlegt. Die Auszeichnung erfolgt durch SGML-Elemente. Die meisten dieser HTML-Elemente werden durch ein Tag-Paar markiert, das heißt durch einen Starttag und einen Endtag. Um solche Dokumente zu erstellen benötigt der Benutzer gute Kenntnisse über HTML. Selbst mit einem Editor benötigt er gewisse Grundlagen. Dies ist für einen Laien nicht zumutbar.
Ein weiteres Problem von HTML tritt dann auf, wenn eine Aufgabe Bildelemente beinhaltet. Standardmäßig können diese nicht direkt in HTML abgelegt werden und müssten in einer separaten Datei gespeichert werden. Eine weitere Möglichkeit wäre die Bilder mittels Parsern in Base64 zu encodieren. Somit könnten die Bilder zwar im HTML-Dokument eingebunden werden, allerdings wäre für die Parser ein zusätzlicher Aufwand oder der Einsatz externer Tools notwendig.

XML

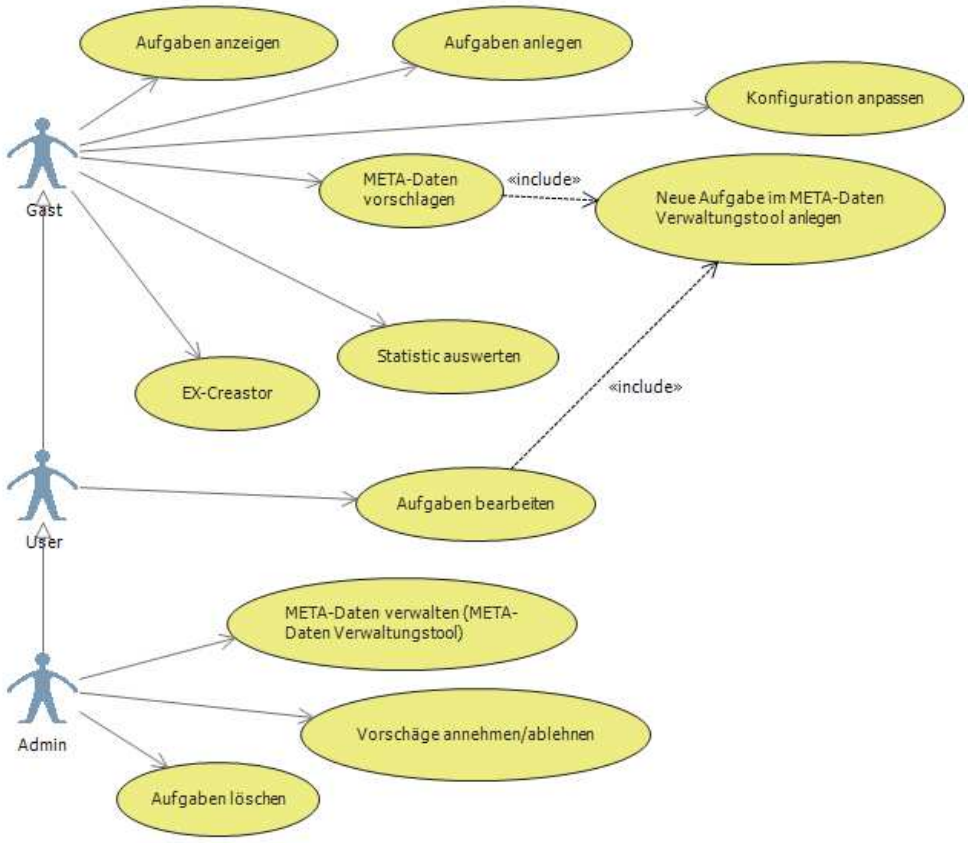
Extensible Markup Language (engl. für „erweiterbare Auszeichnungssprache“), abgekürzt XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten. XML wird u.a. für den Plattform- und Implementationsunabhängigen Austausch von Daten zwischen Computersystemen eingesetzt, insbesondere über das Internet. XML benötigt etwas mehr Verständnis als HTML, da die Dokumente nur die Daten, also den reinen Inhalt beschreiben ohne Formatierung oder optischer Hilfe. Wie erwähnt beschreibt XML nur die Struktur von Daten, unser Programm müsste also das Formatieren und Anzeigen der Daten komplett -mit großem Aufwand- selbst implementieren. Einfacher als bei HTML wäre dagegen die Einbindung von base64-codierten Bildern in den XML-Files.

Word Dateien

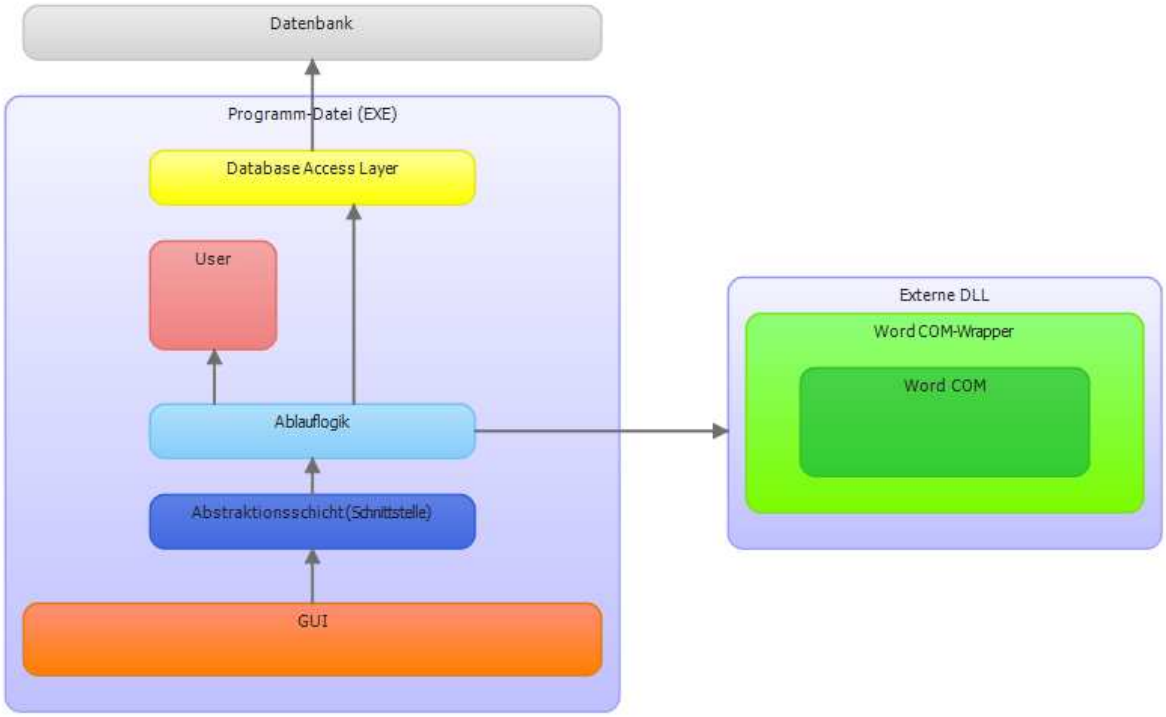
Die endgültige Entscheidung fiel auf die Verwendung von Word Dateien. Der größte Vorteil ist sicherlich der große Bekanntheitsgrad von Microsoft Word. Für die meisten Lehrkräfte zählt der Umgang mit Office zum täglichen Geschäft. Daher sollte die Erstellung und Bearbeitung von Aufgaben im Word-Format –ohne Schulungsaufwand- möglich sein, da auf vorhandenes Grundwissen zurückgegriffen werden kann. Ein weiter Vorteil ist das eine .docx Datei als Container Datei fungiert, es können Bilder eingefügt werden ohne, dass diese separat auf der Festplatte abgelegt werden müssen.
Nachteilig müssen nur die häufigen Versionswechsel betrachtet werden. Um den Programmieraufwand in Grenzen zu halten, muss sich der Auftraggeber auf die Verwendung von Office 2007 und 2010 beschränken. Die Kompatibilität zukünftiger Office-Versionen kann nicht garantiert werden.

