Reengineering einer internetbasierten Mensabestellsoftware für Schulen

Bachelorarbeit (Informatik)

Armin Feistenauer
Matrikelnummer: 4139614

Betreuer: Prof. Dr. Lutz Prechelt

Institut für Informatik, Freie Universität Berlin
am
22. April 2010

Inhaltsverzeichnis

1. Einführung

1.1. Thema der Arbeit

Das Thema meiner Bachelorarbeit lautet „Reengineering einer internetbasierten Mensabestellsoftware für Schulen“. Der Begriff Reengineering bedeutet dabei im Kontext der Softwareentwicklung die Untersuchung und Anpassung eines Softwaresystems mit dem Ziel die Softwarequalität zu verbessern und/oder neue Funktionalität zu implementieren.

> *„[Reengineering is] the examination and alteration of a system to reconstituite it in a new form.“ Chikofsky und Cross [1]*

Die im Arbeitstitel genannte internetbasierte Mensabestellsoftware ist eine von Sven Seeberg und mir in PHP entwickelte Webanwendung, mit der auch Essensbestellungen getätigt und abgerechnet werden. Die Mensafunktionalität ist ein Modul des Gesamtsystems, das den Namen „elektronische Schulverwaltung“ kurz ESV trägt. Die Funktionalität und Entstehungsgeschichte von ESV werde ich im Abschnitt „Evaluation der vorhandenen Software“ genauer vorstellen.

Um im Rahmen einer Bachelorarbeit zu bleiben beschränke ich mich beim Reengineering von ESV auf die Basisfunktionalitäten der Benutzerverwaltung und des Bestellvorgangs. Außen vor gelassen wird im Besonderen die Verwaltung und Sicherheit der Bankdaten, da diese schwerlich im Rahmen der Bachelorarbeit in ausreichender Tiefe behandelt werden kann sowie das Programm zur Essensausgabe, das die im System erfassten Daten vor Ort verwendbar macht.

1.2. Reengineering in der Wissenschaft

Reengineering ist ein Begriff der sowohl in der Softwareentwicklung als auch für Geschäftsprozesse verwendet wird. Mich interessiert hier natürlich der Stand des Felds im Softwarebereich.

Software-Reengineering wird in der wissenschaftlichen Literatur meist als Prozess verstanden, der die Wartung eines Softwaresystems vereinfachen soll. Das ist bedeutend, da Software-Wartung einen Anteil von etwa 50-80 % an den Kosten über den gesamten Lebenszyklus zugeschrieben wird.

Hier steckt also ein enormes Sparpotenzial.

In die wissenschaftliche Ausbildung ist Software-Reengineering mindestens in Bremen und Stuttgart vorgedrungen. An den Universitäten dieser beiden Städte gibt es Vorlesungen zu genau diesem Thema.

Die Arbeiten beschäftigen sich in der Regel mit diversen Möglichkeiten den Reengineeringprozess möglichst kostengünstig zu gestalten. Dabei geht es um die Anwendung von graphenbasierten oder CASE-Werkzeugen.

1.3. Vorgehensweise

1.3.1. Der Reengineeringprozess

In der oben aufgeführten Definition von Reengineering wird ein zumindest zweiphasiger Prozess deutlich. Ich bin jedoch der Meinung, dass sich drei Phasen besser zur Darstellung der Prozessschritte eignen.

1.3.1.1. Phase 1: Untersuchung

Als erste Phase gilt die Untersuchung der vorhandenen Software um die Funktionalität und Struktur des Programms zu erkennen. Wenn hier nicht auf Spezifikationen oder eine Dokumentation der Software zurückgegriffen werden kann, wird diese Phase auch als Reverse Engineering bezeichnet. Unter Reverse Engineering versteht man die Analyse eines Softwaresystems um, unter Umständen aus dem Quellcode, ein Modell mit höherem Abstraktionsgrad zu erstellen.

1.3.1.2. Phase 2: Strukturveränderung

Bevor man danach jedoch zur Anpassung des Systems kommt, muss in meinen Augen eine weitere nicht explizit erwähnte Phase stehen: die Veränderungsphase, in der das in der ersten Phase gewonnene Modell verbessert und angepasst wird, bevor man mit der Implementierung beginnt. Wenn die Funktionalität und Struktur nicht verändert werden soll, kann diese Phase natürlich übersprungen werden, dann stellt sich aber die Frage worin sich Reengineering und Refaktorierung unterscheiden sollen. Refaktorierung bezeichnet die Verbesserung der Struktur von Quelltexten ohne Veränderung der Funktionalität.

1.3.1.3. Phase 3: Anpassung

In der Anpassungsphase findet dann die tatsächliche Veränderung des Quellcodes statt. Aufbauend auf den ersten zwei Phasen werden die modellierten Strukturen und Verhaltensweisen nun implementiert.

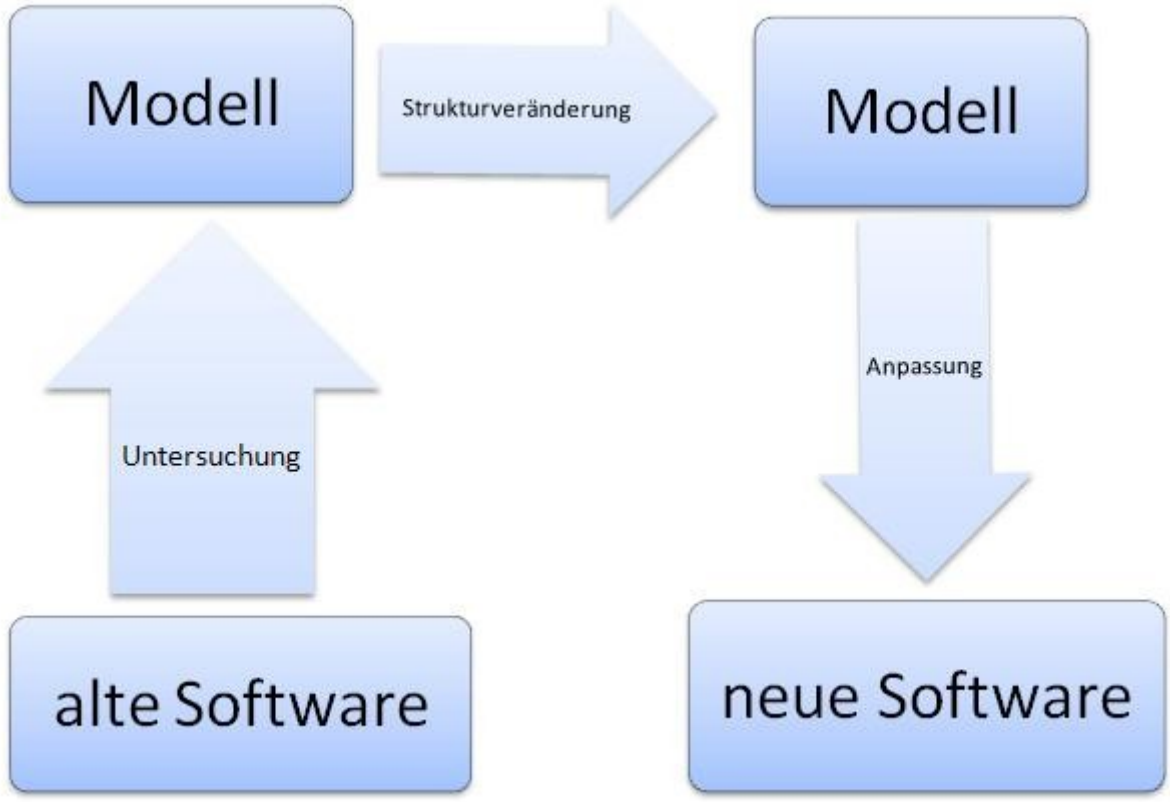

Abbildung 1: Der Reengineeringprozess

Wie sich in meinen Ausführungen zum Reengineeringprozess gezeigt hat, gibt es keine einheitliche Definition der Arbeitsschritte des Reengineerings und nur eine verschwommene Grenze zu anderen Techniken wie Reverse Engineering und Refaktorierung.

1.3.2. Die Ziele

Wie bereits oben erwähnt verfolgt Reengineering meist mindestens eins dieser Ziele.

1. **Die Softwarequalität verbessern**
   Der Begriff Softwarequalität hat viele Facetten und kann verschieden aufgefasst werden. Im Groben wird zwischen der Qualität des Designs und der Implementierung unterschieden. Es gibt viele verschiedene Qualitätsfaktoren die mit unterschiedlichen Metriken gemessen werden.
   Für das Reengineering ist oft das Qualitätsmerkmal der Wartbarkeit von besonderer Bedeutung, da Software durch Funktionsanpassungen einem Alterungsprozess unterliegt, der die Softwarequalität verschlechtert. Um diese wieder zu verbessern und weitere Wartungen zu ermöglichen bzw. zu vereinfachen kann Reengineering eingesetzt werden.
2. **Den Funktionsumfang verändern/erweitern**
   Wenn in einem vorhandenen Softwaresystem neue Anforderungen realisiert werden sollen muss der Funktionsumfang angepasst werden. In engeren Definitionen von Reengineering dient der Prozess nur zur Vorbereitung daran anschließender Funktionserweiterungen. Liberalere Definitionen erlauben die Veränderung der Funktionalität schon als Teil des Reengineeringprozesses.
3. **Wechsel der Plattform**
   Manche Programme sollen auf andere Hardware, z. B. von PC auf Mobiltelefon, oder andere Software, z. B. von Flash auf Silverlight, portiert werden. Auch hier kann man von Reengineering sprechen.

1.3.3. Mein Reengineeringplan

In meiner Bachelorarbeit werde ich mich an den von mir oben skizzierten Phasen des Reengineerings orientieren.

Bevor das Reengineering startet, werde ich mit einigen Beispielen darlegen, warum ein Reengineering von ESV notwendig ist. Hier muss ich von meiner persönlichen Erfahrung und meinem subjektiven Empfinden abstrahieren und objektive Probleme im System identifizieren.

Nachdem die Notwendigkeit offensichtlich ist, werde ich den Reengineeringprozess starten.

Zuerst werde ich die ESV-Komponenten analysieren und in diverse Diagramme überführen. Dabei hilft mir der vorhandene Quellcode sowie meine Erfahrung mit dem System.

Dann werde ich die Funktionalität der Modelle verändern, damit dies den tatsächlichen Anforderungen der Kunden besser entspricht. Hierfür kann ich auf das Nutzungsverhalten diverser Schulen und deren Wünsche an unser System zurückgreifen.

Das so entstehende Modell wird dann mit testgetriebener Entwicklung in Ruby on Rails implementiert.

Anschließend soll die Wirksamkeit des Reengineerings in meinem Fall durch eine Evaluation der Softwarequalität vorher und nachher überprüft werden.

