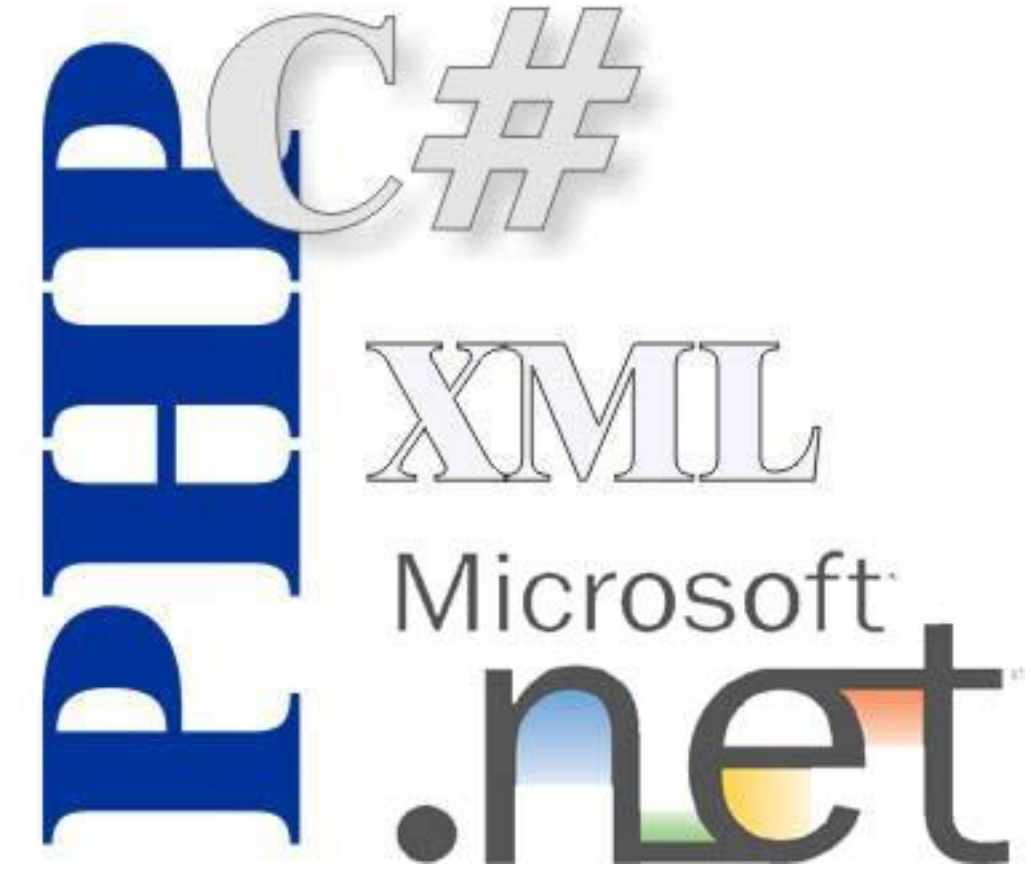
Abschlussbericht



Wirtschafts-Schwerpunkt-Praktikum

Informatik

SS 2004

Inhaltsverzeichnis

1. Abschlussbericht der .NET-Gruppe

1.1 Allgemeine Vorgehensweise

Vorbereitung/erste Schritte

Unser erster Schritt der Vorbereitung bestand darin Bücher über .Net zu besorgen, um uns einen allgemeinen Eindruck über sowohl die Sprache als auch die Entwicklungsumgebung zu beschaffen. Auch unsere Internet-Recherchen blieben nicht ergebnislos: Einige erklärende Dokumente über den Aufbau, Sinn und Zweck der DataSets ließen sich finden.

Die Installationsphase

Natürlich blieb uns bei unseren Experimenten mit der Original-DataSet-Implementierung die Installation von Visual Studio .Net nicht erspart. Gewisse Anfangsprobleme mit verschiedenen Webservern ließen sich ebenso wenig vermeiden. Das hatte seine Gründe:

Als erstes beschäftigten wir uns mit dem Apache Webserver - unser Versuch eine Zusammenarbeit mit .NET herzustellen schlug jedoch kläglichst fehl. Darauf hin fragten wir Hrn. Kaufmann telefonisch um Rat. Dieser empfahl uns den IIS, den Windows XP direkt mitlieferte. Nur leider war der IIS nicht unter der ursprünglichen Arbeitsplattform Windows XP Home Edition verfügbar. So blieb uns nichts anderes übrig als ihn auf einen anderen Rechner zu installieren, der unter der Professional Edition lief. Ab diesem Zeitpunkt – da wir uns nun eine lauffähige Arbeitsumgebung geschaffen hatten - konnten wir zum eigentlichen Teil der Aufgabenstellung übergehen.

1.2 Codegenerierung

Diese bestand nun im Wesentlichen in der Generierung eines DataSets unter .NET und der Analyse des vom Codegenerator ausgegebenen Quelltexts. Zu Testzwecken generierten wir ein selbst erdachtes Bespiel eines DataSets, um uns zunächst einmal mit der Arbeitsweise und dem Output des Codegenerators vertraut zu machen. Da keiner von uns Visual-Basic versiert war, entschieden wir uns bei der Codegenerierung nicht zuletzt aufgrund der Affinität mit Java für die Sprache C#. Verständlicherweise, da es uns auch an .NET Versiertheit mangelte, erwies sich auch die Einarbeitung in die Arbeitsweise der Umgebung hin und wieder als problematisch, z. B. bei der Organisation in Solutions und Objekte.

1.3 Codeanalyse

Nach unserem ersten Einstiegsversuchen generierten wir einen umfangreicheren DataSet. Dieser baute mehr oder weniger auf unserem Testversuch auf. Aus diesem DataSet heraus generierten wir erneut Code in C#.

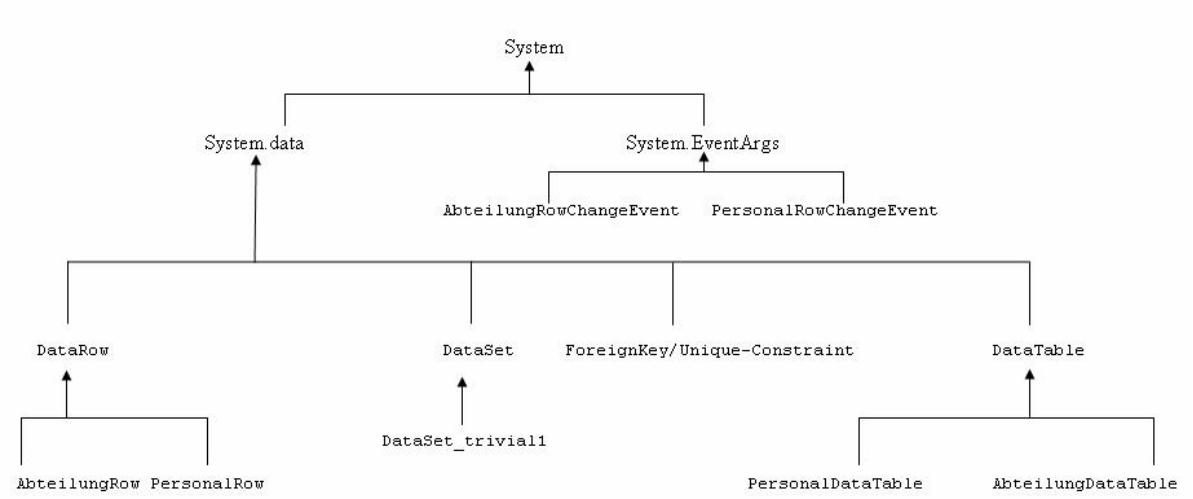
Um dem PHP-Teil unseres Teams einen guten und einfachen Start zu ermöglichen und uns selbst zunächst einen Überblick zu erschaffen, kommentierten wir den generierten Code mit Hilfe der MSDN, .NET-Büchern und einigen Internet-Recherchen komplett durch.

Wie wir es nicht anders erwartet hatten, blieb auch diese Phase nicht ohne Probleme.
Da war beispielsweise der Verlust einer Arbeitssession, da die Suns Probleme gemacht haben und der Administrator Hr. Franke in Urlaub war. Dieser Rückstand in unserer Zeitplanung konnte aber relativ zügig aufgeholt werden.
Abgesehen davon mangelte es dem PHP-Teil aufgrund des Umfangs an Überblick in unserem Code. Dieses Problem führte zu einem Gesamttreffen aller Gruppenmitglieder, wo wir die weitere Vorgehensweise beratschlagten. Die PHP-Gruppe wollte einen schrittweisen Aufbau und mehr Infos. Darüber hinaus waren viele Klassen im generierten Code nicht unbedingt für die ersten Schritte des PHP-

Teils relevant. In dieser Situation erhielten wir von Herrn Kaufmann die optimale Vorgehensweise.

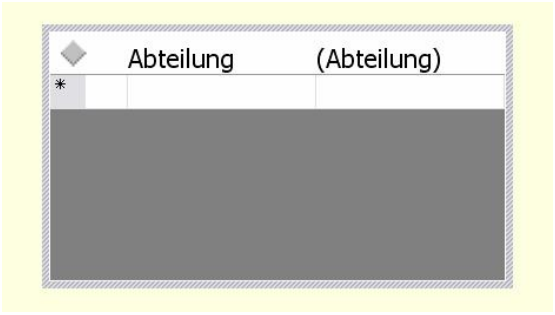
Schrittweiser Aufbau des DataSets

Aufgrund der vorhergegangenen Probleme folgte eine Überarbeitung der Dokumentation des generierten Codes – eine Art schrittweise Verfeinerung und die Vererbungshierarchie als Diagramm.