In der Datenbank wird das File anschließend im Binärcode abgespeichert. Somit können neben dem Inhalt auch alle Formatierungen übernommen werden.

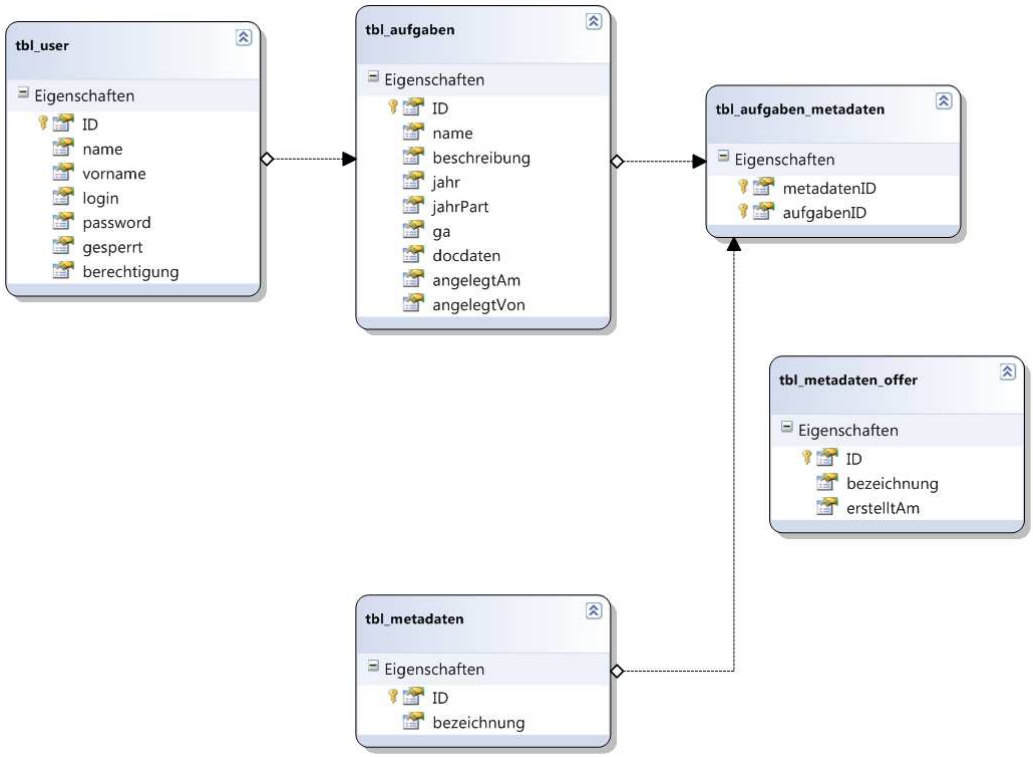
II. UseCase: Planungsphase



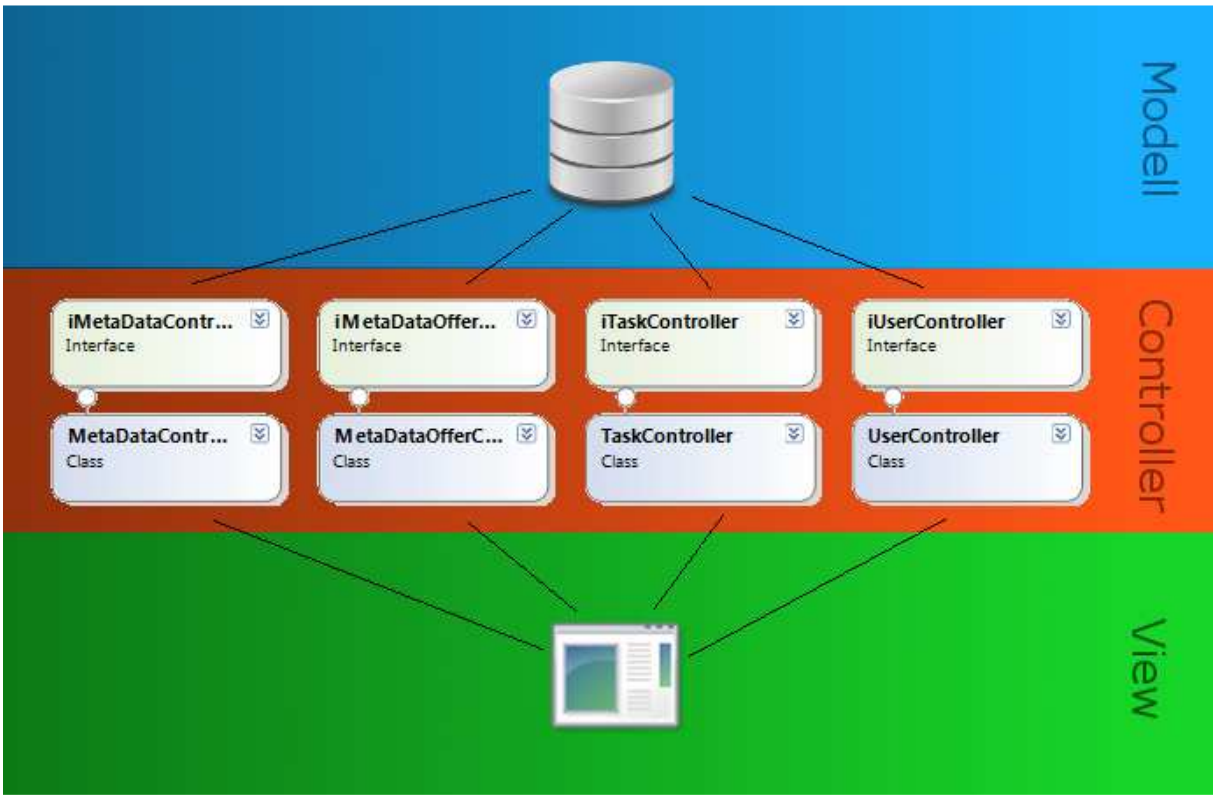
III. Layerdiagramm



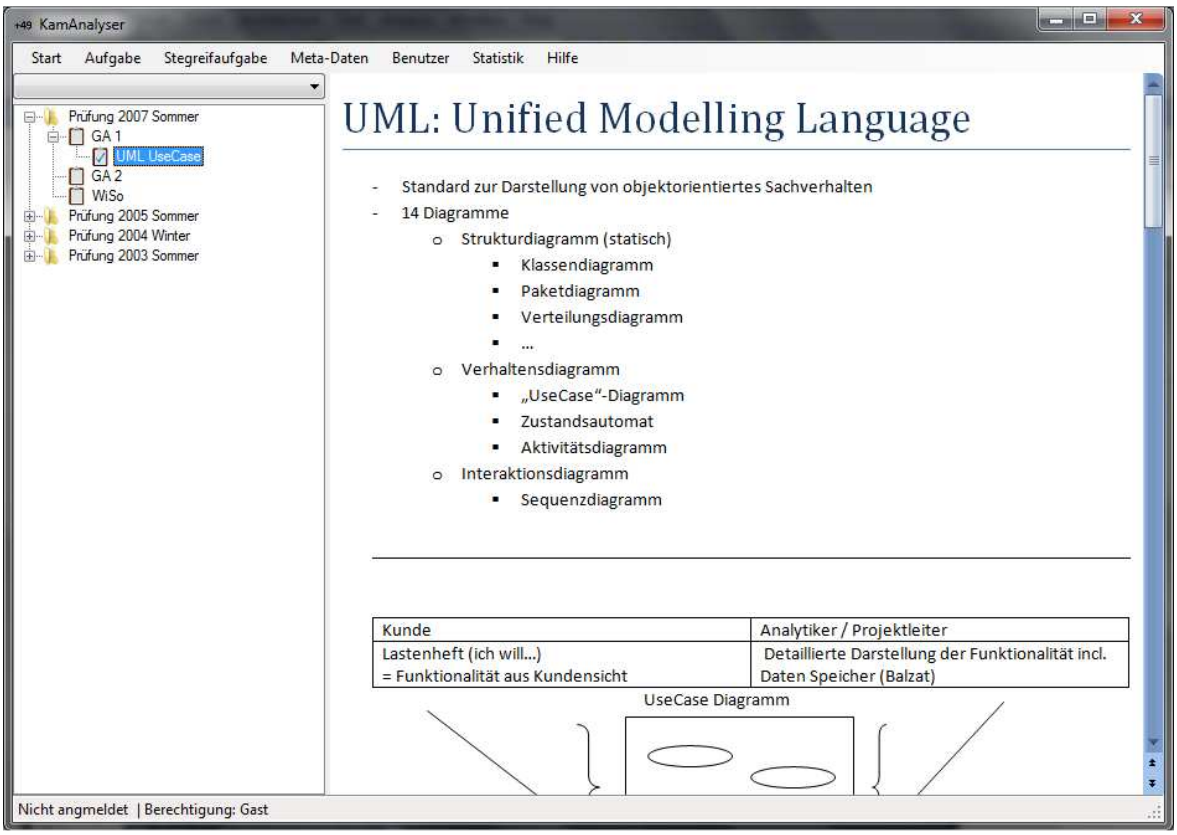
IV. Datenbankmodel



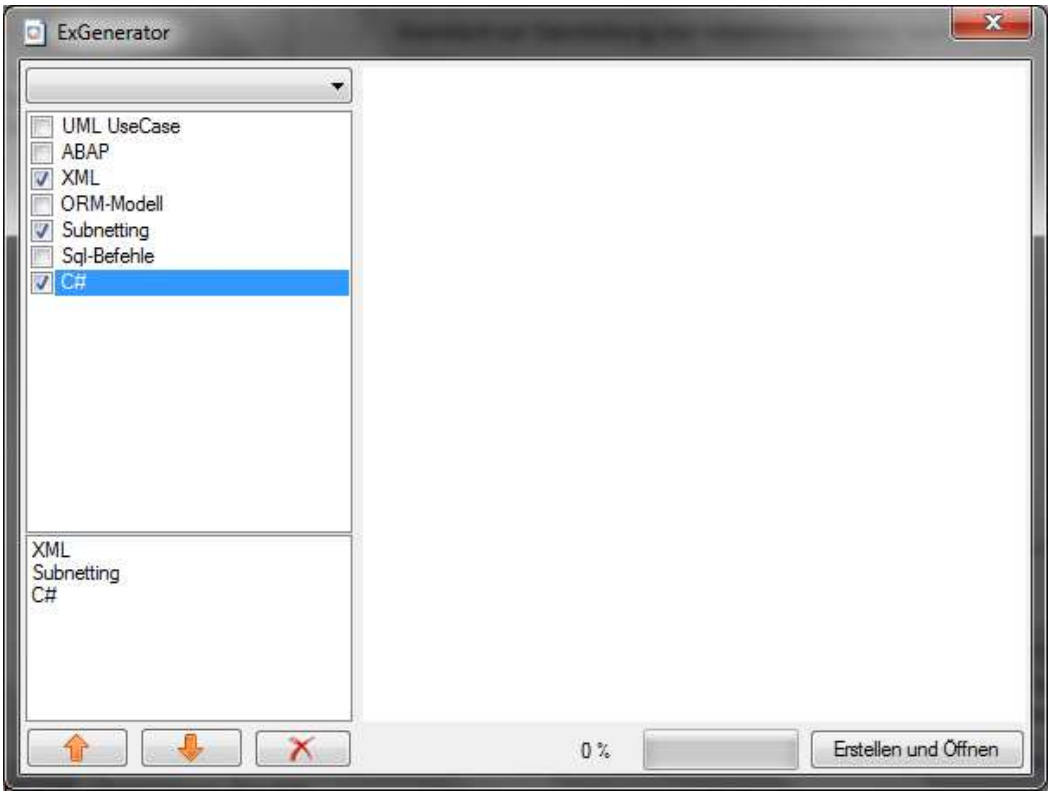
V. Model View Controller



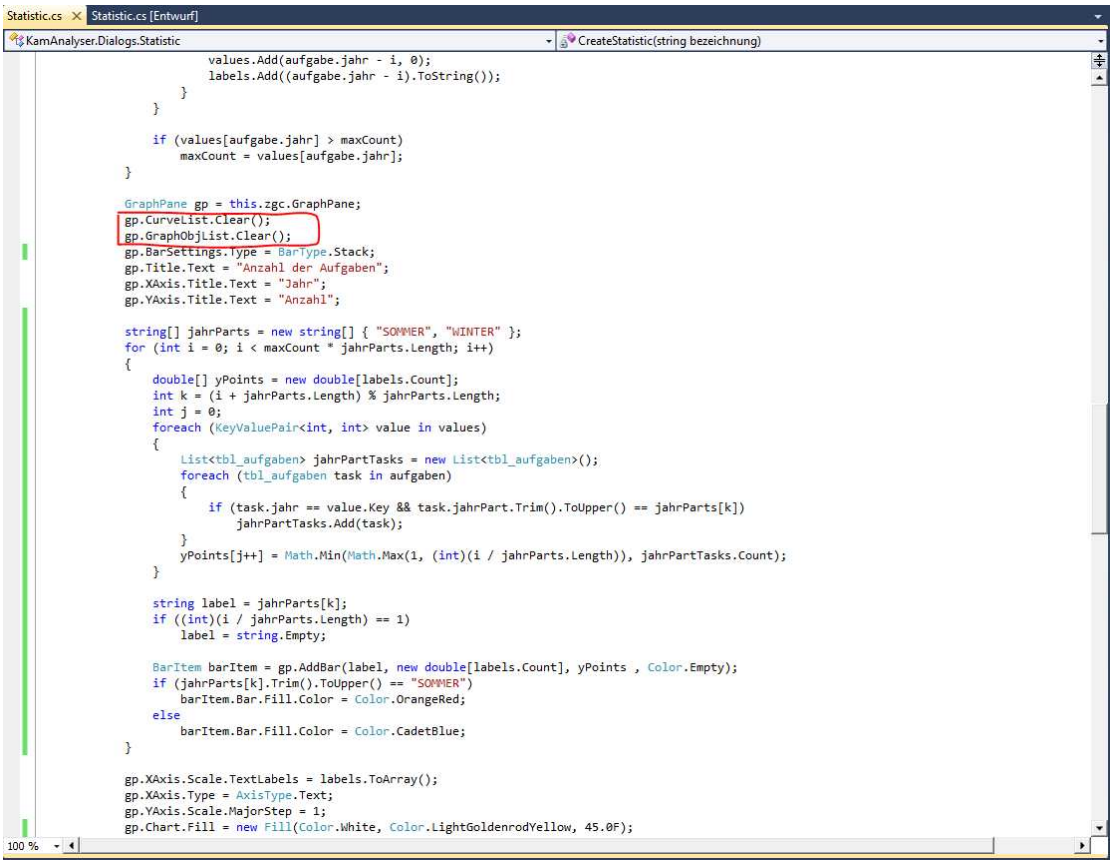
VI. Startbildschirm



VII. ExGenerator

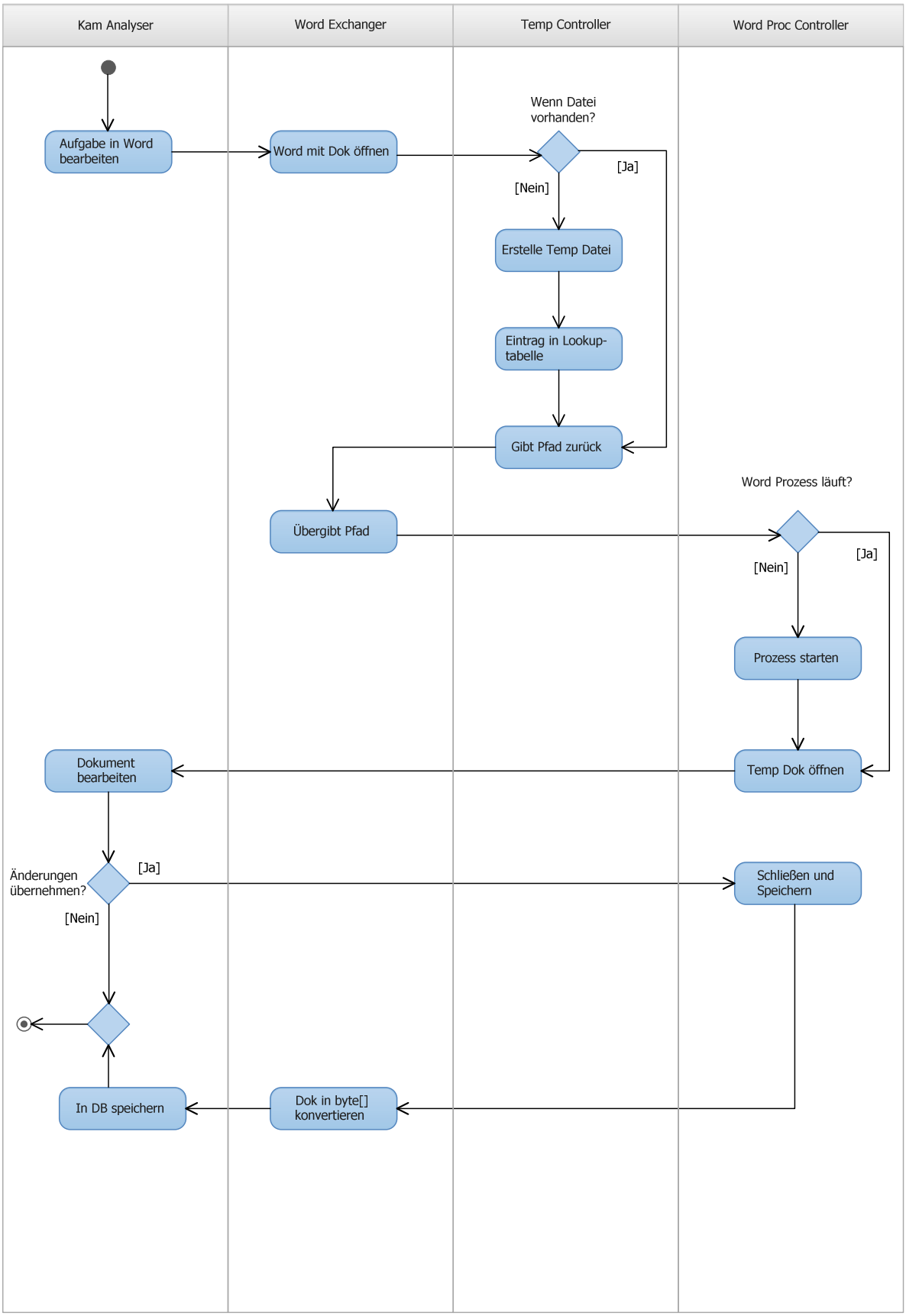


VIII. Statistik.cs mit Bereinigungsfunktion

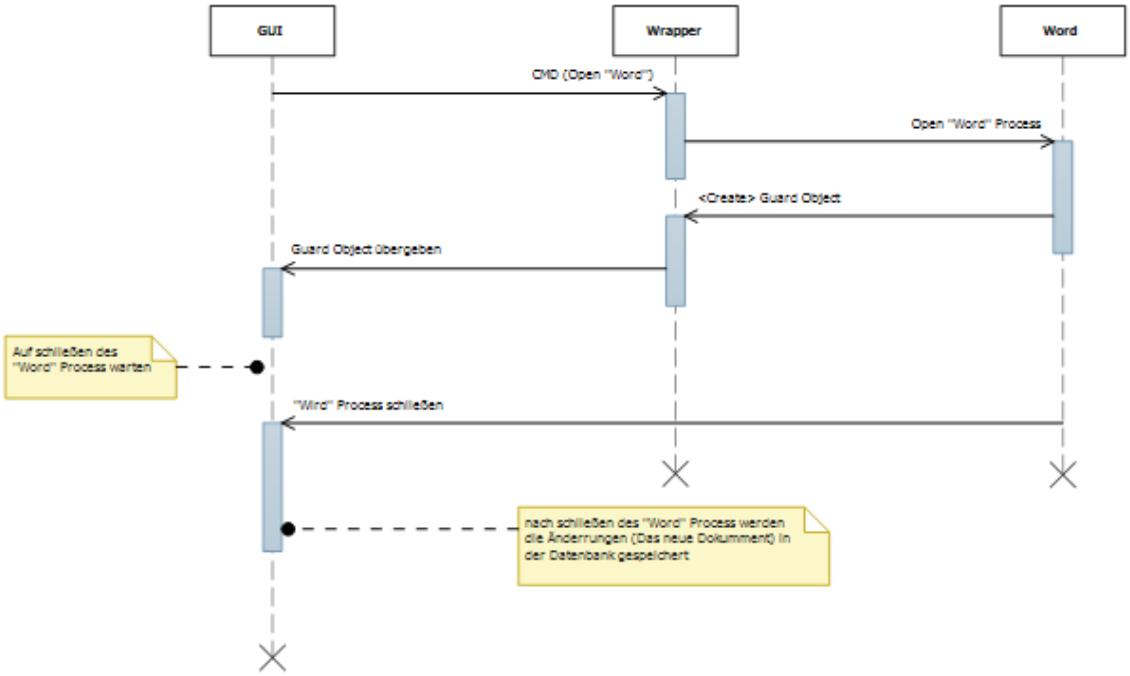


```
Statistic.cs | Statistic.cs [Entwurf]
KamAnalyser.Dialogs.Statistic | CreateStatistic(string bezeichnung)

                    values.Add(aufgabe.jahr - i, 0);
                    labels.Add((aufgabe.jahr - i).ToString());
                }
            }