Mit dieser Arbeit verfolge ich also folgende Ziele:

1.3.3.1. im Bereich Softwarequalität

- Verbesserung der Struktur durch objektorientierte Programmierung
- Zuverlässigkeit und geringe Fehlerrate durch eine höhere Testabdeckung
- Wartbar- und Erweiterbarkeit
- Verständlichkeit des Quellcodes durch Kommentare und Programmierstil

1.3.3.2. im Bereich Funktionalität

- Die Bedienung soll weniger komplex sein
- Die Benutzerführung soll vereinheitlicht werden
- Ungenutzte Funktionalität soll entfernt werden
- Das System soll bessere, mehr und einheitlichere Rückmeldungen an den Benutzer liefern

1.3.3.3. im Bereich Plattformwechsel

- Wechsel der Programmiersprache von PHP auf Ruby
- Wechsel von selbst geschriebenem PHP-Projekt zu Ruby on Rails als Framework
- Wechsel von Script auf objektorientierte Programmierung
- Einführung einer MVC (Model-View-Controller) Struktur

1.3.4. Erwartete Herausforderungen

Jedes Softwareprojekt hat diverse Risikofaktoren, deren Erkenntnis und Behandlung kritisch für den Projekterfolg sein können. Daher möchte ich mir so früh wie möglich Gedanken über die größten Herausforderungen meiner Arbeit machen um diesen Risiken vorzubeugen, bzw. geeignete Antworten auf die Probleme zu definieren.

Bei der Evaluation von ESV erwarte ich Probleme damit, objektive Metriken zu finden, um die Softwarequalität zu messen. Das liegt daran, dass ESV nicht objektorientiert implementiert ist, sich die meisten mir bekannten Metriken aber auf objektorientierten Quellcode beziehen. Problematisch ist auch, dass ich nur einen kleinen Teil der Funktionalität untersuchen will, im Softwaresystem selbst aber keine so klare Trennung der Komponenten stattfindet.

Diesem Problem kann ich aber relativ unbeschwert entgegentreten, da die Evaluation der erste Schritt ist und daher nicht unter dem größten Zeitdruck stattfindet. Ich muss nur rechtzeitig einen guten Kompromiss zwischen objektiver Evaluation und dem restlichen Projekt finden. Deshalb setze ich mir eine Deadline um hier nicht zu viel Zeit liegen zu lassen und bereits am Anfang in Verzug zu kommen.

Bei der Implementierung der neuen Software möchte ich auf Ruby on Rails zurückgreifen. Dieses Framework liegt gerade in der Version 3.0 Beta vor, die ich mit Blick auf die Zukunftssicherheit benutzen möchte. Der Betastatus kann zum Problem für mich werden, falls das System nicht stabil läuft und falls mir dadurch diverse Plug-ins zur Erleichterung alltäglicher Webentwicklungsaufgaben (noch) nicht zur Verfügung stehen.

Deshalb beobachte ich die Reaktion der Ruby on Rails Community auf den Release der Betaversion. Die ersten Reaktionen waren sehr positiv und es scheint große Bereitschaft zu geben schnell umzusteigen und die Plug-ins zu portieren.

Über die ganze Arbeit werde ich mit dem Problem der Subjektivität meiner Ergebnisse und Erkenntnisse zu kämpfen haben. Durch die langjährige Arbeit mit ESV und den ESV-Benutzern scheinen für mich viele Dinge selbstverständlich, die vielleicht nicht offensichtlich und möglicherweise sogar falsch sind.

Hier möchte ich, soweit im Rahmen einer Bachelorarbeit möglich, meine „Gefühle“ mit Beispielen aus dem Quellcode oder meiner Erfahrung belegen und durch die Nutzung eher objektiver Softwaremetriken ergänzen. Die Grundsatzentscheidung, dass ein Reengineering notwendig ist, wurde zwar bereits getroffen, doch möchte ich bis zum Ende die Erfolgsbewertung offen halten.

Mit dem Wissen, was erreicht werden soll, wie der Weg dorthin ist und welche Probleme mich auf dem Weg erwarten, kann ich mich nun an die Durchführung des Reengineerings machen.

2. Durchführung

2.1. Evaluation von ESV

2.1.1. Die Geschichte

Damit der Leser einen Überblick über die Situation bekommt, wie sie sich mir zu Beginn meiner Bachelorarbeit darstellt, möchte ich hier eine kurze Geschichte der vorliegenden Software anführen.

Im März 2006 beschlossen Sven Seeberg und ich, die Gelegenheit zu nutzen und für unsere Schule eine eigene Mensabestellsoftware zu schreiben, um unnötige Ausgaben im Rahmen der Mensaeinrichtung zu vermeiden. In Zusammenarbeit mit der zukünftigen Küchenchefin und unserem Schulleiter erhoben wir die Anforderungen an unser System. Das verlief weder systematisch noch existieren heute davon noch irgendwelche Dokumente. Nach wenigen Wochen Entwicklungszeit begann im Juni 2006 die Essensausgabe mit unserem System.

Schnell stellte sich heraus, dass immer weitere Anforderungen Änderungen an unserem System notwendig machen. Außerdem beschlossen wir zu diesem Zeitpunkt, anderen Schulen ebenfalls unsere Software anzubieten.

Daher entstand nach etwa einem Jahr eine neue Version des Mensasystems, das nun aber als elektronische Schulverwaltung flexibler aufgebaut wurde, um die Erweiterbarkeit zu verbessern und individuelle Einstellungen für verschiedene Schulen zu erlauben. Die Anforderungen hierfür wurden zum Teil per Mail vom Küchenpersonal der Mensa, sowie aus unserer Erfahrung als Administratoren des Systems, erhoben.

Diese Version nannten wir ESV (elektronisches Schulverwaltungssystem) und das Mensamodul, als ein optionaler Bestandteil des Systems heißt AMSS (automatisiertes Mensasystem für Schulen).

In den letzten drei Jahren wurde das System regelmäßig weiter entwickelt, um den Anforderungen neuer Schulen gerecht zu werden und neue Aufgaben zu übernehmen.

Inzwischen ist das System an etwa 10 Schulen, bzw. Internaten und auch Kindergärten, im Einsatz und zum Mensamodul kamen noch sechs weitere Module hinzu. (siehe [2])

2.1.2. Die Struktur

ESV ist logisch aufgebaut als ein Basismodul, welches die Benutzer-, Gruppen- und Rechteverwaltung, Systemeinstellungen, Backups und den Zahlungsverkehr mit Lastschriften bereitstellt. Auf diese Basis greifen dann Module zu, die für spezielle Einsatzgebiete entwickelt sind. Zum Beispiel das Mensamodul AMSS.

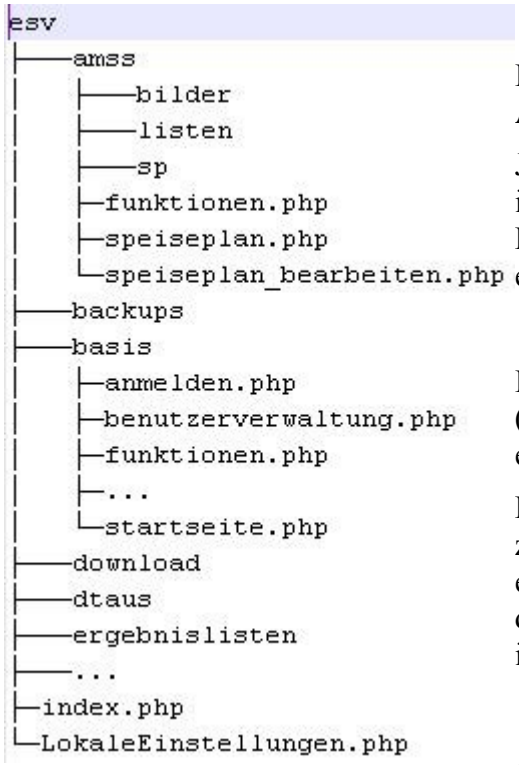

Abbildung 2: Ordnerstruktur von ESV

In ESV hat jedes Modul einen eigenen Ordner (BASIS, AMSS, …) in dem alle modulspezifischen Dateien liegen.

Jedes Modul kann außerdem eine funktionen.php benutzen, in der häufig verwendete PHP-Funktionen gespeichert sind. Diese Datei wird dann automatisch vom Basissystem eingelesen.

Innerhalb der Module ist jede Seite, die eine Aufgabe erfüllt (z. B. den Speiseplan bearbeiten), in einer Datei mit entsprechendem Namen realisiert.

Das hängt eng mit dem Rechte- und Navigationssystem zusammen, welches dateiweise arbeitet. Jeder Datei wird ein Menüeintrag, mit einem Recht zugewiesen. Benutzer, die das Recht dieses Menüeintrags haben, können diesen in ihrer Navigationsleiste sehen und die Seite aufrufen.

Außerdem gibt es noch ein paar andere Ordner für Backups oder zum Speichern von Dateien zum Herunterladen.

2.1.3. Softwarequalität

2.1.3.1. Code Smells

Bevor ich eine quantitative Untersuchung der Softwarequalität in ESV durchführe, möchte ich mit Hilfe sogenannter Code Smells, das sind Quelltextkonstrukte, die auf ein tiefer liegendes Problem hinweisen, mögliche Probleme identifizieren. Beim Entwurf des neuen Systems werde ich dann versuchen, diese Probleme durch bewährte Entwurfsmuster zu vermeiden.

Was man als Code Smell bezeichnet, ist dabei unterschiedlich. Ich möchte hier nur drei Smells benutzen, die ich in jeder Taxonomie [3] so oder so ähnlich vorgefunden habe und die ich vermutlich auch in ESV finden werde:

„Duplicate code“, „Large method“ und „Duplicated method“

2.1.3.1.1. Duplicate Code

Dieser Code Smell beschreibt den Zustand, dass in weiten Teilen ähnlicher oder komplett gleicher Quellcode an verschiedenen Stellen zum Einsatz kommt. Dies ist nicht wünschenswert, da so Änderungen an dieser Funktionalität an mehreren Stellen nachgetragen werden müssen.

Als Beispiel für doppelten Quellcode können in besonderem Ausmaß die Dateien des Mensamoduls „speiseplan.php“ und „speiseplan_bearbeiten.php“ dienen.

```
46  for($i=0;$i<$vorbestellwochen;)  // für jede Woche den Speiseplan anzeigen
47  {
48    ...
49    if($spaltenanzahl==0 AND $gibt_es_essen_in_zukunft){$vorbestellwochen++; $i++;}
50    elseif($spaltenanzahl==0 AND !$gibt_es_essen_in_zukunft){$i=$vorbestellwochen;}
51    else
52    {
53      ...
54      echo "<tr><td colspan=$spaltenanzahl><center><button type='submit' onClick='change_h
55      ...
56    }
57    $thisweek = plus_one_week($thisweek);
58  }
```

Abbildung 3: speiseplan.php (gekürzt)

```
56  for($i=0;$i<$hohezahl;)  // für jede Woche den Speiseplan anzeigen
57  {
58    ...
59    if($spaltenanzahl==0 AND $gibt_es_essen_in_zukunft){$i++;}
60    elseif($spaltenanzahl==0 AND !$gibt_es_essen_in_zukunft){$hohezahl = $i;}
61    else
62    {
63      ...
64      echo "<tr><td colspan=$spaltenanzahl><center><button type='submit' onClick='change_
65      ...
66    }
67    $thisweek = plus_one_week($thisweek);
68  }
```

Abbildung 4: speiseplan_bearbeiten.php (gekürzt)

Der hier gezeigte Quellcode dient dazu, den Speiseplan in einer Tabelle anzuzeigen. Die äußerste Schleife wird für jede Woche durchlaufen und die Anzahl der Wochen unterscheidet sich zwischen den beiden Dateien. Sonst sind beide an diesen gekürzten Stellen identisch.