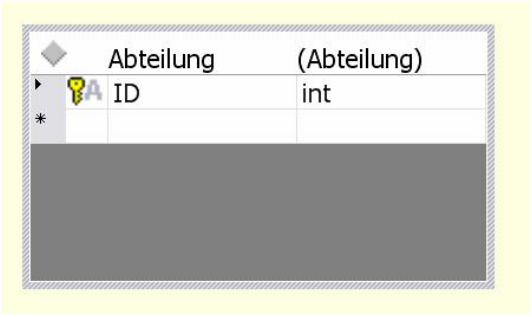


Weitere Experimente an den DataSets

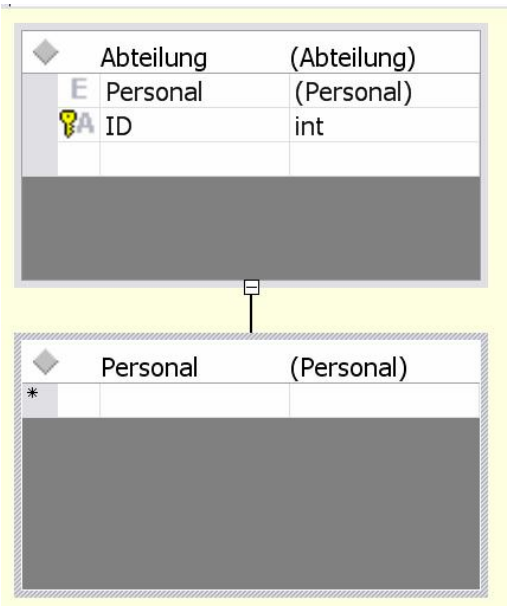
Unsere experimentellen Veränderungen am DataSet bestanden im Wesentlichen im stufenweisen Einfügen neuer Attribute in unsere Tabellen, im Anlegen neuer Tabellen samt deren Verknüpfung mit vorangegangenen sowie in Veränderung der Datentypen und der nachfolgenden Analyse welche Auswirkungen all dies auf den generierten Code hat



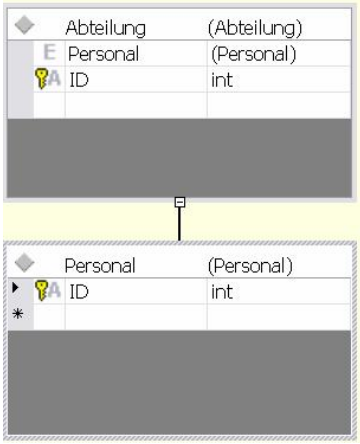
Stufe 1 – primitives Dataset



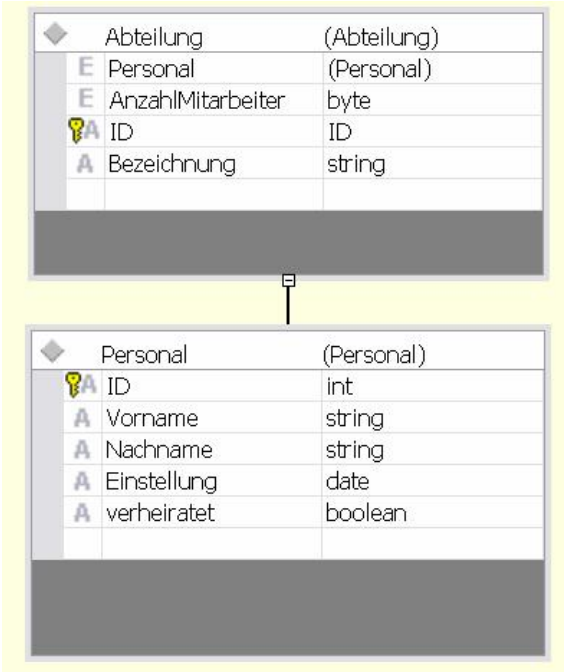
Stufe 2 – mit einem Element als Primärschlüssel(ID)



Stufe 3 – mit zusätzlich verknüpfter Tabelle



Stufe 4 – mit zusätzlichem Fremdschlüssel(ID_0)



Stufe 5 – mit zusätzlichen Datentypen

Den daraufhin generierten Code analysierten wir differenziert nach den einzelnen 5 Stufen, um der PHP-Gruppe die Arbeit zu erleichtern.

Tabellarische Übersicht der Codedokumentation für PHP-Gruppe

- à **für den Codegenerator**

Als weitere Vereinfachung für die PHP-Gruppe haben wir den vorher nach Stufen zerlegten Code in Tabellenform noch weiter aufgegliedert. Dabei haben wir alle Klassen, Methoden und Variablen jeweils separat aufgelistet. Returnwerte und Kommentare haben wir auch nochmal der Übersicht halber herausgefiltert. Durch diese umfangreiche Aufgliederung war jetzt die Arbeitsweise des Codegenerators wesentlich nachvollziehbarer und die PHP – Gruppe hatte den Grundstein für ihre Arbeit.

Einfügen	**Klassen**	**Methoden / Funktionen**	**Variablen / Array**	**Aktion**
Leere Tabelle Abteilung	public DataSet_trivial 1: DataSet		private AbteilungDataTable table Abteilung	
…	…	…	…	…
		public void AddAbteilungRow (AddAbteilungRow)		Zeile hinzufügen

2. Abschlussbericht der PHP-Gruppe

2.1 Was ist PHP?

PHP (rekursives Akronym für "PHP: Hypertext Preprocessor", ursprünglich "Personal Home Page Tools") ist eine Skriptsprache mit einer an C bzw. Perl angelehnten Syntax, die hauptsächlich zur Erstellung dynamischer Webseiten verwendet wird. PHP zeichnet sich besonders durch die leichte Erlernbarkeit, breite Datenbankunterstützung und Internet-Protokolleinbindungen, sowie die Verfügbarkeit zahlreicher, zusätzlicher Funktionsbibliotheken aus. Es existieren zum Beispiel Bibliotheken, um allein mit PHP GTK-Anwendungen zu entwickeln.
PHP ist eine serverseitig interpretierte Sprache. Das bedeutet, dass im Gegensatz zu HTML oder weitestgehend JavaScript der Quelltext nicht direkt an den Browser übermittelt, sondern erst vom Interpreter auf dem Webserver ausgeführt wird. Die Ausgabe des Skriptes wird dann an den Browser geschickt. Die Ausgabe ist in den meisten Fällen eine HTML-Seite, es ist aber auch möglich, mit PHP andere Datentypen wie z. B. Bilder zu generieren.
Die Vorteile der serverseitigen Ausführung sind, dass beim Client (Browser) keine speziellen Fähigkeiten erforderlich sind oder Inkompatibilitäten auftreten können, wie es z. B. bei Javascript und den verschiedenen Browsern der Fall ist. Außerdem bleibt der PHP-Quelltext der Seite auf dem Server, nur der generierte Code ist für den Besucher einsehbar. Gleiches gilt für andere Ressourcen wie z.B. Datenbanken, die daher auch keine direkte Verbindung zum Client benötigen.
Nachteilig ist, dass jede Aktion des Benutzers erst bei einem erneuten Aufruf der Seite erfasst werden kann.
Außerdem wird jede PHP-Seite vom Server interpretiert, wodurch die Auslastung des Servers steigt. Diese Vor- und Nachteile sind nicht PHP spezifisch, sondern treten bei grundsätzlich jeder Webapplikation auf.
PHP ist zeitweise etwas ungesteuert gewachsen, so funktioniert der Zugriff auf eine Datenbank mittels der MySQL-Funktionen anders als über ODBC; noch deutlicher wird dies beispielsweise bei Inkonsistenzen der Funktionen zur String-Bearbeitung.

Zwar besitzt PHP bereits seit Version 3 grundlegend die Funktionalität, um objektorientiertes Programmieren zu unterstützen (diese wurden in Version 4 deutlich verbessert), bisher ist jedoch die gesamte Standardbibliothek prozedural angelegt. Auch bei objektorientierten Sprachen übliche Features wie Kapselung der Daten (z.B. private Variablen), Destruktoren (ersatzweise lässt sich aber in den meisten Fällen die Funktion register_shutdown_function() (http://www.php.net/register-shutdown-function) verwenden) oder Fehlerbehandlung per Exceptions suchte man in PHP 4 noch vergeblich.

2.2 Allgemeine Vorgehensweise

Einarbeiten in PHP

Nachdem die Projektgruppe in zwei Untergruppen geteilt wurde, hat sich die PHP-Gruppe zunächst 3 Wochen lang in PHP eingearbeitet. Hierzu gehörte neben dem allgemeinen Erlernen der Programmiersprache PHP auch die Installation einer geeigneten Entwicklungsumgebung.

Eine erste Unklarheit war hierbei für welche Version von PHP man sich entscheiden sollte.

Die PHP-Gruppe hat sich für PHP 4 entschieden, da zwar PHP 5 einige zusätzliche Funktionen bietet, zum Zeitpunkt des Projektbeginns aber nur in Betaversionen verfügbar war und besonders die Dokumentation der neu eingeführten Funktionalität noch entwickelt wurde. Mit der durch dieses Projekt gesammelten PHP-Erfahrung würde diese Entscheidung nun anders ausfallen: PHP 5 hat hauptsächlich Neuerungen im Bereich der Objektorientierung, und gerade diese fehlten in der Programmierpraxis mit PHP 4 (besonders die Schlüsselwörter public, private und protected). Auch konnte beoachtet werden, dass andere neu gegründete PHP-Projekte zur gleichen Zeit schon sehr oft auf PHP 5 basierten. Im OpenSource-Bereich scheint Zurückhaltung bei Beta-Versionen damit eher unangebracht.

Ein weiterer Punkt war, dass für die Ausführung von PHP ein Webserver notwendig ist und die Installation eines solchen nicht unbedingt einfach ist.

.Net DataSet

Nach der Einarbeitung in PHP hat sich die PHP-Gruppe zunächst auf der Seite http://www.msdn.microsoft.com/ allgemein über das .Net DataSet erkundigt und hierbei alle Klassen herausgesucht, die für das DataSet wichtig sind. Diese wurden dann auf die drei Mitglieder der Gruppe zur Implementierung aufgeteilt.
Nun hat sich jeder genauer über die jeweiligen Klassen in .Net erkundigt und diese in PHP nachimplementiert.
Eine Schwierigkeit war hierbei, dass versucht wurde die Funktionalität der .Net-Klassen so weit wie möglich zu übernehmen aber auch abzugrenzen was für den gegebenen Fall nicht unbedingt notwendig ist oder in PHP überhaupt nicht zu realisieren ist.

Dokumentation

Parallel zu der Implementierung wurde die Dokumentation geführt. Hierfür wurde mit PHP-Doc gearbeitet (http://www.phpdoc.org), verwandt wurde Version phpDocumentor 1.3.0RC3. Das OpenSource-Programm phpDocumentor ist ein Dokumentationsgenerator, der als Eingangsdaten speziell formatierte Kommentare in den PHP-Quelltextdateien selbst verwendet. Installiert wird unter Linux denkbar einfach, indem man das Paket entpackt und den Pfad zur ausführbaren Datei »phpdoc« in die $PATH-Umgebungsvariable aufnimmt. phpDoc unter Windows zu installieren ist im Rahmen dieses Projektes nicht gelungen.
Die Einarbeitungszeit in phpDoc fällt recht kurz aus (etwa 1,5h) wenn das von phpDoc erwartete spezielle Textformat der Kommentare schon von JavaDoc oder kompatiblen Dokumentationslösungen bekannt ist. Gegen Ende des Projektes, als die Dokumentation überarbeitet wurde und speziellere Tags und Funktionen von phpDocumentor benötigt wurden, zeigte sich dass phpDoc 1.3.0 noch etwas fehlerbehaftet ist und auch die Dokumentation der spezialisierten Funktionen noch ergänzt und weiter strukturiert werden muss.
Insgesamt ist phpDoc ein wirklich hilfreiches Werkzeug gewesen; damit umgehen zu lernen war in diesem Projekt eine lohnende Investition. phpDoc sorgte für eine einheitlich formatierte Klassendokumentation in Form eines zusammenhängenden Dokuments mit stets aktuellen Hypertext-Links. Die Kollaboration an dieser Dokumentation war durch phpDoc auch völlig unproblematisch, während manuelle Wartung solcher Dokumentation mit großem Aufwand verbunden wäre.

Vorgehensweise:

Phase 1: Einlesen in PHP

Während der ersten Phase des Projekts, die ca. 4 Wochen dauerte, war es die Aufgabe der PHP-Gruppe, sich mit der Syntax und den Besonderheiten von PHP auseinanderzusetzen. Ein sehr gute Dokumentation wurde unter http://www.php.net gefunden. Auf dieser Seite gab es sowohl eine ausführliche Beschreibung der Sprache, wie auch umfangreiche Beispiele, die einem die Einarbeitung in PHP sehr erleichterten.