            if (values[aufgabe.jahr] > maxCount)
                maxCount = values[aufgabe.jahr];
        }

        GraphPane gp = this.zgc.GraphPane;
        gp.CurveList.Clear();
        gp.GraphObjList.Clear();
        gp.BarSettings.Type = BarType.Stack;
        gp.Title.Text = "Anzahl der Aufgaben";
        gp.XAxis.Title.Text = "Jahr";
        gp.YAxis.Title.Text = "Anzahl";

        string[] jahrParts = new string[] { "SOMMER", "WINTER" };
        for (int i = 0; i < maxCount * jahrParts.Length; i++)
        {
            double[] yPoints = new double[labels.Count];
            int k = (i + jahrParts.Length) % jahrParts.Length;
            int j = 0;
            foreach (KeyValuePair<int, int> value in values)
            {
                List<tbl_aufgaben> jahrPartTasks = new List<tbl_aufgaben>();
                foreach (tbl_aufgaben task in aufgaben)
                {
                    if (task.jahr == value.Key && task.jahrPart.Trim().ToUpper() == jahrParts[k])
                        jahrPartTasks.Add(task);
                }
                yPoints[j++] = Math.Min(Math.Max(1, (int)(i / jahrParts.Length)), jahrPartTasks.Count);
            }

            string label = jahrParts[k];
            if ((int)(i / jahrParts.Length) == 1)
                label = string.Empty;

            BarItem barItem = gp.AddBar(label, new double[labels.Count], yPoints , Color.Empty);
            if (jahrParts[k].Trim().ToUpper() == "SOMMER")
                barItem.Bar.Fill.Color = Color.OrangeRed;
            else
                barItem.Bar.Fill.Color = Color.CadetBlue;
        }

        gp.XAxis.Scale.TextLabels = labels.ToArray();
        gp.XAxis.Type = AxisType.Text;
        gp.YAxis.Scale.MajorStep = 1;
        gp.Chart.Fill = new Fill(Color.White, Color.LightGoldenrodYellow, 45.0F);

100 %
```