2.1.3.1.2. Large Method

In ESV wird relativ wenig mit Funktionen gearbeitet. Häufiger genutzte Dinge werden aber doch in den jeweiligen funktionen.php-Dateien der Module definiert. Hier eine Auflistung der größten Funktionen aus den Modulen BASIS und AMSS.

Funktionsname	**Länge**
zeige_bestellungen($datum, $drucken="nein")	164 Zeilen
mach_bestellungen_liste($datum, $order)	142 Zeilen
createImageWithStats($width, $height, $ueberschrift, $balkenwerte, $balkennamen, $pfad, $bub=false, $wertbezeichnung="")	116 Zeilen
rechnung_anzeigen_amss($vorgangsnummer)	114 Zeilen
menues_bestellen($bestellungen,$bestellungsuhrzeiten=0,$kleinsteid,$groesteid, $ort_der_ausgabe)	100 Zeilen
tagesmenue($wochenmenue, $ausgabeort, $aktion="bestellen")	83 Zeilen
csv_datei_lesen($file_name, & $data, $required_cols, $valid_cols, & $message)	70 Zeilen
menues_zum_bearbeiten_anzeigen($anzeige_array,$kleinsteid,$groesteid)	69 Zeilen

Tabelle 1: Liste der längsten Funktionen ohne Kommentarzeilen

Die Längenangaben sind ohne Kommentare, allerdings wird bei mehrzeiligen Schleifen jede Klammer „{“ und „}“ in eine Zeile geschrieben. Die Formatierung ist also nicht sehr kompakt. Trotzdem sind diese Funktionen deutlich zu lang.

2.1.3.1.3. Duplicate Method

Auf der Suche nach doppelten Funktionen bin ich nur begrenzt fündig geworden. Es gibt eine Funktion, die die Farbe zu einer Kategorie liefert, während es eine allgemeinere Funktion gibt, die jedes Attribut der Kategorie auslesen kann.

Wie bei Duplicate Code bereits gezeigt, besteht das Softwarequalitätsproblem in ESV eher in zu wenigen Funktionen und dadurch zu vielen Codedopplungen.

2.1.3.2. Datenblatt Softwarequalität

2.1.3.2.1. LOC + Kommentare

Die Anzahl der Quellcode- und Kommentarzeilen habe ich mit zwei verschiedenen Werkzeugen gemessen. SLOCCount [4] liefert nur Quellcodezeilen ohne Leer- und Kommentarzeilen. Cloc [5] berechnet auch die Anzahl der Kommentarzeilen, was für meine Metrik, die ich hier erfassen möchte, wichtig ist.

Die Anzahl der Codezeilen sagt nicht viel aus und lässt sich über verschiedene Darstellungskonventionen und Programmiersprachen kaum sinnvoll vergleichen.

Was aber ein Indiz für gute oder schlechte Dokumentation im Programm sein kann, ist der Anteil an Kommentarzeilen im Quellcode.

Diesen möchte ich bestimmen.

Objekt	**Lines of Code ohne Kommentare**
ESV gesamt	24.663
BASIS	5572
AMSS	3375
BASIS funktionen.php	2254
AMSS funktionen.php	1626

Tabelle 2: Source Lines of Code (SLOC): gemessen mit SLOCCount entwickelt von David A. Wheeler

Objekt	**Lines of Code (PHP)**	**Kommentarzeilen**	**Kommentaranteil**
ESV gesamt	23089	1583	6,4%
BASIS	4991	1005	16,8%
AMSS	3416	77	2,2%
BASIS funktionen.php	1527	838	35,4%
AMSS funktionen.php	1628	17	1%

Tabelle 3: Lines of Code gemessen mit cloc entwickelt von Al Danial Copyright (c) 2006-2009, Northrop Grumman Corporation / Information Technology / IT Solutions

Das Ergebnis muss nun noch gedeutet werden.

Die Anzahl der Zeilen stimmt in beiden Berechnungen in etwa überein, was diese Zahlen vertrauenswürdig macht. Der Kommentaranteil in der funktionen.php des Basismoduls ist jedoch vollkommen unrealistisch. Auch durch eine visuelle Inspektion der Datei kann ich nicht erkennen, wie die große Anzahl Kommentarzeilen zustande kommen soll. Allerdings möchte ich aufgrund der Länge der Datei (2556 Zeilen mit Leerzeilen und Kommentaren) keine eigene Zählung durchführen.

Durch diesen gravierenden Messfehler wird auch das Ergebnis für das Basismodul und schließlich das Gesamtergebnis von ESV verzerrt.

Zwar wurde das AMSS-Modul hauptsächlich von mir entwickelt, während Sven Seeberg sich um das Basismodul gekümmert hat, doch dürfte das AMSS-Ergebnis viel näher an der Realität für das gesamte Projekt liegen.

2.1.3.2.2. Laufzeit

Laufzeit ist eine weitere gut erfassbare Softwaremetrik, die ich benutzen möchte. Besonders interessant daran ist für mich, dass Ruby on Rails nachgesagt wird [6], sehr langsam zu sein. Das soll an der Berechnung eines deutlich größeren Frameworks als in ESV realisiert liegen.

Ich messe auf dem gleichen Server, auf welchem später auch die Ruby on Rails Anwendung laufen wird ohne sonstige Last. Jede Aufgabe wird dabei 5-mal ausgeführt und der Durchschnitt hier als Zeit angegeben.

Startseite (nicht angemeldet)	0,14259 s
Anmelden (Anzeigen der Startseite nach Anmeldung)	0,26527 s
Benutzer anlegen	0,24167 s
Menü eintragen	0,21666 s
Menü bestellen	0,22180 s

Tabelle 4: Liste mit Aufgaben und der durchschnittlichen Ausführungsgeschwindigkeit

2.1.3.2.3. Testabdeckung

Die Testabdeckung in ESV beträgt 0%. Es gibt keinerlei Tests.

2.1.4. Abstraktion zum Modell

Das Klassendiagramm [7], [8] für das Basismodul habe ich aus der Analyse der Datenbanken erstellt. Die Operationen (siehe Diagramm im Anhang) sind der „funktionen.php“-Datei entnommen und werden hier aus Gründen der Übersichtlichkeit nicht angezeigt.

Die meisten Operationen sind aber sowieso in den einzelnen Dateien als Script und nur wenige als Funktionen realisiert.

Zur Übersicht habe ich die besonders stark vernetzten Klassen und deren Assoziationen mit einer eigenen Farbe versehen.

Alle von mir hier verwendeten Klassendiagramme werden mit ArgoUML 0.30 [9] erstellt. Dieses Werkzeug ist Open Source und steht kostenfrei zur Verfügung.

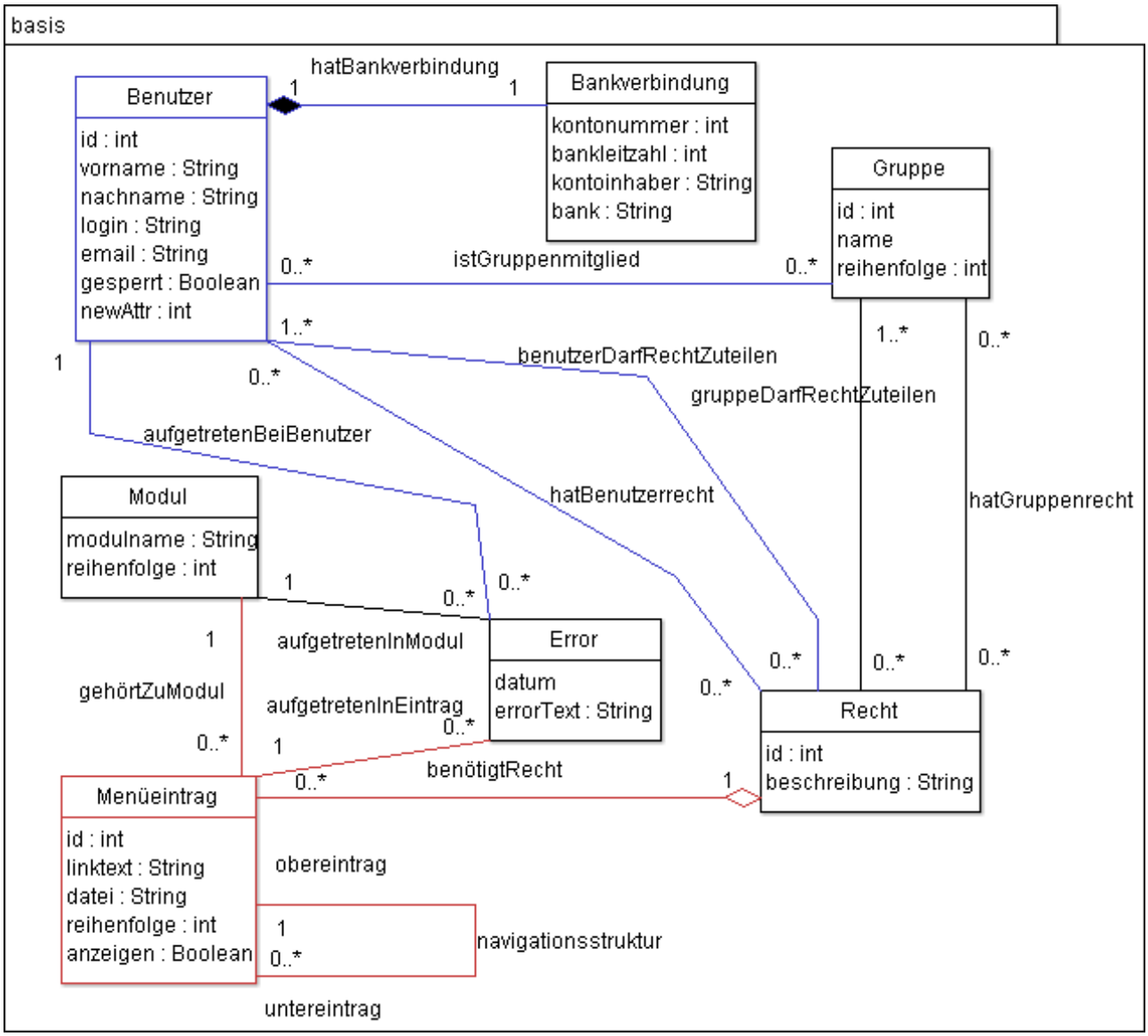

Abbildung 5: Klassendiagramm BASIS-Modul ohne Operationen

Das Klassendiagramm für das AMSS-Modul wurde auf die gleiche Weise erstellt.