Als problematisch stellte sich hierbei heraus, dass man nicht genau wusste, auf welches Gebiet von PHP man die Schwerpunkte legen sollte, da man noch nicht genau wusste, was man eigentlich programmieren sollte. Am Anfang des Projekts wurde zwar kurz angerissen, was ein Dataset darstellt, jedoch musste die PHP-Gruppe auf die Ergebnisse der Dataset-Gruppe warten.

Ein wichtiges Werkzeug zur Dokumentation des Quelltextes wurde mit phpdoc gefunden, was unter http://www.phpdoc.org kostenlos heruntergeladen werden kann. Mit Hilfe dieses Tools war es fortan möglich, durch spezielle Formatierung der Kommentare im PHP-Quelltext, HTML-Dateien erzeugen zu lassen, die eine Übersicht geben, was in einer PHP-Datei an Methoden, Attributen und Klassen enthalten ist. Ferner kann man Angaben über die enthaltenen Dateien eines Packages, Autoren und Daten über Erstellung aus diesen Dokumenten entnehmen.

Phase 2: Definieren des Auftrags

Durch die Dataset-Gruppe wurde die Arbeitsweise und ein Klassenmodell eines .NET-Datasets vorgestellt. Anhand des Klassenmodells sollte nun die Funktionalität des .NET-Datasets in PHP implementiert werden.

Innerhalb der PHP-Gruppe wurde jedem Mitglied eine Klasse zugeteilt, deren Implementierung die Aufgabe war.

Phase 3: Implemetierung der Klassen

Zunächst wurden organisatorische Fragen geklärt. Es wurde vereinbart, sich Montags ab 13.00 Uhr zu treffen, um die Klassen zu implementieren. Die Treffen sollten dazu dienen, auftretende Schwierigkeiten gemeinsam zu lösen und die einzelnen Klassen besser aufeinander abstimmen zu können.

Phase 4: Erstellen der Klassen

Nachdem das Klassendiagramm vorgegeben war, musste man den Aufbau und die Funktionsweise einer Klasse verstehen. Dazu benutzten die Team-Mitglieder die Dokumentation von Microsoft, MSDN (http://msdn.microsoft.com). In dieser sehr detaillierten Beschreibung der Klassen und dessen Funktionen, konnte man sich ausführlich über die Arbeitsweise der Komponenten informieren.

Schritt 1: Extrahieren der relevanten Funktionen und Attribute

Zu jeder Klasse ein .NET-Datasets gehören eine Vielzahl von Funktionen und Attributen, die aus zeitlichen Gründen nicht alle zu implementieren sind. Aus diesem Grund beschlossen die Teammitglieder aus der Ihnen zugeteilten Klasse die für die Implementierung notwendigen Komponenten zunächst in der Klasse festzuhalten.
Zur Verdeutlichung ein kleines Beispiel für die Durchführung des ersten Schrittes:

```
function RelationName(){}
```

Nach diesem Schema wurden die Klassen zunächst gefüllt.

Schritt 2: Kommentieren der Klassen

Die Codeimplementierung wurde zunächst völlig außen vor gelassen. Vorrangige Aufgabe war es, die Klassen vor der Implementierung zunächst vollständig zu dokumentieren.
Um aus der Dokumentation der einzelnen Klassen eine Gesamtdokumentation zu bekommen, hielten wir und an die Regeln von phpdoc, damit später ein Gesamtdokumentation im HTML-Format erstellt werden konnte.
Auf diese Weise konnte man schon vor dem erzeugen des Codes eine annähernd vollständige Dokumentation des PHP-Projekts bekommen. Man hatte eine Übersicht über die Klassen und deren Komponenten.
Phpdoc bietet die Möglichkeit zur Erstellung einer ToDo-Liste. In dieser Liste, die ihren Inhalt auch aus den php-Dateien bezieht, konnte man Aufgaben, die man selber, beziehungsweise die anderen Teammitglieder noch zu erledigen hatten, festhalten.

Schritt 3: Generierung des Codes

Nachdem das Grundgerüst der Klassen gebaut worden war, kam anschließend die eigentliche Hauptaufgabe, die Generierung des Codes.
Um den Code zu schreiben, muss man die Arbeitsweise der zu implementierenden Komponente verstehen. Hier bezog man sich auf die MSDN-Dokumentation der Klassen. Zu annähernd jeder Funktion einer Klasse war ein Codebeispiel in diversen Programmiersprachen gegeben. Anhand dieser Beispiele wurde die Arbeitsweise der Komponenten interpretiert.
Ein Beispiel soll diese Vorgehensweise verdeutlichen. Im folgenden wird in einem DataRow-Objekt ein Array mit Werten belegt, im übertragenem Sinne eine Tupel mit Werten gefüllt:

```
private void CreateRowsWithItemArray(){
    // Make a DataTable using the function below.
    DataTable dt = MakeTableWithAutoIncrement();
    DataRow dr;
    // Declare the array variable.
    object [] myArray = new object[2];
    // Create 10 new rows and add to DataRowCollection.
    for(int i = 0; i <10; i++){
      myArray[0]=null;
      myArray[1]= "item " + i;
      dr = dt.NewRow();
      dr.ItemArray = myArray;
      dt.Rows.Add(dr);
    }
    PrintTable(dt);
 }
```

In dem rot hinterlegten Bereich kann man erkennen, dass in einer Schleife DataRow-Objekte erzeugt werden, die anschließend mit Werten gefüllt werden. Die Aufgabe soll sein, eine Funktion ItemArray zu schreiben, bzw. den Code zu ItemArray zu schreiben. Die Funktionalität des Belegens eines Arrays mit Werten soll in PHP implementiert werden.
Auf diese Weise konnte mit einem Grossteil der Komponenten verfahren werden.

3.4.4 Schritt 4: Testen der Klassen

Nach der Implementierung wurden dann alle Klassen auf ihre korrekte Ausführung hin überprüft, wobei das Programm PHP-Unit (http://phpunit.sourceforge.net) zur Hilfe genommen wurde. PHP-Unit ist eine gerade einmal 20kB kleine Klassen-Kollaboration nach dem Vorbild von JUnit. Eigentlich gedacht zum Unit-Testing im Rahmen des Extreme Programming, war es im Rahmen dieses Projektes einfach hilfreich für strukturiertes Testen und automatisierte Regressionstests.
Installiert wird PHP-Unit, indem man die Datei phpunit.php und, wenn gewünscht, die ebenfalls enthaltene Stylesheet-Datei zur HTML-Ausgabe in ein Verzeichnis innerhalb des PHP-Include-Pfades kopiert. Im einfachsten Fall also in das Projektverzeichnis selbst.
Wie verwendet man nun PHP-Unit? Das Begriffssystem von PHP-Unit ist zunächst etwas verwirrend:

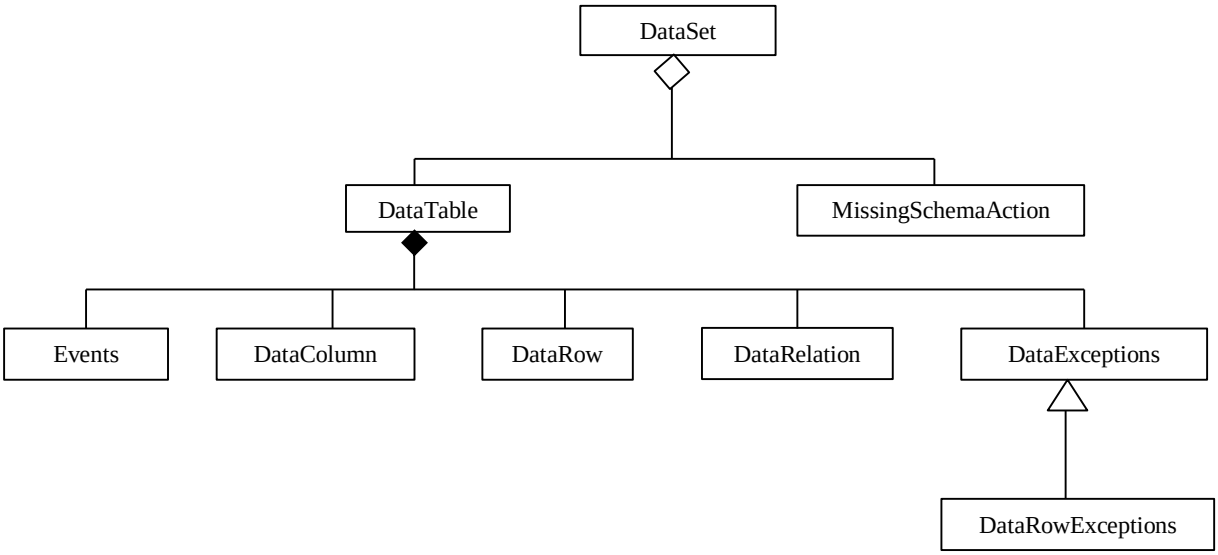
- Eine Testsuite ist eine Instanz der Klasse TestSuite, die einen oder mehrere Tests enthält.
- Ein Test ist eine Testsuite, die wiederum mehrere Tests oder nur einen Test*case* enthält. Üblicherweise gibt es eine Testsuite für je einen Testcase und eine weitere, die alle anderen Testsuites zusammenfasst. So auch in diesem Projekt.
- Ein Testcase ist eine Klasse, die von TestCase erbt. Im Rahmen dieses Projektes wurde ein Testcase für jede zu testende Klasse angelegt, hier für drei der Hauptklassen: DataRow, DataSet und DataTable. Ein Testcase kann mehrere Testmethoden enthalten.
- Eine Testmethode ist eine Methode, deren Name mit »test« beginnt und die zu einer von TestCase abgeleiteten Klasse gehört. Üblicherweise wurde in diesem Projekt eine Testmethode für jede zu testende Methode angelegt. Eine Testmethode kann mehrere Assertions enthalten.
- Eine Assertion ist eine Zusicherung, deren Fehlschlagen einen Programmfehler identifiziert und in der zusammenfassenden Ausgabe von PHP-Unit als Fehlschlagen einer Testmethode zusammen mit einer genaueren Fehlermeldung vermerkt wird.

- Ein Testrunner ist eine Instanz der Klasse TestRunner. Durch einen Befehl lassen sich dann alle in einer TestSuite enthaltenen Tests ausführen, etwa so: $testRunner->run($suitePhpDataSet);

Nach etwas Einarbeitung erwies sich PHP-Unit jedoch als flexibles und skalierbares Testing-Framework, gut geeignet um den eigenen Testcode kompakt zu organisieren (Stichwort »Fixture«) und vom Produktionscode getrennt zu halten. Die automatisierten Regressionstests erwiesen sich mehrmals als besonders hilfreich: Fehlfunktionen bisher fehlerfreier Methoden konnten schneller gefunden werden, weil der Auslöser, eine Änderung in einer anderen Methode, noch bekannt war.
Im TestCode des Projektes verwenden 21 Methoden das PHP-Unit-Framework; sie führen zu folgender Ausgabe des Testrunners (Format »Testcase - Testmethode Testergebnis«):

datarowtest - test_itemarray ok
datarowtest - test_rowstate ok
datarowtest - test_item ok
datarowtest - test_table ok
datarowtest - test_equals ok
datatabletest - test_tablename ok
datatabletest - test_loaddatarow ok
datatabletest - test_importrow ok
datatabletest - test_newrow ok
datatabletest - test_clear ok
datatabletest - test_reset ok
datatabletest - test_relations ok
datatabletest - test_columns ok
datatabletest - test_primarykey ok
datasettest - test_datasetname ok
datasettest - test_clear ok
datasettest - test_reset ok
datasettest - test_tables ok
datasettest - test_containsbasicschemaof ok
datasettest - test_merge_datatable ok
datasettest - test_merge_datarowcoll ok

2.3 Klassendiagramm



2.4 Klassenbeschreibung

Die Klasse DataSet bildet den Speicher für die Daten, die nach dem relationalen Modell im Hauptspeicher gehalten werden. Die Tabellen, die zu einem DataSet gehören, werden hier registriert.