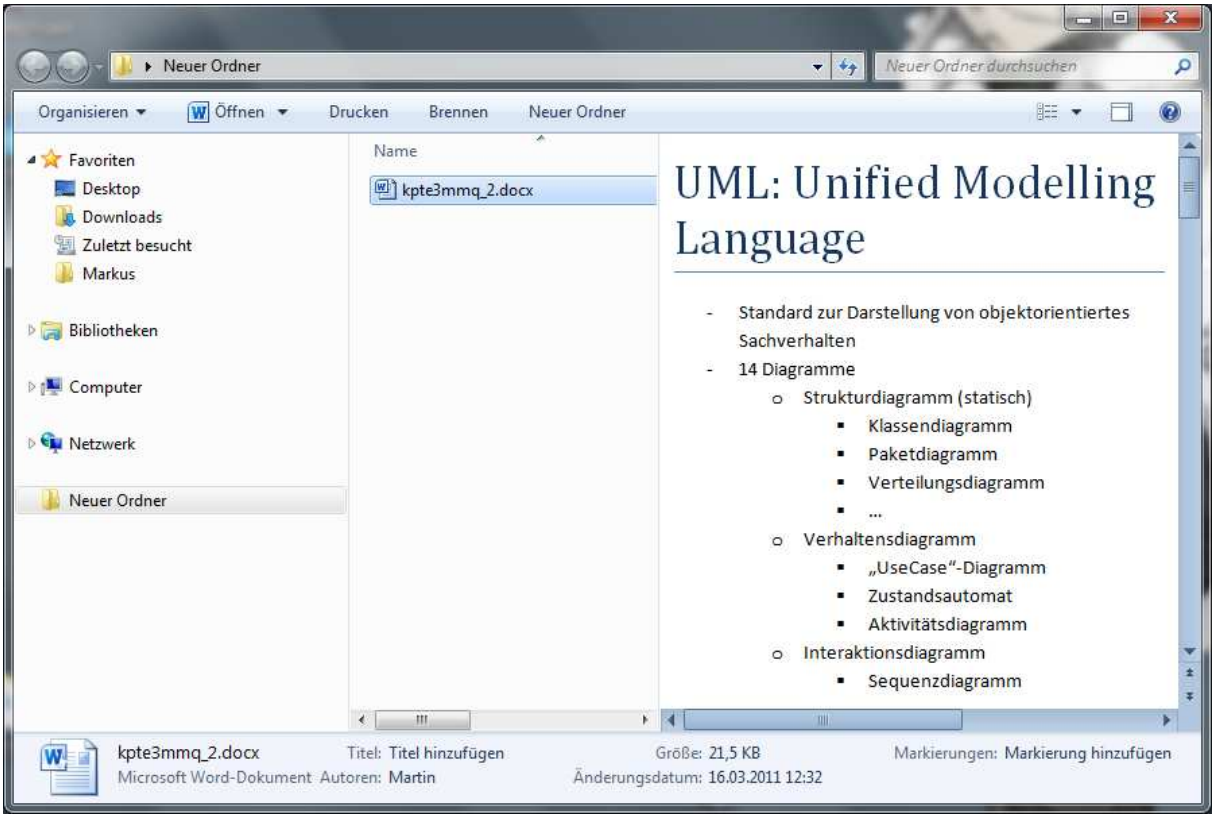
IX. Ablaufdiagramm



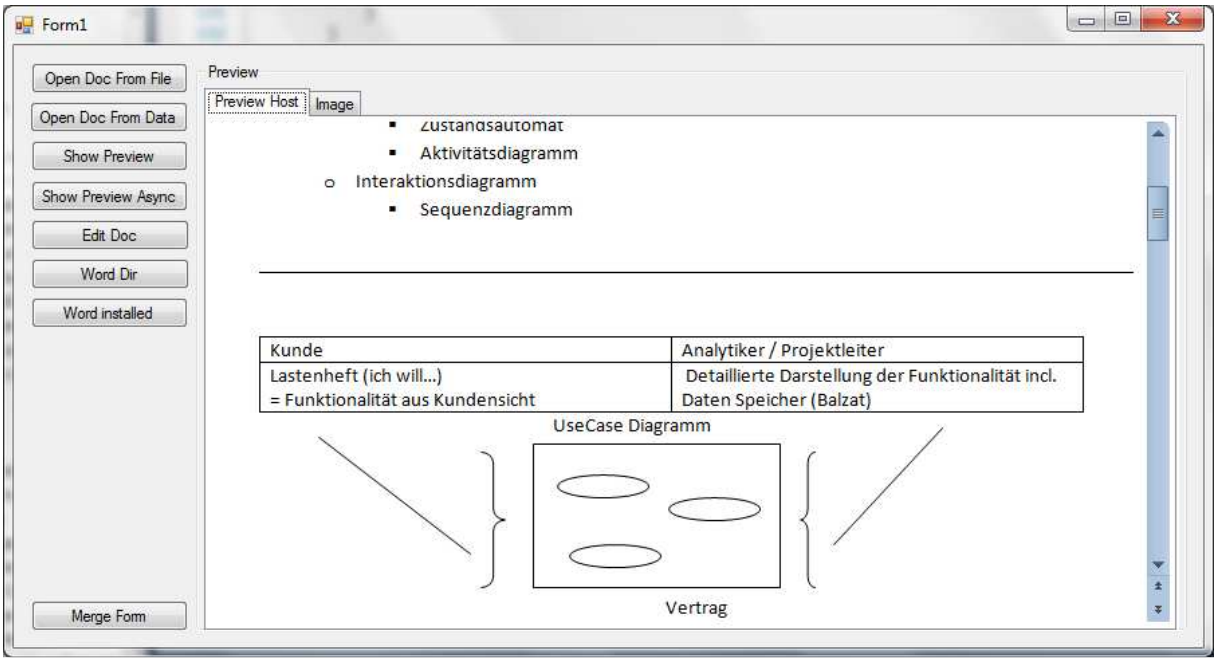
X. Sequenzdiagramm: Dateiüberwachung



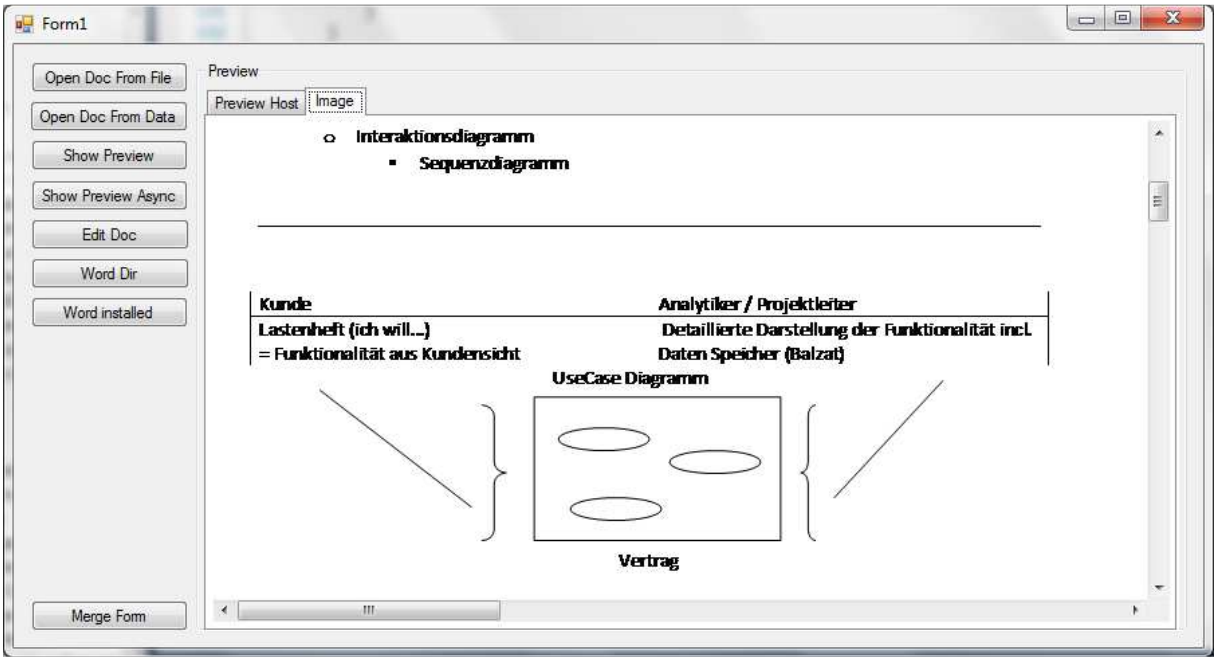
XI. Word Preview Control - Skizzen