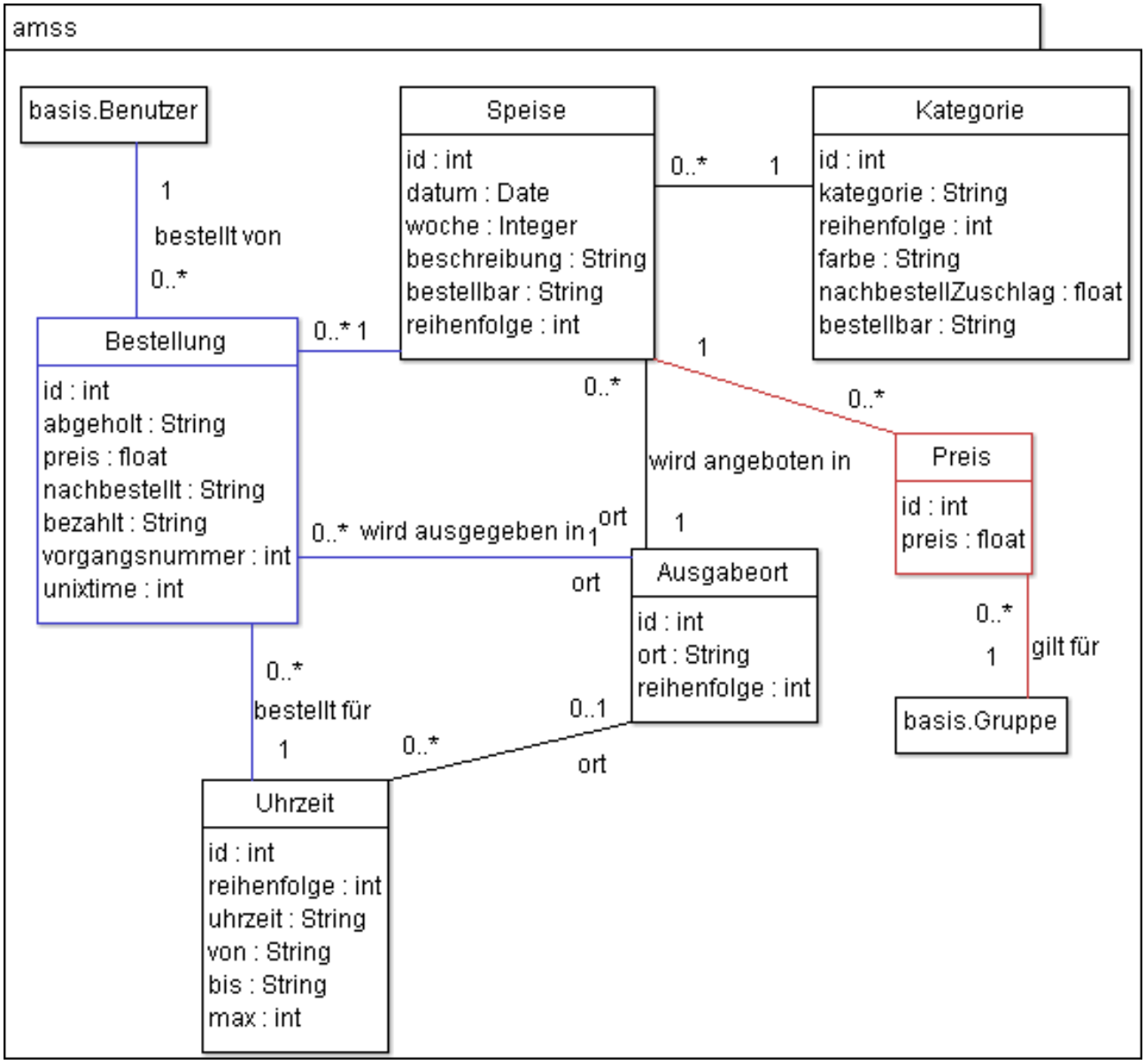

Abbildung 6: Klassendiagramm AMSS-Modul ohne Operationen

Die Schwierigkeit, diese Klassendiagramme zu erstellen, bestand darin, nicht objektorientierten Quellcode in Objekte und Klassen zu übertragen.

2.1.5. Funktionsumfang

Den Funktionsumfang des Schulverwaltungssystems samt der Mensaverwaltung dokumentiere ich mittels Use Cases. Dabei werden nur die Stellen von mir analysiert, welche auch in der neuen Software, im Rahmen der Bachelorarbeit realisiert werden sollen.

Diese Use Cases sind zukünftig Teil der Systemdokumentation und verbessern so die Softwarequalität im Bereich „externe Dokumentation“. Auf Detail-Use-Cases verzichte ich dabei, da diese schwerlich ohne Definition der internen Funktionsweise auskommen, welche ich erst im Planungsschritt für die neue Software festlegen will.

2.1.5.1. Beschreibung der Rollen

In den Use Cases werden folgende Rollen verwendet:

- **Benutzer:** ist jeder Besucher der Webanwendung, der sich angemeldet hat
- **Mensagast:** Benutzer, der am Mittagessen teilnehmen möchte und Bestellungen aufgeben kann
- **Caterer:** Benutzer, der den Speiseplan verwaltet und Bestellungen einsieht; liefert die Essen in die Mensa und wird dafür bezahlt: meist ein Angestellter des Essenslieferanten.
- **Verwalter/Mensaverwaltung:** Benutzer, der den täglichen Betrieb im Mensasystem verwaltet; bestellt für verhinderte Benutzer, rechnet ab und kann das Mensasystem konfigurieren: oft ein Mitarbeiter/eine Mitarbeiterin der Schulverwaltung.
- **Administrator:** Hat die Aufgabe längerfristige Änderungen vor allem im Schulverwaltungssystem vorzunehmen. Dazu gehört die Benutzer-, Gruppen- und Rechteverwaltung und die grundlegenden Einstellungen im System: ist entweder ebenfalls eine Schulverwaltungsmitarbeiterin, der IT-Zuständige der Schule oder ein Servicemitarbeiter des Systemanbieters.

2.1.5.2. Projektüberblick

Für die Use Cases habe ich eine Systematik angewendet, die Herr Professor Prechelt in der Vorlesung Softwaretechnik [10] an der Freien Universität Berlin vorgestellt hat. Diese verwendet keinen reinen Fließtext sondern eine klare Struktur um sicherzustellen, dass jeder Use Case die notwendigen Inhalte aufgreift und um die Lesbarkeit deutlich zu vereinfachen.

Als Beispiel für die insgesamt 19 Use Cases soll hier der Projektüberblick dienen. Die anderen befinden sich im Anhang.

Vor die Entscheidung gestellt, welcher Ablauf, der mit einem Ziel zu definieren ist, als zentral für das Projekt angesehen werden kann, habe ich an die Anfänge der Software gedacht. Auch wenn immer weitere Anforderungen hinzugekommen sind, war und ist das Kernstück des Mensasystems die Information des Caterers über die Anzahl der Essen, welche vor Ort gekocht oder angeliefert werden sollen. Diese Information setzt die Möglichkeit der Bestellung für potentielle Mensagäste voraus. Aus Schulverwaltungssicht bot es sich an, die personal-aufwändige Bezahlung ebenfalls elektronisch zu erledigen. Deshalb habe ich als Projektziel „Mensabestellungen erfassen und abrechnen“ definiert.

Der Hauptakteur, welcher von dieser Information am meisten profitiert ist der Caterer, weshalb der Use Case aus seiner Perspektive beschrieben ist.

Im Haupt-Erfolgsszenario unterstrichene Textstellen verweisen auf weitere spezifischere Use Cases, welche, sofern der Text nicht dem Use Case-Namen entspricht, in Klammern dahinter vermerkt sind.

Daraus entstand der folgende Use Case:

Text 1: Use Case - Projektüberblick

Mensabestellungen erfassen und abrechnen

Hauptakteur: Caterer
Anwendungsbereich: Mensasystem
Niveau: Projektüberblick
Beteiligte und Interessen:

Caterer	Anzahl und Zusammensetzung der Bestellungen erfahren, um diese in der Mensa auszugeben
Mensagast	Bestellung aufgeben und Speise bekommen
Mensaverwaltung	Bezahlung der Catererrechnung anhand der Mensagastbestellungen durchführen

Voraussetzung: Benutzer sind im System angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall: Caterer erfährt wie viele Essen bestellt wurden und wird bezahlt
Mensagast kann Essen bestellen und erhält dieses am Ausgabetag
Haupt-Erfolgsszenario:

1. Caterer trägt den Speiseplan ein (Speiseplan einstellen)
2. Mensagast bestellt (Essen bestellen) für ihn eingestellte Speisen
3. Caterer liest Anzahl der Bestellungen (Bestellungen einsehen) im System
4. Caterer gibt bestellte Essen aus
5. Mensaverwaltung rechnet Ausgabetage ab (Essen abrechnen)
6. Mensaverwaltung bezahlt den Caterer

Erweiterungen: ---

Damit ist die funktionale Beschreibung des Systems, die natürlich noch deutlich ausführlicher ausfallen könnte, abgeschlossen und das zu reengineerende System in groben Zügen beschrieben.

Außerdem wurde die Softwarequalität des vorhandenen Systems evaluiert, um diese nachher mit dem neu entstehenden Produkt zu vergleichen. Besonders schwierig war hierbei, dass es weder eine Dokumentation gab noch eine Objektorientierung angewandt wurde und so die meisten Softwaremetriken nicht praktikabel waren.

Auf Grundlage der bei der Evaluierung entstandenen Artefakte soll nun im nächsten Schritt das neue System entworfen werden.

2.2. Planung der neuen Software

Zuerst möchte ich einige Dinge aufführen, die im Vergleich zum vorhandenen System verändert werden sollen. Dann müssen diese Änderungen beim Entwurf des neuen Systems berücksichtigt werden.

2.2.1. Systemveränderungen

2.2.1.1. Ausweisnummer

Im bisherigen Schulverwaltungssystem ist die Ausweisnummer des Benutzers, welche normalerweise auf einem Ausweis als Barcode dargestellt wird, als interne Identifikationsnummer für den Benutzer verwendet worden (siehe Klassendiagramm in 2.1.4). Die Ausweisnummer wird also in allen Beziehungen zu diesem Benutzer referenziert und ist so in sehr vielen Datenbanktabellen anzutreffen.

Dadurch wird es enorm aufwändig diese Ausweisnummer bei Bedarf zu verändern, zum Beispiel wenn ein Ausweis verloren wurde. Insgesamt verletzt dieses Vorgehen die Devise interne Strukturen durch Kapselung zu verbergen.