DataTables stellen eine Sammlung von DataRow-Objekten dar, die einer bestimmten Tabelle zugeordnet werden.

2.4.1 DataRelation, DataColumn

DataRelation
Die Klasse DataRelation repräsentiert eine Eltern/Kind-Beziehung zwischen zwei DataTables.
Um diese Beziehung herzustellen werden wird eine oder mehrere DataColumns der DataTables dem Konstruktor der DataRelation übergeben.
Bei der Implementierung der Klasse wurde versucht so nah wie möglich an der .Net Klasse zu arbeiten, so wurde zum Beispiel die Überladung des Konstruktors übernommen, so dass es möglich ist entweder nur eine DataColumn oder auch ein Array von DataColumns zu übergeben. Nicht realisiert wurden jedoch Constraints.

Da PHP eine untypisierte Sprache ist wurde auch die Methoden die Typen betreffen weggelassen.

DataColumn

Die Klasse DataColumn entspricht einer Spalte einer DataTable und repräsentiert somit das Schema einer Solchen.
Auch hier wurde versucht die Überladungen des Konstruktors zu übernehmen jedoch ohne Typen zu berücksichtigen.

2.4.2 DataRow, DataRowExceptions

Die Klasse DataRow stellt eine Zeile einer Tabelle dar. Innerhalb eines DataRow-Objekts werden die Daten in einem Array gespeichert. DataRowExceptions behandelt die Ereignisse, die von der Klasse DataRow geworfen werden.

2.4.3 DataTable, DataSet, MissingSchemaAction, DataExceptions, Events

Allgemeine Designentscheidungen

Der Namen von Attributen beginnt mit einem Großbuchstaben. Der Name von get/set-Methoden zu diesen Attributen entspricht dem Namen des Attributs, beginnt jedoch mit einem Kleinbuchstaben.
Eine wesentliche Vereinfachung gegenüber Microsofts .NET-DataSet ist der Verzicht auf die dortige, elaborierte Versionsverwaltung der Daten in einem DataSet.

DataTable

DataTable ist zwar ebenso wie DataSet eine recht zentrale Klasse der Kollaboration, dient jedoch fast ausschließlich der Zusammenfassung anderer Objekte (DataRelations, DataColumns, DataRows und Constraints). Sie implementiert fast keine erwähnenswerte eigene Funktionalität; die weitaus komplexeste Methode ist DataTable::LoadDataRow mit gerade 39 Zeilen.
Die hier beobachtete geringe Elementkomplexität findet sich mehr oder weniger ausgeprägt eigentlich in allen Teilen der DataSet-Kollaboration. »Die Arbeit« wird

nicht von wenigen komplexen Algorithmen, sondern von der Kollaboration vieler einfacher Objekte getan (fast könnte man von Synergie sprechen ...).
Solch ein Design nutzt den Vorteil objektorientierter Programmierung aus. In diesem Fall ist es natürlich nicht unsere Leistung, sondern Microsofts Vorbild. Es im Detail zu studieren und daran zu lernen ist der große Vorteil, den eine Reimplementierungsaufgabe bietet.

DataSet

Die Hauptklasse der DataSet-Kollaboration. In der Designphase wurden gegenüber Microsofts .NET-DataSet folgende Vereinfachungen vorgenommen:

- Der vollständige Verzicht auf verschiedene Versionen der Daten in einem DataSet bewirkt den Verzicht auf DataSet::AcceptChanges, DataSet::GetChanges und DataSet::HasChanges.
- Auf die Standardmethoden Copy, Clone, Dispose, Equals und GetType und ToString wurde verzichtet. Zum Teil ist ihre Implementierung auch unnötig, weil interne Standard-PHP-Funktionen dieselbe Funktionalität bieten.
- Auf die XML-Schnittstelle zu DataSets wurde bisher vollständig verzichtet: es gibt keine der Methoden DataSet::GetXml, DataSet::GetXmlSchema, DataSet::InferXmlSchema, DataSet::ReadXml, DataSet::ReadXmlSchema, DataSet::WriteXml, DataSet::WriteXmlSchema.

Die Implementierung der Klasse DataSet gestaltete sich nur für die Methode DataSet::Merge etwas problematisch. Sie ist die komplexeste Methode der Klasse und erforderte die ebenfalls komplexe Hilfsmethode DataSet::containsBasicSchemaOf. Zur Komplexitätsminderung erwies sich für diese beiden Methoden Rekursion als sehr hilfreich: vor dem rekursiven Aufruf wird der aktuelle Fall lediglich auf einen leicht einfacheren Fall zurückgeführt.

Die Hilfsklasse MissingSchemaAction

MissingSchemaAction ist im .NET-DataSet vom Typ Enumeration. Dieser Typ definiert also Namen für Werte, die bestimmte Verhaltensweisen bei notwendigen Schemaänderungen in DataSets repräsentieren. Nun sind die Namen und ihre Werte in der bisherigen Implementierung gleich: MissingSchemaAction::Add z.B. führt zum String-Wert "Add", die beiden Codefragmente sind also austauschbar. Es wurde trotzdem entschieden, MissingSchemaAction einzuführen, um Namen und ihre Werte

zu entkoppeln, d.h. Schnittstelle und Implementierung zu trennen. Sollte später z.B. der String-Wert "Add" durch den numerischen Wert 1 ersetzt werden, ändert sich aus Benutzersicht nichts.
PHP besitzt keinen eingebauten Typ »Enumeration«. Hier wurde eine Klasse mit statischen Methoden verwendet, um Enumerations zu simulieren. Natürlich sind auch andere Möglichkeiten denkbar:

1. Mehrere globale Konstanten. Wurde vermieden, um dem Benutzer einen unbenutzten globalen Namensraum zu überlassen und Namenskollisionen so völlig auszuschließen.
2. Klasseneigene Konstanten. In PHP können zwar Konstanten innerhalb einer Klasse definiert werden, sie befinden sich dann aber trotzdem im globalen Namensraum und es gilt dasselbe Argument wie für globale Konstanten selbst.
3. Ein assoziatives globales Array. Diese eigentlich naheliegendste Idee entstand erst beim Schreiben dieser Dokumentation. Das Argument, einen unbenutzten globalen Namensraum zu fordern, gilt hier nur eingeschränkt, wird doch nur ein globaler Name definiert. Der Vorteil der hier gewählten Variante mit statischen Methoden ist jedoch, dass sie syntaktisch der Art entspricht, wie im .NET DataSet auf Werte einer Enumeration zugegriffen wird, in C++ z.B. mit dem Bereichsauflösungsoperator: MissingSchemaAction::Add.

Die Exception- und Event-Klassen

Zu Beginn der Implementierung wurde entschieden, der Einfachheit halber auf das Event- und Exception-Konzept zu verzichten, das im Microsoft .NET-DataSet enthalten war. Im weiteren Verlauf zeigte sich, dass diese Entscheidung nicht in allen Fällen durchzusetzen ist: manche Methoden innerhalb der DataSet-Kollaboration müssen Exceptions oder Events auswerten, die von anderen Methoden erzeugt werden, um korrekt funktionieren zu können.
PHP bietet ab Version 5 ein Exception-Konzept. Statt ein ausgereiftes Exception-Konzept für die hier in PHP4 gemachte Implementierung nachzubauen, bot es sich an, eine einfache Alternative zu wählen, die gleichzeitig einen leichten späteren Umstieg auf das Exception-Konzept von PHP 5 ermöglicht.
In diesem Sinn ist die Klasse DataException hauptsächlich eine Vorbereitung, um eine Migration auf PHP5 mit der dort PHP-internen Klasse "Exception" zu vereinfachen.

Das gewählte Konzept der Exception- und Event-Behandlung in PHP4 ist detaillierter in der phpDoc-Dokumentation zur Klasse DataException beschrieben.

2.5 Codegenerator

Nachdem alle Klassen implementiert und getestet wurden hat sich die Gruppe über den Codegenerator informiert um seine Arbeitsweise zu verstehen und diskutiert wie dies in PHP
zu realisieren wäre. Das gewählte Design und seine Alternativen werden in einem eigenen Kapitel ausführlich diskutiert.

Design des Codegenerators von phpDataSet

Zusammenfassung. Dieser Abschnitt beschreibt und begründet das für den Codegenerator von phpDataSet gewählte Design. Es ist aus zwei Gründen recht umfangreich. Zum einen wurde bisher nur ein Teil des Codegenerators implementiert, nämlich DataSet::makeTyped. Wird diese Arbeit durch andere Entwickler fortgeführt, sollten sie einen möglichst einfachen Einstieg vorfinden. Zum anderen ist das gewählte Design recht unkonventionell, erwies sich aber in der bisherigen, teilweisen Implementierung als vorteilhafter und recht kompakt formulierbarer Weg. Ein unkonventionelles Design muss sich in besonderem Maße rechtfertigen und erklären.
Implementierungsstand. Die Implementierung des Codegenerators für phpDataSet ist noch unvollständig. Ziel war es zuerst, nachzuweisen, dass das hier vorgestellte unkonventionelle Design erfolgreich umgesetzt werden kann. Das leistet der bisher implementierte Teil. Er trägt auch zur weiteren Dokumentation dieses Designs durch ein funktionierendes, prototypisches Beispiel bei. Was zur vollständigen Implementierung fehlt, ist in phpDoc-DocBlocs in der Klasse DataSet dokumentiert. Insbesondere gehört dazu die Methode DataSet::ReadXmlSchema.

Aufgabe des Codegenerators für phpDataSet

Aufgabe des Codegenerators in Microsoft .NET

Der Codegenerator ist ein Programm oder Programmteil, das in der Lage ist, die Beschreibung relationaler Datenstrukturen in Form von XML-Schemata zu lesen und daraus Code in einer der .NET-Sprachen zu generieren, der diese relationale Datenstruktur in einem typisierten DataSet abbildet.

Analogie

Der Codegenerator soll möglichst parallel zu seinem Pendant im Microsoft Visual Studio .NET arbeiten, kann aber Vereinfachungen enthalten. Dazu kann z.B. Übergang von typisierten zu untypisierten (oder zumindest nicht typsicheren) DataSets gehören. Der Codegenerator verwendet natürlich phpDataSet (nicht das .NET-Original) um diese relationale Datenstruktur als typisiertes DataSet im Hauptspeicher zu erstellen bzw. die Typen dafür zu generieren.
Der Codegenerator leistet die Verarbeitung von XSD-Dateien entsprechend der Methode DataSet::ReadXmlSchema des .NET DataSets und folgende zusätzlichen Dinge:

4. Zugriff auf DataTables, DataColumns und DataRelations als benamte Attribute statt über die Collections einer Klasse.
5. Typsicherheit, d.i. Möglichkeit zur Typprüfung zur Übersetzungszeit in streng typisierten Sprachen, nicht aber in schwach typisierten Sprachen wie PHP.
6. Klassennamen entsprechend der Aufgabe eines DataSets, einer DataTable usw., d.i. entsprechend dem (semantischen) Typ. Das gibt dem Benutzer die bequeme Möglichkeit, durch ein einziges new ...DataSet(...) das DataSet des gewünschten Typs inkl. seines gesamten Schemas zu erstellen.