(Windows 7 Explorer mit Preview Pane)

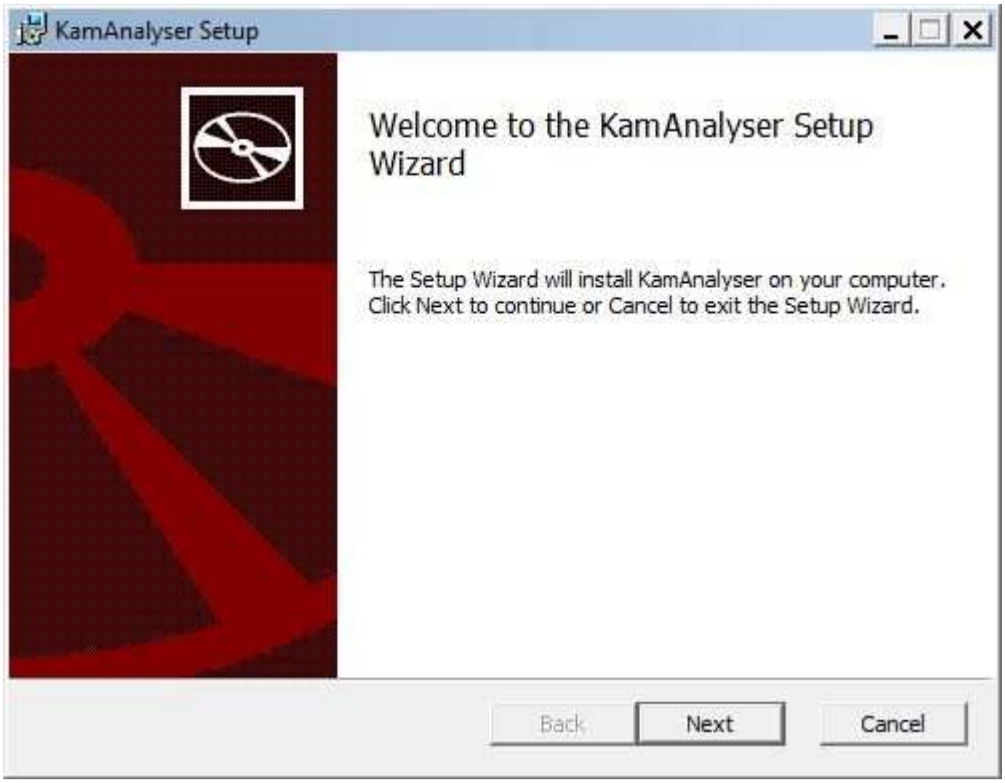


(Testclient mit Preview Pane)



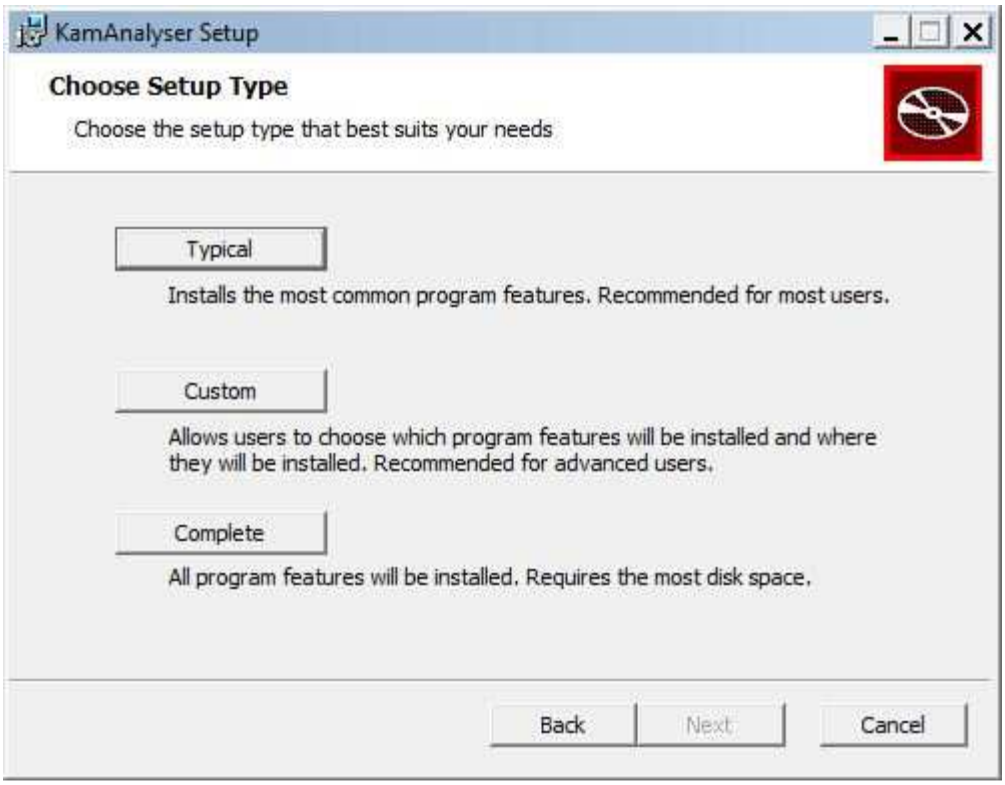
Testclient mit META-File (Fallback Lösung))