Wird diese Änderung im Systementwurf vorgesehen, ist der Aufwand minimal bzw. praktisch nicht vorhanden, während die Änderung im vorhandenen System aufwändig ist.

2.2.1.2. Rechteverwaltung

Das Rechtesystem in ESV ist sehr flexibel angelegt. Die Verteilung von Rechten kann an Gruppen und Benutzer vorgenommen werden, außerdem kann für jeden Link in der Navigationsleiste ein Recht „im laufenden Betrieb“ verändert werden.

Es hat sich aber herausgestellt, dass es nur wenige verschiedene Interessengruppen gibt, die tatsächlich in der Rechteverwaltung berücksichtigt werden müssen. Bei den Use Cases aus der Anforderungsbeschreibung wird das durch die Rollen bereits deutlich.

Außerdem mussten wir feststellen, dass die meisten Schulen keine größeren Veränderungen am System vornehmen wollen und mit den von uns eingestellten Werten arbeiten.

Daher kann das Rechtesystem stärker im Quellcode verankert werden.

Andererseits ist die Konzentration der Rechteabfrage auf die Links in der Navigation ein Hindernis. Oft wäre es sinnvoller, gewisse Verwaltungsfunktionen unter dem gleichen Menüpunkt durchzuführen wie die Benutzung. Für den Speiseplan gibt es im Mensasystem momentan mehrere Links für den Speiseplan. Jeweils einen zum Bestellen, Eintragen und Bearbeiten.

Das neue Rechtesystem soll also verstärkt innerhalb einer Seite differenzieren können.

Zusammenfassend kann die Laufzeitflexibilität der Rechteverwaltung abnehmen, während die Flexibilität abhängig von der Rolle innerhalb einer Seite zunehmen sollte.

2.2.1.3. Zu bezahlende Artikel

Einer der wesentlichen Vorteile des elektronischen Bestellsystems im Vergleich zu anderen Lösungen ist für Schulen die Möglichkeit des bargeldlosen Bezahlens. Dadurch haben sich nach dem Mensasystem immer weitere Aufgaben, die eine Bezahlung erfordern, als weitere Module entwickelt.

Deshalb werde ich versuchen eine einheitliche Grundlage für bezahlbare Artikel und Dienstleistungen zu finden, die dann die Abrechnung und das Anzeigen der Rechnung auch mit verschiedenen Modulen einheitlich gestaltet.

2.2.1.4. Speiseplan

Der am meisten veränderte Bestandteil im Mensasystem ist der Speiseplan. Die Komplexität bzw. Flexibilität nahm mit der Zeit immer weiter zu. Das ist den Anwendungsarten der verschiedenen Schulen geschuldet.

Daher erhebe ich eine neue Anforderungsliste für den Speiseplan, bzw. einzelne Menüs/Speisen darin.

notwendig: (diese Dinge muss das System bieten)

- Verallgemeinerung einer Speise in eine Kategorie
  z .B. aus Spaghetti Bolognese mit Salatbeilage wird Hauptmenü
- Unterscheidung wann Speise abgeholt wird (Schichten)
  z. B. 11:45 oder 12:30 bzw. Schicht1-Schicht2
- verschiedene Preise für eine Speise
  z. B. 3,00€ für jeden außer für Gruppe Sozialfond (2,00€)
- Speisen die nicht vorbestellt werden können und nicht im Speiseplan erscheinen (neu)
  z. B. Dessert für das man sich vor Ort entscheiden kann oder nicht

zukünftig notwendig: (müssen gehen aber nicht im Rahmen der Bachelorarbeit)

- Abonnement einer Speise für einen gewissen Zeitraum – Ausnahmen möglich (neu)
  z. B. jeden Dienstag Hauptmenü um 11:45 bis zum 30.06.2010 für den 22. Juni möchte der Benutzer dann trotz Abo abbestellen
- Katalog von Speisen aus dem gewählt werden kann (neu)
  z. B. Hauptmenü mit Beschreibung Spaghetti Carbonara hat Katalognummer 100
- Bewertung von Speisen (neu)
  z. B. Spaghetti Carbonara am Dienstag 22. Juni 2010 mit 4 von 5 Sternen

wünschenswert: (diese Dinge wären nett und sollten vielleicht zumindest zukünftig möglich sein)

- Unterscheidung des Angebots für Frühstück, Pause, Mittagessen, Abendessen (neu)
  z. B. Müsli beim Frühstück und Hauptmenü beim Mittagessen
- dauerhaft vorhandene Artikel (neu)
  z. B. Müsliriegel, der immer angeboten wird aber nicht vorbestellt werden muss
- Menübestellungen (neu)
  z. B. eine Vorspeise + eine von mehreren Hauptspeisen + Nachtisch = 4,70 €

Anforderungen, die mit (neu) versehen sind, sind vom aktuellen System nicht gewährleistet.

Eine Funktion, die nicht in das neue System übernommen werden soll ist die Unterscheidung, an welchem Ort die Speise ausgegeben wird. Das kommt daher, dass keiner unserer Kunden diese Funktion benutzt.

Notwendige Anforderungen sollen im Rahmen der Bachelorarbeit zumindest grundlegend implementiert werden und müssen im Modell Berücksichtigung finden.

Zukünftig notwendige Anforderungen müssen im Modell möglich sein. Wie diese Anforderungen umgesetzt werden können, sollte zumindest in der Idee klar sein.

Die wünschenswerten Anforderungen sollen bei der Modellierung berücksichtigt werden, um ihre Umsetzung wenn möglich zu vereinfachen und einzuplanen, aber diese zumindest nicht unnötig zu erschweren.

Damit sind die wichtigsten Veränderungen aufgezeigt. Natürlich dient das neue System dazu, weitere Veränderungen vorzunehmen. Um jedoch eine gewisse Vergleichbarkeit der Softwaresysteme zu erhalten und im Rahmen des Reengineerings zu bleiben, sollen diese Schritte erst später folgen.

2.2.2. Modellierung

Die Wahl eines Architekturmusters für die Webanwendung fällt nicht schwer, da Ruby on Rails als Framework darauf ausgelegt ist jede Anwendung mit dem Model-View-Controller (MVC) Muster zu realisieren.

So wird eine gute Trennung der Daten (im Model), der Geschäftslogik (meist im Controller) und der Anzeige (im View) erreicht.

2.2.2.1. Klassendiagramme

Um die Ähnlichkeit mit den Klassendiagrammen aus der Evaluierung des alten Systems zu erhalten, modelliere ich die Datenebene und verzichte auf die Controllerklassen. Um im Rahmen der 5 - 15 Klassen pro Diagramm, welche ein allgemein akzeptierter Richtwert sind, zu bleiben, ist das auch unerlässlich.

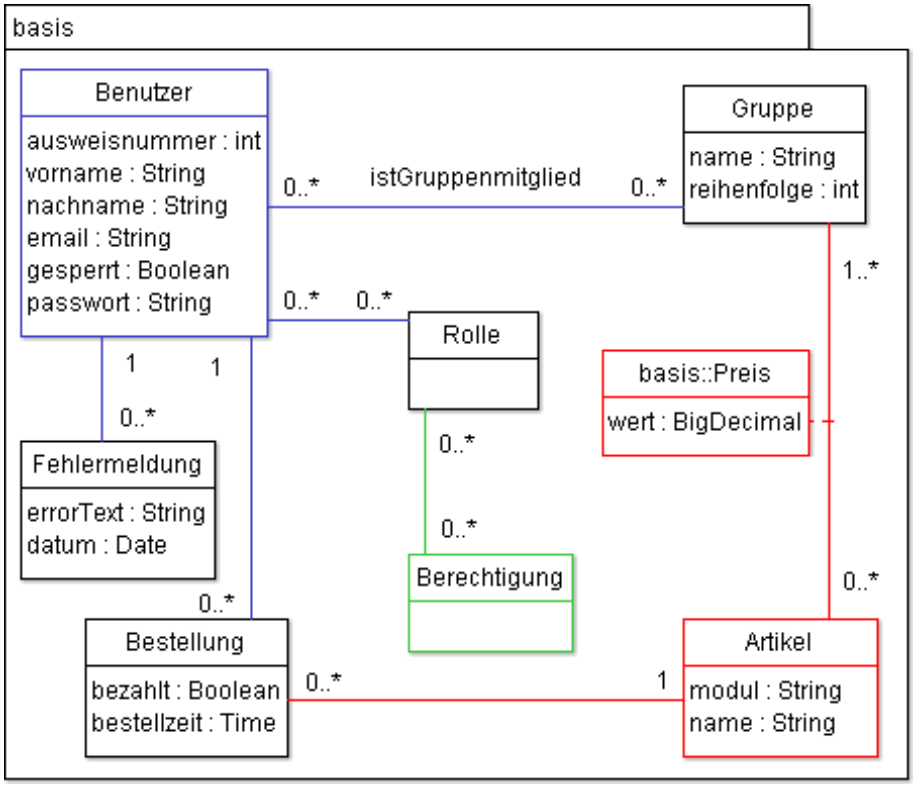

Abbildung 7: Klassendiagramm des neuen BASIS-Moduls ohne Operationen

Im Basissystem gab es einige Veränderungen, welche sich in der neuen Planung auswirken. Das Rechtesystem wird verändert und arbeitet nun mit Rollen, welche Benutzer mit Berechtigungen verknüpfen. Diese Berechtigungen können dann im Quellcode abgefragt werden.

Außerdem hat der Benutzer eine Ausweisnummer bekommen. Die ID Felder, welche im alten System von mir modelliert wurden, fehlen hier, da Ruby on Rails jeder Datenbanktabelle und damit jedem Objekt eine ID gibt.

Auffallend ist auch, dass drei komplett neue Klassen auftreten. Preis als Assoziationsklasse, sowie Artikel und Bestellung sollen Grundklassen für alle Module bereitstellen, um den Zahlungsverkehr im Basissystem zu bündeln. Ein Artikel ist ein Objekt mit einem Preis, eine Bestellung ist das Produkt der Entscheidung eines Benutzers, einen Artikel zu erwerben.

Alternativ habe ich zeitweise überlegt, eine Klasse Rechnung einzuführen. Die Module hätten dann eigenständig Rechnungen erstellen können, deren Werte sie selbst festlegen müssten und das

Basismodul hätte nur anhand dieser Rechnungen die Abrechnung erstellt.

Dann wäre die Verknüpfung des Preises zur Gruppe usw. aber in jedem Modul selbst zu regeln, weshalb ich mich gegen diese Variante entschieden habe.

Die Bankdaten wurden nicht vergessen, sollen aber im Rahmen der Bachelorarbeit nicht realisiert werden, da Sicherheitsaspekte kein zentraler Bestandteil dieser Arbeit sind.