Diese Leistungsmerkmale soll auch der Codegenerator für phpDataSet besitzen.

Funktionsnachweis

Die Funktion des Codegenerators wird nachgewiesen, indem er eine Beispiel-XSD-Datei korrekt in PHP-Code umsetzt. Die Funktion von phpDataSet selbst wird durch Tests nachgewiesen, etwa mit Hilfe des Test-Frameworks PhpUnit.

Typenprüfung zur Übersetzungszeit unmöglich

Ein Vorteil von typisierten DataSets in streng typisierten Sprachen ist: Typprüfung ist zur Übersetzungszeit möglich, Typfehler können schon dann aufgedeckt werden. Dieser Vorteil ist in schwach typisierten Sprachen wie PHP nicht gegeben: Typprüfung (egal ob manuell durch die PHP-interne Funktion is_a() oder automatisch ab PHP5 durch sog. type hints) geschieht hier immer zur Laufzeit. Typsicherheit ist also kein Grund, typisierte DataSets in PHP einzuführen - der Codegenerator für phpDataSet will und kann keinen Code generieren, der zur Übersetzungszeit auf Typfehler geprüft wird. Denn es gibt keine Übersetzungszeit in interpretierten Sprachen wie PHP.

Typprüfung zur Laufzeit gegeben

Typprüfung zur Laufzeit geschieht auch schon in der bisherigen Implementierung von phpDataSet: anhand des Namens von DataSets, DataTables und DataColumns, der sozusagen der Name des semantischen Typs ist. Diese Art Typprüfung zur Laufzeit kann und soll auch im DataSet, wie es der Codegenerator erzeugt, weitergeführt werden, z.B. bei den neu hinzukommenden Attributen und Methoden, die je den Zugriff auf eine Tabelle bzw. Spalte einer Tabelle erlauben. Typprüfung zur Laufzeit wird durch neue, abgeleitete Klassen, die über die XSD-Datei definiert werden, somit nur bequemer (über is_a() statt über den Namen des Objektes), nicht aber ermöglicht.

Design des Codegenerators für phpDataSet

Die Algorithmen des Codegenerators für phpDataSet und Microsoft .NET DataSets müssen nicht identisch sein, sondern nur analoges Verhalten zeigen. Unabhängig davon, wie der Algorithmus in Microsoft Visual Studio .NET intern arbeitet, ist es also möglich, dass sein Pendant für phpDataSet selbst die Klassen von phpDataSet benutzt. Dazu bietet es sich an, die Methode DataSet::ReadXmlSchema zu implementieren und zu verwenden - denn wie oben gezeigt, besteht die Basisfunktionalität des Codegenerators in der Verarbeitung von XSD-Dateien. Unter Verwendung von DataSet::ReadXmlSchema entsteht kein Codegenerator im eigentlichen Sinn mehr: Objekte erzeugen in ihrem eigenen Konstruktor ihr eigenes Schema, also zur Laufzeit, statt dass ein Codegenerator die Verarbeitung von XSD-Dateien zu einem Schema schon vorab durchgeführt hat. Auch wenn so kein

Codegenerator im eigentlichen Sinn mehr entsteht, leistet das Programm doch dasselbe: der Benutzer kann mit einem einzigen Funktionsaufruf ein typisiertes DataSet erstellen.
Das Konzept, erst zur Laufzeit, erst bei der Konstruktion eines Objektes, die XSD-Datei in ein typisiertes DataSet umzusetzen, lässt sich aufgrund einiger spezieller Spracheigenschaften von PHP als einer interpretierten Sprache nämlich durchhalten.
Es ergeben sich die folgenden wesentlichen Verarbeitungsschritte:

- Der Benutzer zeigt an, dass er ein typisiertes DataSet auf Basis einer XSD-Datei erstellen will, indem er indem er dem Konstruktor von DataSet die XSD-Datei als Argument übergibt. Jede XSD-Datei beschreibt ja die Struktur genau eines DataSets.
- Im Konstruktor wird DataSet::ReadXmlSchema aufgerufen und so das eigene Schema erstellt. Jedoch ist das DataSet noch untypisiert, d.h. die oben gelisteten Leistungsmerkmale zusätzlich zu DataSet::ReadXmlSchema sind noch nicht gegeben: Verwendung abgeleiteter Klassen und Zugriff auf DataTables, DataColumns und DataRelations über Attribute statt Collections!
- Im Konstruktor wird anschließend DataSet::makeTyped aufgerufen, um das untypisierte DataSet in sein typisiertes Pendant umzuwandeln: für alle DataTables, DataColumns und DataRelations wird das entsprechende übergeordnete Objekt (also DataTable für DataColumns und DataRelations, DataSet für DataTables) gegen eine Instanz einer dynamisch erzeugten, abgeleiteten Klasse ersetzt. Das ist in PHP möglich, selbst bei $this im Konstruktor! Diese dynamisch erzeugten Unterklassen bieten dann zusätzlich den Zugriff über Attribute (bzw. deren getter- und setter-Methoden) statt Collections.

Nur über den Konstruktor von DataSet soll der Aufruf von DataSet::makeTyped möglich sein, nicht aber für den Benutzer auf beliebigen bereits existenten Objekten - das nämlich führt bei bereits typisierten DataSets zur dynamischen Redefinition existierender Klassen, sicher kein guter Stil. Also wird DataSet::makeTyped eine private Methode.

Vor- und Nachteile dieses Designs

Vorteile dieses Designs

- Wesentlich einfachere Implementierung (nur bestehend aus

DataSet::ReadXmlSchema und DataSet::makeTyped). Typische Probleme der Codegenerierung werden so vermieden (s.u.).

- Klare Strukturierung der Aufgabe in zwei Teile: DataSet::ReadXmlSchema und DataSet::makeTyped.
- Wiederverwendung der vorhandenen Funktion DataSet::ReadXmlSchema. Natürlich ist es möglich, dass auch im Codegenerator des .NET DataSets vorhandener Code aus dem DataSet für diese ähnliche Aufgabe, ein XSD-Schema in ein DataSet-Schema umzusetzen, wiederverwendet wird.
- Leistet, was ein konventioneller Codegenerator auch leistet, jedoch noch mehr:
  - Die Typen für typisierte DataSets müssen nicht wie bisher zur Übersetzungszeit generiert werden, sondern können zur Laufzeit, also dynamisch, generiert werden. Der Umgang mit diesen Typen wird dadurch wesentlich flexibler. So muss z.B. zur Übersetzungszeit nicht mehr bekannt sein, welche Typen für typisierte DataSets es im Programm zur Laufzeit geben wird. (Anmerkung: natürlich gibt es keine eigentliche »Übersetzungszeit« in interpretierten Sprachen wie PHP; gemeint ist hier der Zeitpunkt vor Programmausführung, ab dem der Programmierer den Code nicht mehr ändert).
  - Schemaänderungen in typisierten DataSets sind prinzipiell so denkbar, dass auch auf die geänderten Schemaelemente über Attribute zugegriffen wird, wie von typisierten DataSets gewohnt.

  Der Name der typisierten DataSet-Klasse kann zur Laufzeit bestimmt werden.

  Weil alles zur Laufzeit geschieht, ist kein separater Codegenerator-Lauf notwendig um den erwünschten Effekt zu erzielen. Der Codegenerator kann so innerhalb des Programms bedient werden: automatisierte, softwaregesteuerte Verwendung. Somit eröffnen sich Einsatzmöglichkeiten, die ein separater Codegenerator für compilierte Sprachen nicht bietet: auch aus dynamisch generierten XSD-Dateien können nun typisierte DataSets erstellt werden. Für compilierte Sprachen müssen die XSD-Dateien zur Übersetzungszeit statisch vorliegen, um daraus typisierte DataSets zu erstellen.
- Die Implementierung orientiert sich an den Möglichkeiten und Besonderheiten der Zielsprache PHP als einer intrepretierten Sprache, um das Verhalten des Codegenerators möglichst einfach nachzubilden. Wer will, kann das eine

»elegante Lösung« nennen ...

- Die Implementierung verletzt trotz ihrer unkonventionellen Art, Typen zur Laufzeit zu generieren, keine Prinzipien objektorientierter Programmierung: der Aufruf des Konstruktors von DataSet führt zwar zur Generierung eines Objektes einer (dynamisch erzeugten) Unterklasse von DataSet. Dieses Objekt einer Unterklasse ist jedoch wie stets bei Ableitungen auch ein Objekt der Oberklasse, so dass »new DataSet()« (auch) das leistet, was die Codezeile erwarten lässt: ein DataSet-Objekt zu generieren.

Nachteile dieses Designs

- Das Design führt nicht zu einem Codegenerator im eigentlichen Sinn.
- Dynamisch generierte Typen sind eine unkonventionelle und damit gewöhnungsbedürftige Art zu programmieren. Unkonventionelle Techniken sind darüber hinaus oft experimentell, es fehlt ein breiter Erfahrungsschatz zum Umgang damit. Daraus ergeben sich evtl. schwer einschätzbare Risiken für die Verwendung der fertigen Software.
- »Typsicherheit zur Übersetzungszeit«, ein Hauptziel typisierter DataSets, kann aus prinzipiellen Gründen nicht in interpretierten Sprachen wie PHP realisiert werden.

Ausführliche Diskussion dieses Designs

Das grundlegende Codegenerator-Design und die Probleme dabei

Die naheliegendste Möglichkeit, eine XSD-Datei in PHP-Code umzusetzen, ist Codegenerierung »on the fly«: aus der XSD-Datei wird ein XML-Baum erzeugt, dieser wird traversiert und für jeden dabei besuchten Knoten wird »on the fly« entsprechender PHP-Quelltext erzeugt.
Grundsätzlich tritt dabei das folgende Problem auf: Codefragmente werden nicht in der Reihenfolge generiert, in der sie in die sequentielle PHP-Quelltextdate geschrieben werden können. Betrachten wir etwa den Fall, bei dem für einen Knoten den XML-Baums eine typisierte DataTable generiert werden muss. Dazu sind zwei Codefragmente zu generieren:

1. Die von DataTable abgeleitete Klasse selbst inkl. Konstruktor, Attributen usw..
2. Code für den Konstruktor des übergeordneten DataSets, der eine Instanz dieser

DataTable erzeugt und dem übergeordneten DataSet hinzufügt.