XII. Installer

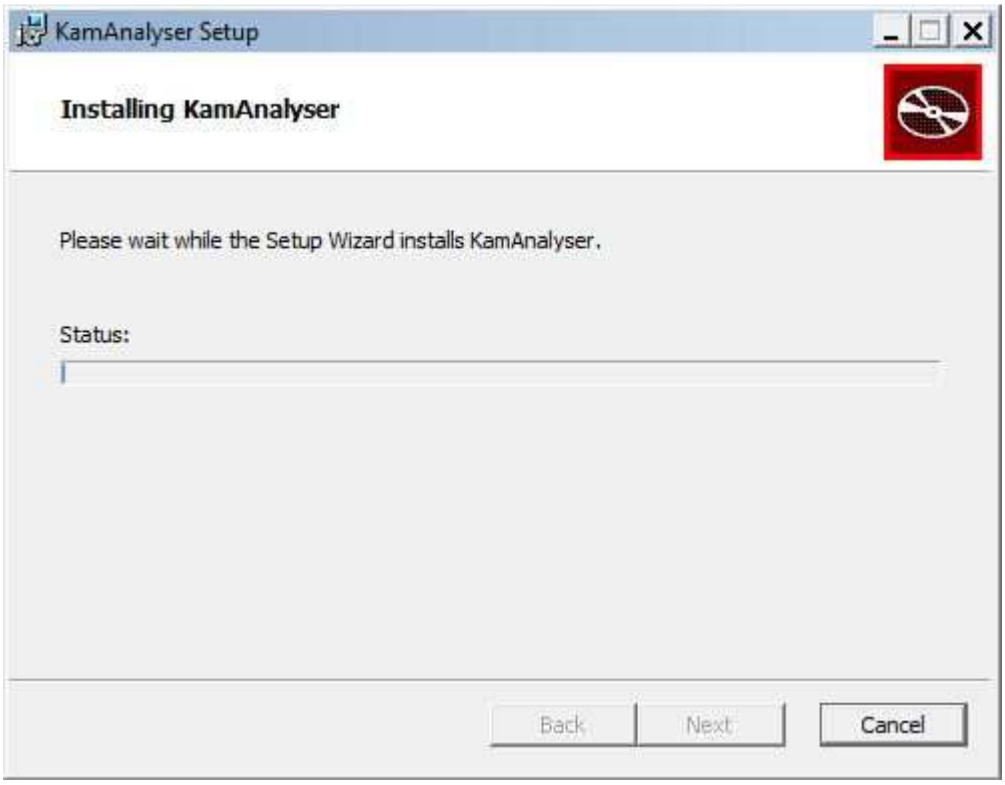


(Startbildschirm des Installers)

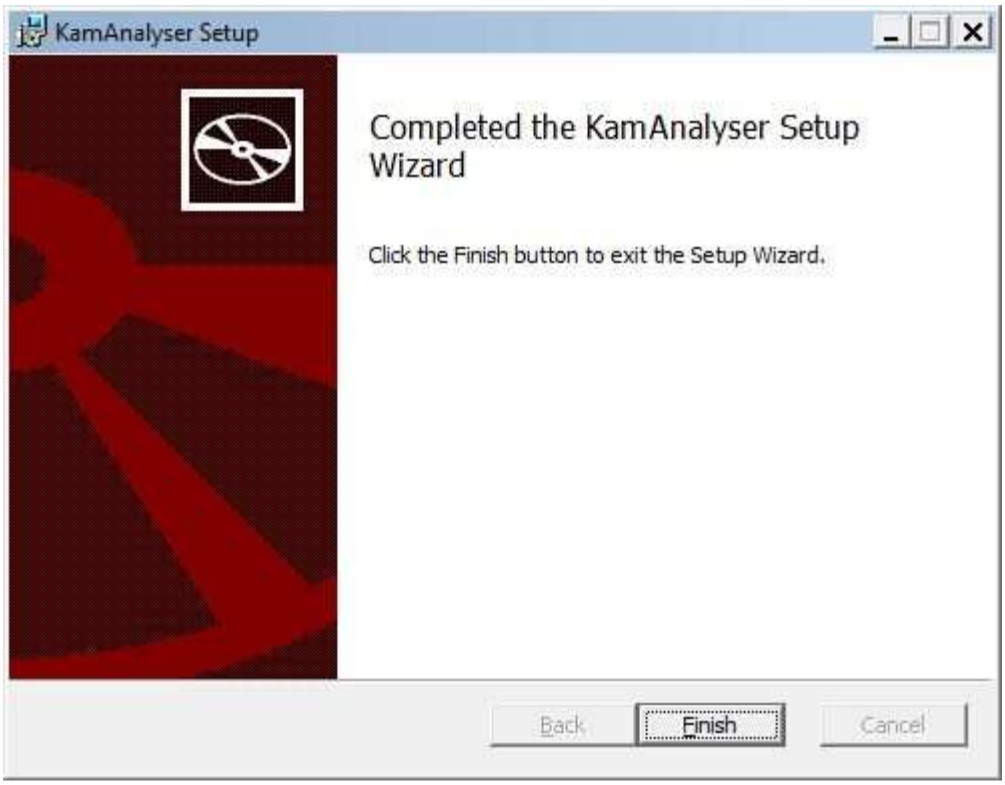
(Lizenzbestimmung)



(Setup-Auswahl)



(inkludierter Prozess-Status)



(*Fertigstellung der Installation)*

PFLICHTENHEFT

KAMANALYSER

PROJEKTTEAM: M. BULLMANN
J. GAUKEL
D. KILIAN
M. MIELERT
J. SCHELLER

AUTOR: J.SCHELLER

SERVER: WIN2008-STE2

1. Zielbestimmung

Im Rahmen des AWP-Unterrichts wurde Plus49 von Herrn Steinam beauftragt ein Programm zu erstellen, mit welchem man aus alten Zwischen- und Abschlussprüfungen Stegreifaufgaben und Schulaufgaben generiert werden können.

1.1 Muss-Kriterien (Logik)

Die Prüfungen liegen als Word-Dokumente vor. Diese müssen während des Einleseprozesses vom Programm in die vorgegebenen Teilgebiete (GA1, GA2, WiSo) aufgesplittet werden. Innerhalb dieser Bereiche sollen die einzelnen Handlungsschritte – zusammen mit sprechenden KeyWords - gespeichert werden.
Berechtigte User (Lehrer) sollen sich daraus Klassenarbeiten oder Übungsblätter zu verschiedenen Themengebieten erstellen können.

1.2 Muss-Kriterien (technisch)

Da es sich um ein Schulprojekt handelt, soll hier nicht speziell auf das eigentliche Ergebnis des Projekts geachtet werden, sondern auf die Verwendung verschiedener Produkte, Richtlinien und sog. kleiner Helferchen, die dem Programmierer bei Großprojekten Arbeit erleichtern, diese komfortabler oder sicherer machen.

1.3 Wunschkriterien

Wünschenswert wäre der Einsatz des Team Foundation Servers. Dieser ist zwar für ein einzelnes Projekt und das kleine Entwicklerteam oversized, allerdings vereint er mehrere Produkte (Bugtracker, Versionsverwaltung, Buildsysteme, …).

2. Produktumgebung

Das Programm KamAnalyser ist an eine Windows-Umgebung gebunden, da das Einlesen neuer Datensätze aus Kammerprüfungen über eine Microsoft-Word-API geregelt wird.

2.1 Software

Windows-Betriebssystem, Microsoft Office (Word), ab Version 2007, Microsoft SQL

2.2 Hardware

Die Anforderungen für einen SQL-Server, sowie Office 2007 sollte der PC/Server erfüllen

3. Produktbereiche

3.1 Login

Die KamAnalyser-Software kann ohne Loginfunktion benutzt werden. Somit kann jeder, der Zugriff auf das Programm erhält, sich Aufgaben erstellen lassen. Ein Login ist nur nötig, um neue (Prüfungs-) Aufgaben einzulesen, Bearbeitungen zu bestätigen und Metatags zu aktivieren. (Super-User)

3.2 Datenbank

Die Datenbank basiert auf den Microsoft SQL-Server. Die Datenbank (inkl. Datensätze und Struktur) soll über die Solution erstellt werden können. Somit können Neuerungen und Anpassungen jederzeit problemlos aus der Solution heraus gewährleistet werden.

3.3 Word API

Die Prüfungen müssen als Worddokumente vorliegen, bereits in einzelne Handlungsschritte zerlegt. Mit Hilfe der Word-API – basierend auf COM - werden die Bytes der Dokumente eingelesen und an die Datenbank übergeben. Somit kann nicht nur der Text, sondern auch die Struktur und Formatierung gespeichert und später aufgerufen werden.