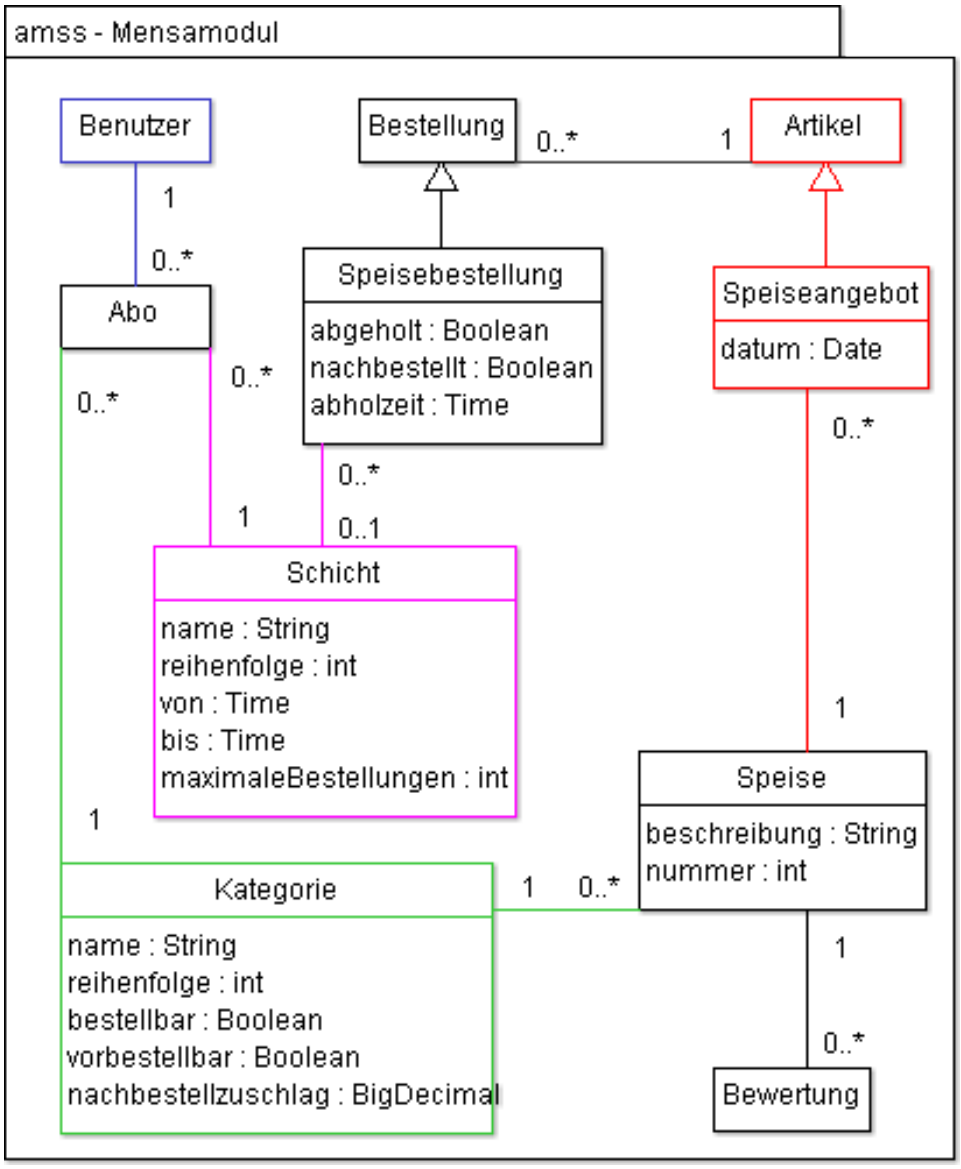

Abbildung 8: Klassendiagramm des neuen amss-Moduls ohne Operationen

Das Mensasystem habe ich mit Hilfe der drei Basisklassen Benutzer, Bestellung und Artikel modelliert. Anhand der neuen Anforderungen für den Speiseplan zeige ich, wie diese sich auf die Modellierung ausgewirkt haben.

notwendig: (diese Dinge muss das System bieten)

- Speisen die nicht vorbestellt werden können und nicht im Speiseplan erscheinen (neu)

Die Klasse Kategorie hat das Attribut vorbestellbar bekommen. Was vorbestellbar ist wird im Speiseplan angezeigt und was bestellbar ist, kann unter Umständen bei der Ausgabe nachbestellt werden, auch wenn es nicht im Speiseplan angezeigt wurde.

zukünftig notwendig: (müssen gehen aber nicht im Rahmen der Bachelorarbeit)

- Abonnement einer Speise für einen gewissen Zeitraum – Ausnahmen möglich (neu)
  Hier könnte es eine Klasse Abo geben, die einer Kategorie und einer Schicht zugeordnet ist und außerdem einen Wochentag (nicht modelliert) enthalten muss. So kann ein Benutzer für einen gewissen Wochentag das Speiseangebot einer Kategorie für eine Schicht bestellen. Das wird aber nicht im Rahmen der Bachelorarbeit implementiert.
- Katalog von Speisen aus dem gewählt werden kann (neu)
  Die Klasse Speise existiert nun unabhängig vom Speiseangebot. Eine Speise kann mehrmals zu verschiedenen Daten im Speiseplan erscheinen ohne jedes mal neu beschrieben werden zu müssen. Angesprochen werden kann die Speise über eine Nummer.
- Bewertung von Speisen (neu)
  Durch den Speisenkatalog wird es einfach, eine Speise unabhängig von einem Tag zu bewerten. Dies ist als Bewertungsklasse modelliert, wird jedoch während der Bachelorarbeit nicht umgesetzt und kann nach einer ersten Recherche auch mit diversen Plug-ins für Ruby on Rails direkt im Modell der Speise realisiert werden.

wünschenswert: (diese Dinge wären nett und sollten vielleicht zumindest zukünftig möglich sein)

- Unterscheidung des Angebots für Frühstück, Pause, Mittagessen, Abendessen (neu)
  Diese Idee habe ich nicht modelliert, allerdings in der Planung auf dem Blatt berücksichtigt. Vermutlich wird das Speiseangebot mit einer Klasse Tageszeit verknüpft, um zu entscheiden ob es für das Frühstück oder Mittagessen usw. angeboten wird. Die Schichten würden dann ebenfalls von der Tageszeit abhängen.
- dauerhaft vorhandene Artikel (neu)
  Hierfür habe ich noch keine besonders einleuchtende Lösung gefunden.
- Menübestellungen (neu)
  Dazu müsste das Speiseangebot verändert werden, um ein Angebot mit mehreren Speisen zu verknüpfen oder eine Bestellung auf mehrere Speiseangebote zu beziehen. Außerdem bräuchte man Regeln, wie diese Menüs zusammengestellt werden dürfen.

Der neue Entwurf ist meiner Meinung nach etwas besser als die Evaluationsdiagramme des alten Systems, da einige Dinge nun nicht mehr Klassen sind (Menüeinträge, Module) und diverse Dopplungen nicht mehr vorkommen. So war früher der Wert, ob etwas bestellbar ist, in der Kategorie und im Speiseangebot vermerkt. Auch den Preis vermerkt das neue System bisher nicht mehr in der Bestellung. Allerdings muss hier überlegt werden, wie man damit umgeht, wenn ein Benutzer nach seiner Bestellung aus der Gruppe entfernt wird, deren Preis er bezahlt. Denn ein Benutzer muss sich auf den Preis, zu dem er bestellt, verlassen können.

Dieses Problem werde ich im Auge behalten und bei der Implementierung nach einer möglichst guten Lösung dafür suchen. (Lösung siehe Kapitel 2.3.7.2)

2.2.2.2. Wahl der Technik

In den Bereich des Softwareentwurfs gehört auch die Technikentscheidung. Dass das System mit Ruby on Rails implementiert werden soll, steht schon fest und muss daher nicht extra begründet werden. Für mich waren dabei folgende Argumente ausschlaggebend:

1. Ruby on Rails verspricht schnelle Entwicklungsfortschritte
2. Ruby on Rails ist auf Testbarkeit ausgelegt
3. eine aktive Community, die bei Problemen helfen kann
4. MVC-Struktur vorgegeben
5. viele Plug-ins, die alltägliche Aufgaben übernehmen

Gerade den letzten Punkt möchte ich natürlich sofort nutzen. Deshalb habe ich mich umgesehen, welche Plug-ins mir Programmierarbeit abnehmen könnten und die in der Community beliebtesten Plug-ins für die Aufgaben Authentifizierung und Autorisierung gesucht. Diese können mit wenig Aufwand in jedes Ruby on Rails Projekt eingebunden werden.

Meine Plug-inrecherche führte mich zur Webseite ruby-toolbox.com [11], auf der Plug-ins mit Links zum Quellcode in verschiedenen Kategorien verlinkt sind. Innerhalb einer Kategorie werden die Werkzeuge nach der Anzahl der Aktualisierungsverfolger und dem Zeitpunkt des letzten Updates geordnet.

So wird sichergestellt, dass die Plug-ins noch entwickelt und langfristig unterstützt werden.

Ich habe jeweils das beliebteste Plug-in ausgewählt und überprüft ob die Funktionalität meinen Anforderungen genügt.

2.2.2.2.1. Authentifizierung

Das Authentifizierungsplug-in soll sich um die Sessionverwaltung kümmern und die Anmeldung und Registrierung der Benutzer unterstützen. Hier gibt es momentan zwei sehr häufig verwendete Plug-ins: restful-authentication [12] und authlogic [13]. Restful-authentication galt bis vor kurzem als Standard für Authentifizierung wurde jedoch bereits seit mehr als 6 Monaten nicht mehr aktualisiert und wird in der Beliebtheit von authlogic überholt. Das ist ein häufiger zu beobachtendes Phänomen in der sehr kurzlebigen Ruby on Rails Community. Plug-ins werden regelmäßig durch verbesserte Nachfolger ersetzt, wobei es meist Anleitungen zur Migration vom bisherigen Standard zum Newcomer gibt.

Authlogic erfüllt meine Voraussetzungen, lässt sich sehr gut und mit weniger Quellcode als restful-authentication anpassen, und scheint zum neuen Standard zu werden. Deshalb entscheide ich mich dafür.

2.2.2.2.2. Autorisierung

Das Autorisierungsplug-in soll angemeldeten Benutzern Rechte zuweisen und die Möglichkeit bieten, diese Rechte für den Aufruf gewisser Funktionen und die Anzeige von Inhalten vorauszusetzen.

Das beliebteste Plug-in ist momentan declarative_authorization [14], das die Autorisierung nicht direkt in der Businesslogik sondern in einer eigenen Rechtekonfiguration vornimmt. Vorausgesetzt wird ein Benutzerobjekt, welches das Authentifizierungsplug-in bereitstellt.

2.2.2.2.3. Test-Frameworks

Da immer wieder bemängelt wird, dass die in Ruby on Rails integrierte Teststruktur zu langsam und etwas umständlich ist, habe ich mich auch nach Test-Frameworks umgesehen. Die beliebtesten sollen Behaviour Driven Development (Verhaltensgetriebene Entwicklung) ermöglichen. In meinen Augen ist der mit den derzeit gängigen Frameworks (cucumber [15], rspec [16]) erzeugte Testcode nicht sonderlich intuitiv zu lesen, auch wenn oder gerade weil, versucht wird, die Tests in möglichst natürlicher Sprache zu verfassen. Daher werde ich mich erst einmal mit den vorhandenen Testfähigkeiten zufrieden geben und bei Bedarf noch einmal nach einer geeigneten Lösung suchen.