Mindestens eines dieser beiden Codefragmente kann nicht ans Ende der bisher erzeugten sequentiellen Datei geschrieben werden. Denn um den Code für den DataSet-Konstruktor sofort schreiben zu können, muss der XML-Baum top-down traversiert werden: oberstes Element des Baums ist das DataSet, so dass der generierte PHP-Code mit dem Konstruktor von DataSet endet, wenn beim top-down traversieren der Knoten erreicht wird, für den unsere typisierte DataTable zu generieren ist. Dann aber ist es nicht möglich, die typisierte DataTable-Klasse nach dem DataSet-Konstruktor in die sequentielle Datei zu schreiben: die zweite und jede folgende zu generierende DataTable fordert ja, dass die sequentielle Datei immer noch mit dem Konstruktor von DataSet endet.
Für dieses Problem, die Baumstruktur der XSD-Datei korrekt auf die sequentielle Struktur des PHP-Quelltextes abzubilden, sind mehrere Lösungswege gangbar:

- **Templating:** verwende eine PHP-Datei mit dem Code-Grundgerüst und besonderen Markern, die durch Codefragmente ersetzt werden, die »on the fly« während des Traversierens des XML-Baums generiert wurden.
- **Einfügen:** Code kann nicht nur am Ende einer sequentiellen Datei angefügt werden, sondern auch an weiteren Stellen eingefügt werden. Diese weiteren Stellen sind noch nicht abgeschlossene Codeblöcke, unter anderem: die Konstruktoren von DataSets und DataTables, die Attribute von DataSets und DataTables und ihre get- und set-Methoden, die Liste der typisierten DataRow-Klassen, die Liste der typisierten DataTable-Klassen, die Liste der typisierten DataSet-Klassen. Statt sich mehrere Positionen in der sequentielle Datei zu merken, an denen eingefügt werden kann, ist es für die Implementierung geschickter, die noch nicht abgeschlossenen Codeblöcke im Hauptspeicher in je einer eigenen Datenstruktur zu halten und sie erst zu einer sequentiellen Datei zu montieren, wenn alle Codefragmente generiert wurden.
- **Anderer Designansatz:** Codegenerierung »on demand« statt »on the fly«: statt den XML-Baum zu traversieren, traversiert man die bereits bekannte Grobstruktur des zu erzeugenden PHP-Quelltextes. Diese Grobstruktur erfordert an jeder ihrer sequentiell geordneten Positionen die Generierung eines geeigneten Codefragmentes. Dieses wird dann »on demand« mit Hilfe des XML-Baums erzeugt. So ist ggf. zwar ein mehrmaliges Traversieren des XML-

Baumes nötig, die Position generierter Codefragmente ist aber zum Zeitpunkt der Generierung vollständig bekannt.

- **Vereinheitlichen und bottom-up traversieren:** Es bietet sich an, den XML-Baum bottom-up zu traversieren, weil so der Code aller Details (tiefere Ebenen im Baum) vor dem der jeweils zusammenfassenden Elemente generiert werden kann, wie es von der Abhängigkeit her auch geboten ist. So würden alle typisierten DataRow-Klassen vor der ersten typisierten DataTable-Klasse generiert, alle typisierten DataTable-Klassen vor der typisierten DataSet-Klasse. Problematisch dabei, den Baum ebenenweise zu traversieren, beginnend von den Blattknoten ist: Ebenen des XML-Baums einer XSD-Datei müssen keine gleichartigen Elemente enthalten. So können in einer Ebene DataColumns als auch DataTables (als nested DataTables) auftreten. Letztere entsprechen einer separaten DataTable und einer DataColumn, mit der sie per DataRelation verbunden ist. Es ist also notwendig, den XML-Baum vor dem bottom-up Traversieren zu vereinheitlichen, um beim anschließenden bottom-up Traversieren alle Elemente einer Ebene gleichartig behandeln zu können und überhaupt wissen zu können, welche Elemente man gerade behandelt. Eine solche Vereinheitlichung wäre die eben dargestellte Umwandlung von nested DataTables in DataRelations.
- **Zwischendarstellung:** Unschön bei einem Codegenerator, der beim Traversieren des XML-Baums »on the fly« den fertigen Code erzeugt, ist: dieser Algorithmus ist komplex, weil er zuviel können muss. Er muss sowohl einen XML-Baum traversieren, mit Aufbau und Semantik von XSD umgehen können, PHP-Code generieren und ihn auch noch an der richtigen von mehreren Stellen ablegen. In solchen Fällen bietet sich eine Komplexitätsminderung an, die zwei getrennte Algorithmen und eine geeignete Zwischendarstellung als ihre Schnittstelle verwendet. Diese Zwischendarstellung soll »on the fly« beim Traversieren des XML-Baums erzeugt werden können und aus ihr sollen beim Traversieren »on the fly« die Codefragmente sofort in der richtigen Reihenfolge erzeugt werden können. Die für die Lösungsalternative »Vereinheitlichen und bottom-up traversieren« vorgeschlagene vereinheitlichte Struktur des XML-Baums ist bereits ein erster Ansatz für eine solche Zwischendarstellung: der darauf aufsetzende Algorithmus muss nicht mehr die gesamte Semantik von XSD kennen, sondern nur einen Teil.

Für den hier zu entwerfenden Codegenerator wurde der Lösungsweg »Zwischendarstellung« aus der obigen Liste gewählt. Die Gründe für diese Designentscheidung:

- Es gibt eine sehr geeignete Zwischendarstellung: Objekte untypisierter DataSets. Sie eignet sich insbesondere, weil das weitere Design dieses Codegenerators fordert, generierte Codefragmente sofort auszuführen statt in einer sequentiellen Datei abzulegen. Untypisierte DataSets sind dabei gerade die Objekte, auf denen die generierten Codefragmente ausgeführt werden müssen und die dabei zur Laufzeit in typisierte DataSets verwandelt werden.

- Die Zwischendarstellung durch Objekte untypisierter DataSets eignet sich dazu, beim Traversieren die Codefragmente »on the fly« zu generieren. Denn ein DataSet-Objekt mit seinem Schema aus DataTables und DataColumns hat dieselbe Baumstruktur wie der oben unter »Vereinheitlichen und bottom-up traversieren« vorgestellte vereinheitlichte XML-Baum einer XSD-Datei. Es kann also ebenso bottom-up traversiert werden, wobei »on the fly« Codefragmente erzeugt und ausgeführt werden, ohne dass irgendwelche Abhängigkeiten zwischen Codefragmenten verletzt würden.

- Die Zwischendarstellung durch Objekte untypisierter DataSets ist auch besonders geeignet, weil Algorithmus, der darauf operiert, keinerlei Kenntnisse mehr über Aufbau und Semantik von XSD benötigt. Beim alternativen Lösungsweg »Vereinheitlichen und bottom-up traversieren« war das noch zum Teil erforderlich.

- Die Zwischendarstellung durch Objekte untypisierter DataSets hat auch den Vorteil, dass sie ohne eigene, weitere Datenstrukturen auskommt. DataSets usw. existieren bereits und werden einfach für die Zwischendarstellung verwendet.

- Die Klasse DataSet hat bereits eine Methode, um die Zwischendarstellung zu erzeugen: DataSet::ReadXmlSchema erzeugt das Schema eines untypisierten DataSets aus einer XSD-Datei. Ein guter Teil der Arbeit des Codegenerators wird so von dieser Methode übernommen. Statt spezialisierten Code schreiben zu müssen kann hier eine vorhandene und für vielseitige Verwendung gedachte Methode eingesetzt werden.

Der Unterschied zum konventionellen Codegenerator-Design

Bis zur Generierung der Codefragmente folgt das Design des Codegenerators für phpDataSet einem üblichen Design. Nun aber weicht es ab: statt die generierten Codefragmente in einer sequentiellen Datei abzulegen, werden sie sofort ausgeführt. Ihr Effekt ist, die untypisierten DataSets, DataTables und DataRows der Zwischendarstellung in die typisierten Pendants zu verwandeln.
Dieser Schritt löst natürlich nicht das Problem, das die Zwischendarstellung lösen soll: die Anforderungen an die Reihenfolge der Codefragmente entsprechend ihren Abhängigkeiten (s.o.). Denn wenn ein Codefragment ausgeführt werden kann, muss aller Code, von dem es abhängig ist, schon vorher generiert und ausgeführt worden sein. Dabei ist es dann irrelevant, ob die Generierung zuerst in einer sequentiellen Datei protokolliert und diese dann ausgeführt wird, oder ob generierte Codefragmente sofort ausgeführt werden.
Ein Codegenerator, der den erzeugten Code nicht in einer Datei ablegt, ist natürlich kein Codegenerator im eigentlichen Sinn mehr. Wir behalten diese Bezeichnung hier trotzdem bei, weil sein Effekt dem des Codegenerators für typisierte DataSets in Microsoft .NET entspricht.
Insgesamt ergibt sich so eine zweistufige Arbeitsweise:

- In DataSet::ReadXmlSchema wird der XML-Baum traversiert und dabei »on the fly« ein untypisiertes DataSet-Objekt mit einem Schema versehen, das der XSD-Datei entspricht.
- In DataSet::makeTyped wird dann das Schema des untypisierten DataSets traversiert und »on the fly« der zum jeweiligen Schemaelement gehörende PHP-Code erzeugt und sofort ausgeführt.

Hinweise zum Vorgehen bei der Implementierung

Es bietet sich an, die Implementierung entsprechend der zweistufigen Arbeitsweise dieses Codegenerators zweizuteilen:

- Implementierung von DataSet::ReadXmlSchema entsprechend dem Vorbild aus Microsofts .NET-DataSet.
- Implementierung einer Methode DataSet::makeTyped, die die eigene Instanz eines DataSets in die typisierte Variante überführt.

Diese beiden Teilaufgaben können völlig unabhängig voneinander gelöst werden. Auch eine Teillösung ist bereits brauchbar; dazu bietet sich besonders DataSet::makeTyped an. Denn diese Methode ist die Besonderheit im Codegenerator, während DataSet::ReadXmlSchema eigentlich noch zur Implementierung der Klasse DataSet nach dem Vorbild aus Microsofts .NET-Framework gehört. DataSet::makeTyped ist auch ohne DataSet::ReadXmlSchema einsetzbar, nämlich auf DataSet-Objekten, deren Schema durch eigenen Code statt mit Hilfe einer XSD-Datei erstellt wurde. Bisher implementiert ist DataSet::makeTyped, jedoch fehlen noch die Hilfsfunktionen DataTable::makeTyped und DataRow::makeTyped sowie deren Hilfsfunktionen.

Hinweise zur Implementierung von DataSet::ReadXmlSchema

Das Format der XSD-Dateien für Microsofts .NET-DataSets enthält Bestandteile, die vereinfachend ausgelassen werden können oder sogar sollten. So sollten etwa die Microsoft-spezifischen msdata:-Attribute nicht zulässig sein.
Der Artikel »Generating DataSet Relational Structure from XML Schema (XSD)« aus dem Microsoft Developer Network (MSDN) dokumentiert die Verwendung von XSD-Dateien und den daraus generierten typisierten DataSets.
Eine grundlegende Einführung in die manuelle Erstellung von XSD-Dateien anhand gegebener XML-Dateien bietet:
http://www.w3schools.com/schema/schema_example.asp.
Der Artikel »Generating a Strongly Typed DataSet« (MSDN) dokumentiert, wie man typisierte DataSets per Codegenerator aus XSD-Dateien erzeugen kann. Eine XSD-Datei im Quelltext ist enthalten.
Eine weitere, wesentlich umfangreichere XSD-Datei findet sich in »Using Annotations with a Typed DataSet« (MSDN).