3.4 Benutzerverwaltung

Administrator des Projekts bleibt Plus49. Super-User können über das Programm angelegt werden und werden entsprechend in der DB gespeichert. Eine Registrierung über die Oberfläche der Software ist nicht vorgesehen.

4. Produktdaten

Im System werden keine Daten über die Nutzung der einzelnen User gespeichert. Auch erstellte Aufgaben-Blätter werden nach dem Verlassen des Systems wieder gelöscht.
Nur die Eingelesenen Prüfungsteile werden gespeichert, sowie deren zugeordnete Metatags.

5. Benutzungsoberfläche

Im Folgenden werden die grobe Struktur und das äußere Erscheinungsbild unserer Tools beschrieben. Das Programm wird so programmiert, das ein nachträgliches Ändern der Oberfläche jederzeit möglich ist, ohne im Code eingreifen zu müssen.

5.1 Seitenaufbau
--- Wird direkt abgeklärt, wenn das Projekt soweit ist ---

5.2 Startseite
--- Wird direkt abgeklärt, wenn das Projekt soweit ist ---

5.3 Benutzermenü
--- Wird direkt abgeklärt, wenn das Projekt soweit ist ---

5.4 Bildschirmlayout

--- Wird direkt abgeklärt, wenn das Projekt soweit ist ---

5.5 Aufgaben anlegen

--- Wird direkt abgeklärt, wenn das Projekt soweit ist ---

6. Qualitätszielbestimmungen

Auf welche Qualitätsanforderungen (Zuverlässigkeit, Robustheit, Benutzungsfreundlichkeit, Effizienz, ...) wird besonderen
Wert gelegt?

	sehr wichtig	wichtig	weniger wichtig	nicht relevant
Design		X		
Performance		X		
Kompatibilität			X	
Stabilität	X			
BS-Unabhängigkeit				X

7. Entwicklungsumgebung

Plus49 hat sich verpflichtet nur Freeware (Open-Source-Software) oder von der Schule bereit gestellte Software zu verwenden.

7.1 Betriebssysteme

Vorwiegend Windows XP / 7 auf Schulrechnern / Privat-Laptops

Windows Server 2008 R2 auf Server

MacOS X Snow Leopard auf Privat-Laptop

7.2 Software (Server)

Microsoft SQL Server
Microsoft Team Foundation Server

7.3 Software (PC)

Microsoft Visual-Studio 2010
Microsoft Office 2007
GIMP

8. Plus49

Die Projektgruppe besteht aus fünf Auszubildenden/Schülern der Klasse 12 FI 2 der Klara-Oppenheimer-Schule. Diese wurden im Rahmen der Projektarbeit im AWP-Unterricht gegründet. Der Name entstand in Anlehnung an eine Stegreifaufgabe, die am Gründungstag abgehalten wurde.
Plus49 stellt seinem Kunden für jeden Bereich entsprechende Ansprechpartner zur Verfügung.

Teamleiter:	Jochen Scheller
Server:	Martin Mielert
Datenbank:	Jeremy Gaukel
Word-API:	Markus Bullmann
Architektur:	Dominik Kilian

9. Sonstiges

Alle verwendeten Bilder, Logos und Texte sind geistiges Eigentum von Plus49. Eine Verwendung ohne schriftliche Genehmigung wird ausdrücklich untersagt.
Der KamAnalyser und sein Quelltext dürfen nicht weitergegeben, kopiert oder anderweitig verwendet werden.

10. Anhang

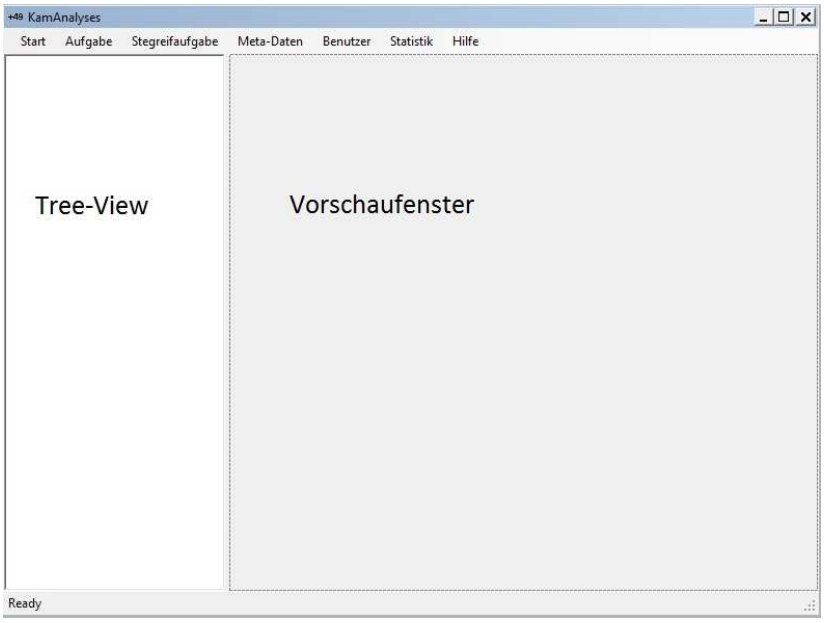


Abbildung 1: Main-Window

Abbildung 2: Aufgaben anlegen

XIV. Exportiertes Protokoll: Team-Meeting

Aufgabe #60: Diskussionsrunde (ALLE)
Titel: Diskussionsrunde (ALLE)
Zugewiesen zu: Martin Mielert
Zustand: Geschlossen
Grund: Abgeschlossen
Bereich: KamAnalyser
Iteration: KamAnalyser
Priorität: 2
Revisionen
07.02.2011 08:10:52, Bearbeitet von Martin Mielert
07.02.2011 08:06:16, Bearbeitet von Martin Mielert
07.02.2011 08:04:46, Bearbeitet von Dominik Kilian
07.02.2011 08:04:11, Bearbeitet von Martin Mielert
07.02.2011 08:02:43, Bearbeitet von Martin Mielert
07.02.2011 08:01:53, Bearbeitet von Martin Mielert
07.02.2011 07:59:50, Bearbeitet von Martin Mielert
07.02.2011 07:56:07, Bearbeitet von Martin Mielert
07.02.2011 07:55:54, Bearbeitet von Martin Mielert
07.02.2011 07:55:38, Bearbeitet von Martin Mielert
07.02.2011 07:55:25, Bearbeitet von Martin Mielert
07.02.2011 07:54:36, Bearbeitet von Martin Mielert
07.02.2011 07:46:03, Bearbeitet (Aktiv in Geschlossen) von Martin Mielert

Anwesend:
Alle

Start:
08:10 Uhr
Ende:

08:45 Uhr
Thema:
1. Setup
2. Dokumentation
3. Build
4. Word
5. MetaDaten (Email an Admin)

Diskussion:

1. Das Setup und das Programm müssen kontrollieren ob Word (2007 oder höher) installiert ist (Bullmann, Kilian). Setup Tests durch Kilian

2. Bullmann: Word Schnittstelle; Jeremy: Datenbank und Statistik (ZedGraph); Jochen & Martin: GUI Dokumentation; Dominik: Setup, Build;

3. Build Ergebnis mit letzten Build (Release). Setup Pfad?

4. Problem Word Datei bearbeiten. Lösung: Dauerhafter File Access Versuch bis Word Datei freigegeben wird (Word geschlossen wird).

5. Email fällt weg. UseCase Änderung Kilian. Hinweis wenn Admin einloggt und neue MetaDaten vorhanden sind

XV. Aufgabenübersicht TFS

Die Aufgabenübersicht wurde in einer Excel-Tabelle beigefügt. Aus Übersichtsgründen konnte diese nicht in den Anhang übernommen werden