Da ich testgetrieben entwickeln will, ist das Test-Framework natürlich ein wesentlicher Bestandteil der Entwicklungsumgebung.

2.3. Implementierung

Nachdem das neue System nun entworfen ist, kann die Umsetzung in Quellcode erfolgen. Dabei möchte ich zuerst die Implementierungsreihenfolge festlegen. Da ich alleine programmiere, ist es nicht notwendig, die parallele Entwicklung verschiedener Teile zu planen.

Dann erläutere ich das Prinzip des Test Driven Development, welches ich bei der Realisierung beachten will, um die Softwarequalität auf einem hohen Niveau zu halten.

Um erfolgreich zu entwickeln, braucht man auch eine Entwicklungsumgebung, die den Programmierer dabei unterstützt. Wie diese bei mir eingerichtet wird und welche Möglichkeiten für Ruby on Rails Entwicklung sich hier bieten, möchte ich ebenfalls kurz vorstellen.

Sobald die Entwicklungsumgebung eingerichtet ist, kann die Anwendung angelegt und die ersten Zeilen Quellcode, beziehungsweise bei mir natürlich Tests, geschrieben werden.

2.3.1. Planung der Reihenfolge

Zuerst möchte ich in enger Verbindung mit Authlogic [13], dem Authentifizierungsplug-in, mit der Benutzerverwaltung beginnen. Der Anmeldeprozess ist die Grundvoraussetzung für die Benutzung dieser Webanwendung, weshalb ich mit der Sessionverwaltung und Anmeldung starten will.

Dann sollen Gruppen und Rollen für die Benutzer verfügbar gemacht werden. Mit der Rollenverwaltung lässt sich dann das Autorisierungssystem einbinden.

Im Basissystem soll es mit den Klassen Bestellung, Artikel und Preis die Möglichkeit zur Abrechnung diverser in den Modulen erstellter Artikel geben. Bevor ich mich dem Mensasystem zuwende, möchte ich diese Klassen vorbereiten.

Das Mensasystem beginne ich mit den recht unabhängigen Kategorien, welche sich problemlos implementieren lassen sollten. Daran anschließen möchte ich mit dem neuen Konzept der Speisen in einer Art Katalog. Die Neuigkeit liegt hier nicht im technischen sondern im Vergleich zum vorherigen System. Sobald der Speisenkatalog besteht, kann ich mit der zentralen Funktionalität des Mensasystems beginnen. Das ist in meinen Augen der aktuelle Speiseplan.

Bevor nun die Bestellungen im Mensasystem implementiert werden können, sollten Schichten zur Auswahl stehen. Mit den Bestellungen und der Möglichkeit, sich die Anzahl dieser für einen Ausgabetag anzeigen zu können, ist das Mensasystem in den Grundzügen fertig.

Nun soll im Basissystem die Abrechnung von Bestellungen ermöglicht werden und eine Konfiguration der Anwendung, die sich je nach Schule unterscheiden kann, möglich sein.

Ein einfaches Layout und eine simple Menüführung sollen zur Demonstration des Systems hinzugefügt werden.

2.3.2. Bemerkung zum Test Driven Development

Unter Test Driven Development (TDD, oder im deutschen testgetriebene Entwicklung [17]) versteht man eine Programmiertechnik, bei der das gewünschte Verhalten des Systems mit Tests „überprüft“ wird, bevor der Anwendungscode für das System geschrieben wird.

Der TDD Prozess wird zyklisch angewandt und setzt sich aus folgenden Schritten zusammen:

1. **Ein neuer Test wird geschrieben**
   Dieser erwartet eine noch nicht vorhandene Funktionalität und wird auf Basis der Anforderungen erstellt.
2. **Alle Tests ausführen**
   Hierbei muss der neue Test fehlschlagen. Das stellt sicher, dass er nicht immer erfolgreich ist und die geprüfte Funktionalität nicht schon vorhanden ist.
3. **Funktionalität implementieren**
   Dabei soll so wenig Code geschrieben werden wie absolut notwendig ist, um den Test zu bestehen. Dadurch gibt es keinen ungetesteten Quellcode.
4. **Die Tests wieder ausführen**
   Nun sollte der neue Test erfolgreich sein und keine anderen Tests plötzlich nicht mehr erfolgreich durchlaufen werden.
5. **Quellcode Refaktorierung**
   Nun kann der Anwendungsquellcode verbessert werden und durch das Ausführen der Tests wird sichergestellt, dass die Funktionalität dabei erhalten bleibt.

Da Model, View und Controller in Rails getrennt entwickelt werden, gibt es dafür auch jeweils eigene Tests. Ich möchte die Anwendungslogik in den Controllern und die Datenlogik in den Models testgetrieben entwickeln.

Die Präsentation der Inhalte in den Views werde ich vorerst nicht mit Tests überprüfen, da ich hier nur eine einfache Darstellung implementierten möchte, die sich später unter Umständen noch deutlich ändert.

Da ich das Autorisierungssystem „declarative_authorization“ verwende, welches – wie der Name sagt – deklarativ arbeitet, d. h. die Rechtezuweisung in einer Konfigurationsdatei speichert, möchte ich diese Rechte nicht für jeden Controller überprüfen. Wenn das System prinzipiell funktioniert, ist es meiner Meinung nach deutlich unübersichtlicher, in jedem Controller Autorisierungstests zu machen, während die Rechte zentral in einer Datei definiert sind.

2.3.2.1. Vorteile

Von der testgetriebenen Entwicklung erhofft man sich diverse Vorteile [18]. Dadurch, dass Funktionalität nur implementiert wird, um Tests erfolgreich durchlaufen zu lassen, soll eine Testabdeckung von bis zu 100 % erreicht werden.

Außerdem erhofft man sich durch das Schreiben der Tests vor dem eigentlichen Quellcode, dass sich die Entwickler viel Gedanken über gut verwendbare Schnittstellen machen – woraus verständlicherer Quellcode resultiert.

Durch das häufige Ausführen der Tests in Schritt vier kann sichergestellt werden, dass neue

Features keine Defekte in anderen Teilen der Anwendung verursachen.

Zu guter Letzt sollen die Tests zur Refaktorierung ermutigen, da der Entwickler sofort Rückmeldung bekommt, ob er aus Versehen die Funktionalität verändert bzw. zerstört hat.

2.3.3. Entwicklungsumgebung

Meine Entwicklungsumgebung habe ich auf meinem Laptop eingerichtet, der mit Windows Vista, sowie Ubuntu 9.10 betrieben wird. Bei der Wahl einer IDE muss ich berücksichtigen, dass diese in Zukunft vielleicht auch von anderen Entwicklern eingesetzt werden soll.

Mein Partner beim Schulverwaltungssystem ist Sven Seeberg, der nur unter Linux arbeitet.

Der häufig verwendete Editor „Textmate“, welcher als Standard für Ruby on Rails Entwicklung gilt, ist für Mac konzipiert und kommt daher nicht in Frage. Eine ähnliche Umsetzung für Windows ist nicht kostenlos und ebenfalls nicht für Linux verfügbar.

„RailsWay“, das bisher einzige deutsche Magazin für Ruby on Rails, hat in seiner Ausgabe 1.2009 [19] diverse IDEs getestet und dabei „Netbeans“ [20] für besonders ausgereift und einsatzbereit befunden.

Da Netbeans dank Java auf allen wesentlichen Plattformen verfügbar ist und von mir bereits früher für andere Programmiersprache eingesetzt wurde, habe ich mich entschieden, Netbeans als IDE für die Entwicklung zu benutzen.

Diese Entscheidung hatte Konsequenzen für das Projekt. Netbeans ist noch nicht auf die Betaversion von Ruby on Rails 3 ausgerichtet, wodurch einige Funktionen wegen API-Änderungen nicht zur Verfügung standen.
Da im Internet auch im Zusammenhang mit den von mir gewünschten Modulen und der Betaversion von diversen noch nicht gelösten Problemen geschrieben wurde, entschied ich mich am Anfang meiner Implementationsphase dazu, doch das stabile Rails 2.3.5 zu verwenden.
So minimiere ich das Risiko und kann mich auf die wesentlichen Aufgaben meiner Arbeit konzentrieren.

Entgegen der gängigen Praxis, Railsprojekte in einem git-Repository zu betreiben, entschied ich mich dafür, Subversion zu benutzen. Die Vorteile von git [21], welches sich besonders für große Projekte, mit einer breiten Entwicklergemeinschaft und Entwickler ohne ständigen Internetzugang, eignet, waren für mich nicht bedeutend.

Ich entwickle also unter Ubuntu Linux mit der IDE Netbeans in Rails 2.3.5 und sichere meinen Quellcode in einem Subversion-Repository.

2.3.4. Erste Schritte

Als ersten Befehl führte ich „rails esv“ aus und erzeugte damit ein neues Railsprojekt namens esv.

Bevor es mit der Benutzerverwaltung losging, stellte ich fest, dass ich eine wichtige Tatsache nicht beachtet hatte. Rails setzt auf das Prinzip „Convention over Configuration“ [22]. Das bedeutet, dass viele Dinge nach einer Rails-Konvention gehandhabt werden und man sich viel

Konfigurationsaufwand spart, wenn man diesen Konventionen folgt. Ein Railsstandard ist, dass Models englische Namen haben und die dazugehörigen Datenbanktabellen dann mit dem Plural des Namens benannt werden. In meiner Planung bin ich von deutschen Klassennamen ausgegangen. Um die Railsvorteile wirklich nutzen zu können, entschied ich, den Quellcode und alle Modulbezeichnungen komplett englisch zu halten.

Dann machte ich mich direkt an die ersten Tests für das User-Model. Modelle werden in Rails mit Unit-Tests, Controller und Views mit Functional-Tests getestet. Bei der Benutzerverwaltung orientierte ich mich eng am Beispiel [23] für Authlogic und hatte noch Probleme, tatsächlich testgetrieben zu arbeiten.

Das Group-Model und der Group-Controller wurden zum ersten Mal komplett testgetrieben entwickelt. Bis auf die Verbindung des User-Models mit dem Group-Model gab es dabei aber auch keine wesentliche Veränderung im Vergleich zum User.

Danach ging es Model für Model und Controller für Controller weiter.

2.3.5. Nennenswerte Lösungen

Bei der Implementierung der Schichten im Model Mealtimes musste ich die Start- und Endzeit validieren, wofür ich in Rails selbst nur reguläre Ausdrücke benutzen könnte. Dadurch würde aber besonders die Bedingung, dass die Startzeit kleiner sein muss als die Endzeit sehr kompliziert.

Also entschied ich mich dafür, ein weiteres Plug-in zu installieren, welches Zeit und Datumsvalidatoren (siehe Glossar Validation) mit sich bringt. Mit dem Plug-in „validates_timeliness“ [24] wurde ich hier fündig.