Projektmanagement

3.1 Projektdefinition

Der Projektleiter hat in der Projektdefinition einige Vorgaben, Rahmenbedingungen und Eingangsparameter für die nachfolgende Planung festgelegt. Darin wird das Projekt- und Funktionsziel genannt – nämlich die Implementierung des .NET-DataSet Konzeptes in PHP und die Gruppenmitarbeiter und der Projektleiter genannt. Des Weiteren werden – soweit vorhanden, Zeitpunkte genannt, zu denen Ergebnisse vorliegen müssen, z.B Anfang und Ende des Projekts und Abgabe der Dokumentation. Außerdem werden Rahmenbedingungen festgelegt, um die Kommunikation via E-Mail zu vereinfachen und eine Vorlage für den Projektstatusbericht erstellt, den jedes Projektmitglied auszufüllen hat. Diese Berichte werden dem Auftraggeber wöchentlich per E-Mail übermittelt.

Technische Voraussetzungen:
Jedes Projektmitglied muss zur Implementierung eine .NET Umgebung und eine PHP-Umgebung auf einem Rechner installiert haben.

Projektbeginn und -ende:
Am 5.4.2004 findet eine Einführung in PHP und .NET statt. Mit diesem Background können Teilaufgaben von mir gestellt und von den Gruppenmitgliedern bearbeitet werden. Ende des Projekts ist Ende Juni und die Dokumentation wird Ende Juli abgegeben.

Rahmenbedingungen:
Um die Projekt bezogenen E-Mails besser filtern zu können, werde ich den E-Mail Verkehr ein wenig regulieren:

In jeder E-Mail, die sich auf das Projekt bezieht, ist in der Betreffzeile [.NET Praktikum] einzufügen. Bitte nicht die Klammern vergessen.
Außerdem beträgt die E-Mail Antwortzeit 24 Stunden, d.h. wenn ich in einer E-Mail eine Antwort verlange (Bericht / Status), soll diese innerhalb von 24 Stunden bei mir eintreffen.
Ferner möchte ich von jedem Projektmitglied einen wöchentlichen Statusbericht, in dem ich über den Projetktfortschritt informiert werde, bis Freitag 22 Uhr entsprechend beigefügter Vorlage erhalten.
Mit der Verabschiedung des Projektantrages durch Herrn Kaufmann wird dieser zum Projektauftrag, das Projekt ist gegründet.

3.2 Vorlage Statusreport:

In diesem Bericht werden die Aufgaben , die aktuell bearbeitet werden, ihr Bearbeitungsstatus und der
wöchentliche Zeitaufwand aufgelistet.
Anhand dieser Angaben kann man analysieren (siehe später), inwieweit die Zeitschätzungen eingetroffen sind.

Gruppe:	Dataset-Konzept		Datum:	
Projektleiter:	Martin Cebulla		Autor:	
Bearbeitete Aufgaben:	Aufgabe:			
	Neuer Status:			
	fertig am:			
	Aufgabe:			
	Neuer Status:			
	fertig am:			
	Aufgabe:			
	Neuer Status:			
	fertig am:			
Zeitaufwand diese	Woche (in Std.):			
Bemerkungen:				

3.3 Schätzungen

3.3.1 Aufwandsschätzung

Zu Beginn des Projektes wurden die Arbeitspakete identifiziert:

- .NET: Beispielerstellung eines Datasets
- .NET: Codegenerierung
- .NET: Struktur (Schema aus .xsd) rauslesen, Klassenbeschreibung, Methodenbeschreibung – entsprechende Algorithmen aus dem Codegenerator sollten möglichst gut beschrieben sein: Grundlage / Dokumentation
- PHP: Einarbeitungszeit zu Beginn des Projekts
- PHP: öffentliche Schnittstellen der Klassen und ebengenannte Beschreibung ist Grundlage für die Methoden- und Klassenimplementierung in php
- PHP: Formular in php als Test und Veranschaulichung
- .NET + PHP Klassen- und Methodenbeschreibung schnell fertigstellen, damit PHP eine Grundlage zum Arbeiten hat
- Austausch nach 4 (spätestens 5) Wochen: .NET: Anlernen in PHP, PHP: eventuell Einstieg in .NET

In der Aufwandsschätzung werden diese Arbeitspakete über einen Zeitraum von 12 Wochen (Projektdauer) zeitlich sinnvoll angeordnet.
Die 5. Woche erscheint zweimal auf dem Plan, da in dieser Zeit die .NET-Gruppe über die technischen Möglichkeiten von PHP informiert wird.

Zweck: Informationsaustausch über die beiden Programmierkonzepte.

Diese Schätzung ist bis auf gelegentliche Verschiebungen von einer Woche in dieser Form eingetroffen.
Der Punkt „Formular in PHP“ konnte nicht realisiert werden, da keine grafische Oberfläche implementiert wurde.
In der 7. – 9. Woche – in der die .NET-Gruppe eigentlich „arbeitsfrei“ war – wurden noch einige benötigte Informationen (u.a. schrittweiser Aufbau von DataSets, siehe auch 3.5) gefordert.
Da die .NET-Gruppe zu dieser Zeit unbeschäftigt war, bedeutete dieser zusätzliche Arbeitsaufwand keine Verschiebung in der Aufwandsschätzung.

3.3.2 Zeitschätzung zu Projektbeginn

Zu Beginn des Projektes hat der Projektleiter folgenden Zeitumfang geschätzt, den das Projekt zur Bearbeitung benötigen wird: Von zwölf Wochen ist eine Woche an Orientierung in Bezug auf die Aufgabenstellung abzuziehen. An Organisation und Veranstaltungsbesuch ist pro Person wöchentlich 1,5 Stunden zu rechnen. Die PHP-Gruppe sollte für die Einarbeitung in PHP und für die Nachimplementierung der DataSets 150 Stunden benötigen.
Die .NET-Gruppe sollte für die Einarbeitung in C# und .NET und für die Codeanalyse 100 Stunden benötigen. Hier wurde weniger Zeitaufwand vermutet, da hauptsächlich vorgegebener Code zu analysieren war, währenddessen die PHP-Gruppe eigenen Code und die Konzepte dafür realisieren muss.
Zusammengefasst beträgt die Gesamtarbeitszeit (inklusive Organisation) der PHP-Gruppe voraussichtlich 195 Stunden und der .NET-Gruppe 145 Stunden. Hier wird von einer realistischen Schätzung ausgegangen, ohne dass ein Puffer eingebaut ist.

3.3.3 Berechnung des tatsächlichen Zeitaufwandes

Nach Projektende kann man aus den Projektstatusberichten die gesamte Arbeitszeit der Projektmitglieder berechnen (in Stunden):

	Martin C.	Nadja S.	Daniela M.	Andreas K.	Matthias A.	Susanne M.
Organisation ** und Veranstaltungsbesuch	28	20	20	20	29,37	20
Programmieren *	52	51	57	49	100,25	44
Projektmanagement	16	---	---	---	---	---
Gesamtdauer	96	71	77	69	130,02	66

Unter Programmieren * ist in Bezug auf die PHP -Gruppe die Programmierleistung gemeint und in Bezug auf die .NET-Gruppe die Arbeit unter .NET. Unter Organisation ** ist Gruppentreffen zu verstehen.
Die Gesamtarbeitszeit berücksichtigt nicht die Arbeit an der Dokumentation und am Pflichtenheft.

3.3.4 Soll / Ist Vergleich

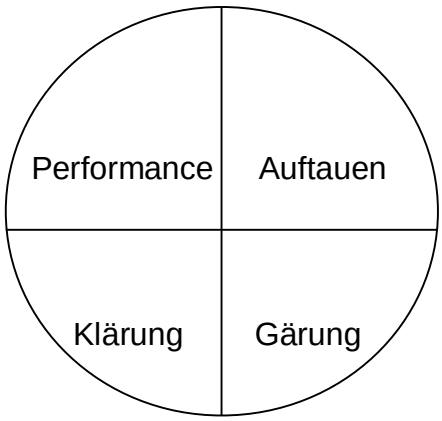
Aufgrund der Zeitschätzung und der Zeitberechnung kann man nun erörtern, inwieweit sich der geschätzte Zeitaufwand bewahrheitet hat.
Addiert man die tatsächlichen Zeiten der .NET Gruppe, kommt man auf 244 Stunden. Bei einer geschätzten Zeit von 145 Stunden hat sich der Projektleiter um ca verschätzt.
Addiert man die tatsächlichen Zeiten der PHP Gruppe, kommt man auf 241,02 Stunden.
Bei einer geschätzten Zeit von 195 Stunden hat sich der Projektleiter um ca $^{1}/_{5}$ verschätzt.

3.4 Meinungen über die Arbeitsweise Projektmanagement

Für alle Projektmitglieder war die Arbeitsweise in einem Projekt neu. In der Veranstaltung Software-Technik 2 war zwar eine ähnliche Arbeitsweise vorgesehen, jedoch gab es nicht zwingend ein Projektleiter, sodass keine einheitliche Struktur nötig war. Außerdem verfügten die Studenten zu dieser Zeit noch nicht über das benötigte Programmierverständnis, sodass der Lerneffekt etwas mager ausfiel.
In der Zeit des Schwerpunktpraktikums befanden sich alle Projektmitglieder im 6. Semester, die sich im Verlauf des Studiums ausreichendes Programmier-Knowhow angeeignet hatten.
Durch die klare Aufteilung in Projektmitarbeiter und Projektleiter, der regelmäßig Rücksprache mit dem zuständigen Professor halten musste, kam eine regelmäßige und strukturierte Arbeit zustande.

3.5 Meinung des Projektleiters

Die Kommunikation und den Verlauf der Arbeit wird anhand der Teamuhr dargestellt.
Die Phasen bedeuten im Einzelnen:
Auftauen: sich gegenseitig kennenlernen (hängt von der Größe ab)
Gärung: Klarlegen der eigenen Ziele (hier entstehen Konflikte)
Klärung: Wenn bestimmte Aspekte bei den vergebenen Aufgaben unklar sind (z.B. Zuständigkeitsbereiche) müssen Vereinbarungen getroffen werden.
Performance: Alle Aufgaben sind klar verteilt, es gibt darüber keine Diskussionen mehr, die Projektmitarbeiter arbeiten mit hoher Effizienz.



Das Auftauen war kein Problem, denn die meisten Teammitglieder kannten sich aus früheren Veranstaltungen. Diejenigen, die sich nicht vertraut waren, fanden sich schnell in die Gruppe ein.
In der Gärungsphase identifizierte der Projektleiter die Arbeitspakete und integrierte sie in einen Zeitplan (s. Aufwandsschätzung). Die Klärungsphase begann, als die PHP-Gruppe versuchte anhand der durchdokumentierten Codeanalyse mit der Umsetzung der DataSets in PHP zu beginnen. Hier herrschten Unklarheiten, schließlich entwarf die .NET-Gruppe eine Vererbungshierarchie der DataSet-Klassen, eine tabellarische Gesamtübersicht und einen schrittweisen Aufbau von DataSet-Beispielen zur Unterstützung der PHP-Gruppe.
Nun hatte die PHP-Gruppe genügend Input einerseits aus der Einarbeitung in PHP, andererseits ausreichend Know-How von der .NET-Gruppe, um in den letzten vier Wochen produktiv arbeiten zu können (Performance).

3.5 Meinung der Projektmitarbeiter

Susanne

Bei einer Projektaufgabe an der mehrere Mitglieder beteiligt sind ist ein Projektleiter immer vorteilhaft, da dieser einen Überblick über das ganze Projekt mit seinen Fortschritten und Problemen hat und so die Projektgruppe koordinieren kann.

Auch bei diesem eher kurzen Projekt mit einer kleinen Projektgruppe war ein Projektleiter sinnvoll. Gleich zu Beginn wurde von ihm festgelegt, wann und wie regelmäßige Zwischenberichte abgegeben werden sollten damit immer ein aktueller Stand des Projektes bekannt war.
Des Weiteren war es auch positiv, dass sich unser Projektleiter regelmäßig mit dem Auftraggeber treffen musste und so genau wusste was erwartet wurde und mögliche Änderungen an die Mitglieder weiterleiten konnte.
Ebenfalls wurde von unserem Projektleiter eine Zeitschätzung aufgestellt, an die man sich richten konnte, die jedoch, wie beim ersten Projekt zu erwarten, etwas zu optimistisch ausgelegt war.
Auch verständlich ist, dass es zu Beginn des Projektes einige Anlaufschwierigkeiten gab, was darauf zurückzuführen ist, dass sich die Projektmitglieder nicht alle kannten

und vor allem die Mitglieder der PHP-Gruppe anfangs alleine in die Programmiersprache PHP eingearbeitet haben.

Zusammenfassend würde ich sagen, dass unsere Projektgruppe gut zusammengearbeitet hat und unser Projektleiter seine Aufgabe gut gemacht hat. Weiter würde ich auch für jedes Projekt, egal wie klein es ist, ein Projektmanagement empfehlen.

Meinung: Daniela

Dieses interessante Projekt zeigte, daß wir nach Softwaretechnik 2 doch recht gute Fortschritte gemacht haben, was Teamfähigkeit angeht und auch das Koordinieren der Aufgaben unten den einzelnen Projektmitgliedern. Die Projektarbeit im Team muß wirklich geübt werden, und ich finde es gut, daß man während des Studiums ein paar Mal die Möglichkeiten hat, sich hierbei zu verwirklichen. Nicht zuletzt lag es an unserem Projektleiter, der die Organisation gut gemeistert hat und die Fähigkeit besaß, auch in schwierigen Zeiten die Projektmitglieder zu motivieren, daß unser Projekt so erfolgreich verlief. Natürlich mussten die typischen Anfangsschwierigkeiten überwunden werden. Die einzelnen Projektmitglieder haben zuvor nie zusammengearbeitet. Man musste sich erstmal untereinander näher kennenlernen, Stärken und Schwächen abschätzen, damit die Zusammenarbeit besser funktionieren konnte. Hinzu kam, daß man sich mit neuer Materie befassen mußte, was eine zusätzliche Herausforderung darstellte.
Ich kann sagen, daß unsere Gruppe viel über .Net und Webserver gelernt hat und, daß das Semester dadurch wirklich eine Bereicherung an Wissen war. Das Projekt hat auch gezeigt, wie wichtig Projektmanagement ist und, daß Studenten, die damit noch keine Erfahrung gesammelt haben, dieses unbedingt noch als Kurs belegen sollten.

Matthias

Das Projekt phpDataSet begann als Schwerpunktpraktikum an der FH Gießen-Friedberg, Studiengang Informatik. Dieses Dokument beschreibt Erfahrungen mit dem gruppeninternen Projektmanagement aus Sicht von Matthias Ansorg, einem Gruppenmitglied der Teilgruppe »PHP«.

Über die Teilung in zwei Teilgruppen: Im Projekt phpDataSet wurde zu Beginn entschieden, die sechs Teilnehmer auf zwei Teilgruppen aufzuteilen: die .NET-Gruppe sollte sich mit der Analyse von DataSets inkl. Codegenerator in Microsoft .NET beschäftigen, die PHP-Gruppe sollte die Analyseergebnisse in eine Reimplementierung in PHP umsetzen. Gegen Ende des Projektes halte ich eine Teilung in Gruppen von 2-3 Mitgliedern immer noch für sinnvoll, weil es in einem »improvisierten Umfeld« wie der studentischen Softwareentwicklung Flexibilität bewahrt, wohingegen z.B. Gruppentreffen für 6 Studenten gleichzeitig sich schon als Problem erweisen (so geschehen zu Beginn unseres Projektes). Jedoch meine ich, dass die Gruppen ungünstig eingeteilt wurden. Jede Gruppenteilung bildet eine Engstelle der Kommunikation. Sie sollte also so erfolgen, dass möglichst wenig Kommunikation über diese »Schnittstelle« gehen muss. Ähnlich wie beim Klassenentwurf fordert das eine »thematisch kohärente« Gruppeneinteilung: nicht nach Art der Tätigkeit (Beschäftigung mit .NET oder Programmieren in PHP), sondern nach Thema. Etwa »DataSet Basisfunktionalität« und »DataSet Codegenerator«. Jede Teilgruppe würde dann sowohl Analysten als auch PHP-Programmierer enthalten.
Die beschriebenen alternativen Arten der Gruppeneinteilung entsprechen dem Unterschied zwischen klassischer verrichtungsorientierter Organisation und objektorientierter Organisation in Unternehmen. Die hier als ungünstig dargestellte verrichtungsorientierte Organisation führte während der Laufzeit des Projektes u.a. zu folgenden Problemen, hauptsächlich begründet in Problemen der gruppenübergreifenden Kommunikation:
Zu geringe Auslastung der PHP-Gruppe zu Projektbeginn, bevor erste Analyseergebnisse der .NET-Gruppe vorhanden waren.

Missverständnisse bei der Implementierung des Codegenerators, die zu unnötigem Arbeitsaufwand führten. Die PHP-Gruppe hatte Dokumente mit einer Beschreibung des Algorithmus des Codegenerators für typisierte DataSets im Microsoft .NET Framework von der .NET-Gruppe angefordert. Erst als diese fast fertig waren fand die PHP-Gruppe heraus, dass sich unter PHP ein ganz anderes Design für einen Codegenerator anbot, mit einem anderen, wesentlich kompakteren Algorithmus.

Unzureichende Absprachen und ein ebenfalls unzureichendes Verständnis für die Arbeitsweise der jeweils anderen Gruppe führten dazu, dass die Analyseergebnisse der .NET-Gruppe wesentlich weniger hilfreich bei der Implementierung des

Codegenerators in PHP waren als eigentlich gedacht.

Abschließend möchte ich anmerken, dass ich selbst zu Beginn des Projektes die Entscheidung über die Art der Teilung zu verantworten hatte und sie zu diesem Zeitpunkt interessanterweise für gut hielt.
Der Versuch, Gruppenmitglied zu sein: Ich habe zu Beginn des Projektes bewusst abgelehnt, die Rolle des Projektleiters zu übernehmen. Denn dieser Posten ist nach meinen bisherigen Erfahrungen als Projektleiter in studentischen Projekten stets mit einem Mehraufwand verbunden, zu dem ich in diesem Semester schlichtweg keine Zeit hatte. So fand ich mich also in der Position eines Gruppenmitglieds, d.h. dass ich diesmal das Projekt nicht selbst organisieren konnte. Nach anfänglichen Schwierigkeiten konnte ich mich doch daran gewöhnen; dazu trug der kooperative Führungsstil unseres Projektleiters bei und die Tatsache, dass die PHP-Gruppe weitgehend selbständig arbeiten konnte und mir dort informell die Aufgabe des Analytikers und Organisators zufiel.
Über bessere Projektarbeit: Softwareentwicklungs-Projekte, selbst in ihrer »prototypischen« studentischen Form, sind gute Ideenlieferanten. Deshalb abschließend einige spontane Verbesserungsvorschläge für (studentische) Projektarbeit in der Softwareentwicklung:

7. Man verwende zur Kollaboration ein auf Projektarbeit spezialisiertes Internet-Portal. Unter den vorhandenen, kostenfrei nutzbaren Portalen bietet sich http://www.sourceforge.net an.

8. Webbasiertes CVS im Portal zum Versionsmanagement.

9. ToDo-Liste und Bugtracking-System zur Aufgabenverteilung im Team.

10. Zeittracking im Portal für alle Gruppenmitglieder, um zu große Unterschiede in der Lastverteilung ausgleichen zu können auch wenn nicht gemeinsam programmiert oder gearbeitet wird. Die aufgewandte Zeit wird jeweils Einträgen der ToDo-Liste zugeordnet.

11. Automatisch erstellte Statusberichte. Dazu werden die Informationen des Zeittracking-Systems und der ToDo-Liste durch ein Programm ausgewertet und daraus ein Bericht für den Projektleiter erstellt. Die leicht vergessenen manuellen Statusberichte und ihre manuelle Auswertung / Zusammenfassung durch den Projektleiter entfällt.

12. Parallelführung von Forum und Terminkalender des Portals über eine Mailingliste,

um häufiges, unbequemes Login nur um Neuerungen zu erfahren unnötig zu machen. In einem anderen studentischen Projekt machte ich die Erfahrung, dass dies kritisch für den erfolgreichen Einsatz einer Groupware (bzw. eines Portals) ist.

13. Pair Programming nach XP-Vorbild, wo gegenseitige Hilfe im Team notwendig ist. Das haben wir in unserem Projekt gegen Projektende mit gutem Erfolg probiert.

Andreas

Der subjektive Eindruck bei der Arbeit unter einem Projektleiter war gut. Es ist gut zu sehen, dass man bei größeren Projekten einen Leiter benötigt, der den Umfang der Arbeit gut einschätzen kann. Durch die Abhängigkeit von anderen Personen ist es zwingend notwendig, das man eine zentrale Person hat, die als zentraler Ansprechpartner bei Problemen fungiert. Eben diese Person ist auch dafür verantwortlich, dass der zeitliche Rahmen eingehalten wird. Gerade der zeitliche Rahmen ist mit die wichtigste Komponente bei der Softwareentwicklung, da für ein softwareproduzierendes Unternehmen, gerade der kleineren Art, die Einhaltung der Fristen wichtig ist. Sicher ist die Ausführung des Jobs des Projektleiters einfacher, wenn er in einem Unternehmen besseren Zugang zu den Leuten und dem Produkt hat. In der Form des PHP-Projekts war es für den Projektleiter schwer, den genauen Status des Entwicklung zu überblicken, trotz Statusberichte der einzelnen Mitglieder. Eine Sache, die mir wenig gefallen hat, war die Spaltung der Gesamtgruppe in zwei einzelne Gruppen, die mehr oder weniger selbstständig gearbeitet haben. So konnte man seitens der PHP-Gruppe nur wenig Interesse der anderen Gruppe an den erstellten Dateien verzeichnen, was vielleicht aber auch mit der Aufgabenstellung zu tun hatte.

Etwas vermisst hatte ich einen Fahrplan, mit Wegpunkten zur Realisierung des Projekts. Die großen Punkte wurden durch den Kursleiter vorgegeben, von dem Projektleiter wurden konkrete Daten nicht festgelegt.

Prinzipiell würde ich sehr gerne die Erfahrung machen, gerade im Softwarebereich, wie ein solches Projekt in der Praxis in einem Unternehmen abläuft. Es wird mit Sicherheit viel mehr Probleme geben, wahrscheinlich auch im zwischenmenschlichen Bereich, da man sich hier die Gruppenmitglieder nicht aussuchen kann.