Eine größere Herausforderung war auch die Umsetzung der Vererbung. Der Speiseplan, Menu, erbt von Article und die Speisebestellungen, MenuOrders, erben von den Bestellungen, Orders.

In Rails gibt es zwei Möglichkeiten dies zu implementieren. Die einfachere Variante nennt sich Single-Table-Inheritance (STI [25]) und speichert, wie der Name schon erahnen lässt, die Objekte aller vererbten Klassen in der gleichen Datenbanktabelle. Für jedes Objekt werden dann nur die Spalten gefüllt, die Attribute der Klasse abbilden. Dieses Konzept eignet sich vor allem, wenn die Klassen viele gemeinsame Attribute in der Elternklasse haben und sich sonst wenig unterscheiden.

Unweigerlich ist damit aber natürlich eine sehr enge Verknüpfung der Elternklasse mit den Kindklassen verbunden. Die zweite Möglichkeit der polymorphen Assoziation [26] erlaubt mir hier die verschiedenen Module besser zu trennen.

Dabei bekommt jede Klasse eine eigene Tabelle und ein eigenes Model. In der Tabelle der Elternklasse gibt es ausnahmsweise zwei Spalten um auf das Kindobjekt zu verweisen. Wie immer wird das Kindobjekt mit einer ID referenziert, aber da es diverse Kindklassen geben kann, wird in einer „kind_type“-Spalte der Klassenname des referenzierten Objekts vermerkt.

Die Besonderheit „polymorphic“ muss dann im Model explizit angegeben werden, wonach man die Assoziation in Rails wie gewohnt verwenden kann. Notwendige Casts zur passenden Klasse macht das System automatisch.

Beim Menu-Model musste ich zum ersten Mal eigene Validations schreiben. Zum einen möchte ich dort sicherstellen, dass von jeder Kategorie an einem Tag nur ein Menu eingetragen wird. Da ein

Menu keine Kategorie hat sondern nur das mit dem Objekt verbundene Meal, also die Speise, reichen hier die Railsvalidations nicht aus.

Außerdem stelle ich über die polymorphe Assoziation mit Article sicher, dass das Rechnungsdatum (billing_date) am Tag nach dem Ausgabetag des Menüs liegt. So können keine Bestellungen dafür abgerechnet werden, die unter Umständen noch vom Benutzer gelöscht werden.

2.3.6. Probleme während der Implementierungsphase

Während der Implementierung traten noch diverse Probleme auf. Diese lassen sich grob in zwei verschiedene Problemfelder aufteilen. Testen und testgetriebene Entwicklung und organisatorische Probleme während des Entwicklungsprozesses.

2.3.6.1. Testingprobleme

Test Driven Development hat mich vor einige Probleme gestellt, die ich hier kurz aufführen möchte.

Als erstes ist der Mangel an Tests für meine Views zu nennen, bei denen ich das Prinzip nicht angewandt habe. Dadurch musste ich deren Funktion manuell testen und stieß zum Teil erst später zufällig darauf, dass das Absenden eines Formulars gar nicht zum gewünschten Ergebnis führt.

Als zweites Problem hat sich bald der schrittweise Aufbau von Fixtures, das sind Testdatensätze für die Datenbank, herausgestellt. Durch das Hinzufügen neuer Attribute zum Model, sowie häufiger noch von neuen Verknüpfungen zu anderen Models, mussten diese regelmäßig angepasst werden.

Das führte häufig dazu, dass vorher erfolgreiche Tests dann fehlschlugen und diese wieder angepasst werden mussten. Das wurde so störend, dass ich ernsthaft in Erwägung zog, zum Ende der Implementierungsphase ein komplett neues Set von Fixtures für die Anwendung zu erstellen. Der damit verbundene Aufwand und die Notwendigkeit diverse Tests umzuschreiben haben mich dann davon abgehalten. Es gibt auch die Möglichkeit, Tests zu schreiben, die weniger von Fixtures abhängen, was von manchen Railsentwicklern als der bessere Weg angesehen wird [27].

Mehrmals bin ich auch mit dem Anfängerproblem konfrontiert worden, nicht zu wissen, wie etwas in Rails normalerweise funktioniert. Bei den Controllern war die Frage, in welchem Format Parameter übergeben werden.

Zum Beispiel, ob die ID zum Bearbeiten oder Löschen einer Speise mit params[:id] oder params[:meal][:id] übergeben wird. Solche Dinge musste ich dann mit einer einfachen View ausprobieren und die resultierenden Anfragen in der Ausgabe des Servers überprüfen.

Immer wieder musste ich abwägen, wie detailliert ich Modelle oder deren Attribute testen soll. Der Datentyp wird in den Migrations bereits festgelegt und kann daher gar nicht verletzt werden, da selbst ohne Validation im Model ein SQL-Fehler das Speichern des Objekts verhindert. Hier habe ich versucht, einen gesunden Mittelweg zu finden und nur Eigenschaften zu testen, die dann mittels Quellcode in den Models tatsächlich sichergestellt werden.

2.3.6.2. Organisatorische Probleme

Obwohl die Vorgehensweise der Implementierung gut vorher geplant war, bin ich organisatorisch über einige Dinge gestolpert.

So habe ich beim Implementieren des Menu-Models die Vererbung von Article komplett vergessen, bis mir beim Controller auffiel, dass ich so gar keine Preise vergeben kann. Das kann sicher auch daraus resultieren, dass ich Basis- und Mensasystem in zwei Diagrammen modelliert und so den Zusammenhang verloren habe.

Unabhängig davon habe ich in den Modellen zum Teil Attribute vergessen, deren Notwendigkeit sich dann erst später gezeigt hat. Beim Erstellen der Abrechnung musste ich die Frage beantworten, welche Bestellungen in einer Abrechnung zusammenzufassen sind. Dadurch wurde es notwendig, ein Rechnungslegungsdatum für den Artikel festzulegen, was ich vorher nicht geplant hatte.

Meine Entscheidung statt „git“ auf „Subversion“ als Repository zu setzen habe ich auch mehrmals bereut. Das Hinzufügen vieler Dateien, bei einem Plug-in oder später der Railsumgebung, zum Repository führte mehrmals dazu, dass meine Umgebung dachte, gewisse Dinge noch einreichen (im englischen „to comit“) zu müssen, welche im Repository schon bekannt waren. Ich vermute, dass hier das Subversionplug-in für Netbeans die Problemursache war.

2.3.7. Abschlussstand

2.3.7.1. Was ist implementiert

Die Implementierung verlief insgesamt erfolgreich. Die grundlegende Funktionalität ist vorhanden und die in der Vorbereitung erstellten Modelle wurden umgesetzt.

Den Quellcode der neuen Anwendung habe ich unter die GPL Version 3 [28] gesetzt um anderen weitreichende Möglichkeiten zur Verwendung meiner Arbeitsergebnisse zu geben. Veröffentlicht wird der Quellcode zuerst auf einer Webseite am Institut und später auf meiner privaten Homepage unter http://www.armin-feistenauer.de.

Hier ein kleiner Einblick in den Speiseplan des alten und des neuen Systems. Das Design der neuen Anwendung ist noch sehr rudimentär.

Abbildung 9: Speiseplan im alten System mit vielen Modulen in der Navigationsleiste

Abbildung 10: Speiseplan im neuen System, mit Erfolgsmeldung für die Bestellung

2.3.7.1.1. Entwicklungsstand Klassendiagramme

Um den derzeitigen Umfang der Software zu zeigen habe ich den aktuellen Entwicklungsstand in neuen Diagrammen dokumentiert.

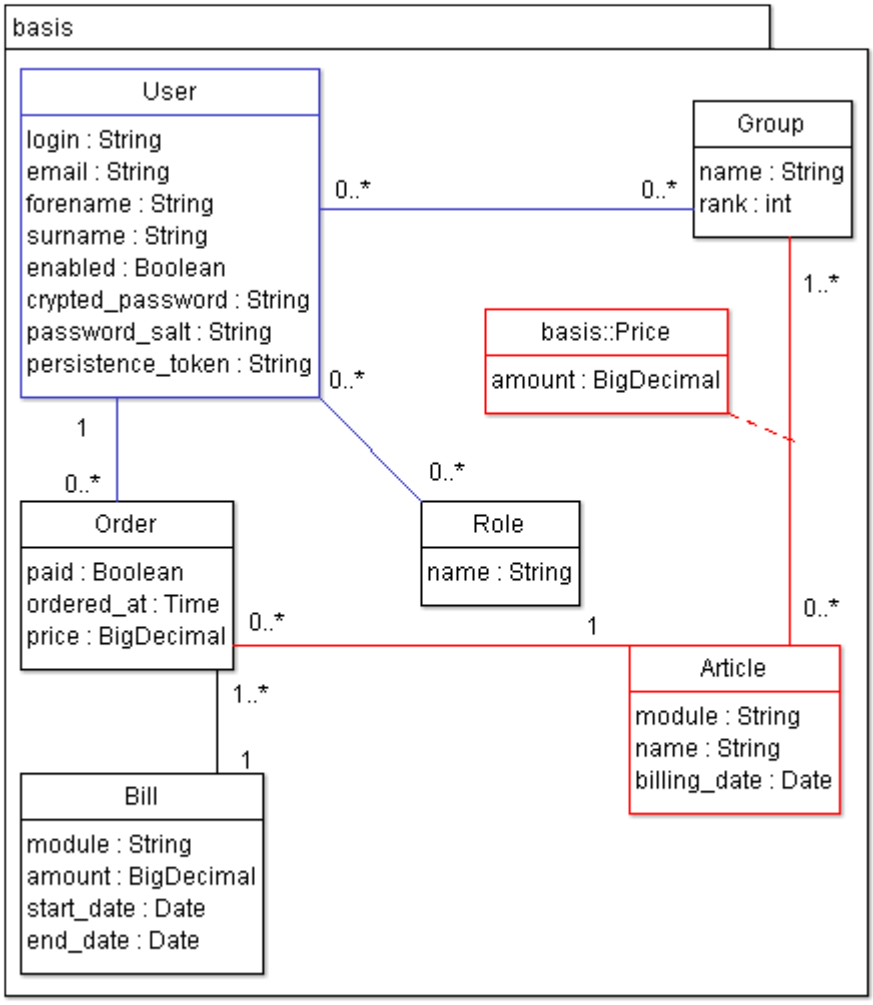

Abbildung 12: Klassendiagramm des Basismoduls (aktueller Entwicklungsstand)

Die im Entwurf (siehe Kapitel 2.2.2.1) vorhandene Klasse Berechtigung wird hier nicht mehr aufgeführt, da Autorisierung mit einer Konfigurationsdatei realisiert wird, in der Berechtigungen gesetzt und mit Rollen verknüpft werden.

Durch die noch nicht implementierten Klassen Bewertung und Abo sieht das Mensamodul deutlich einfacher aus:

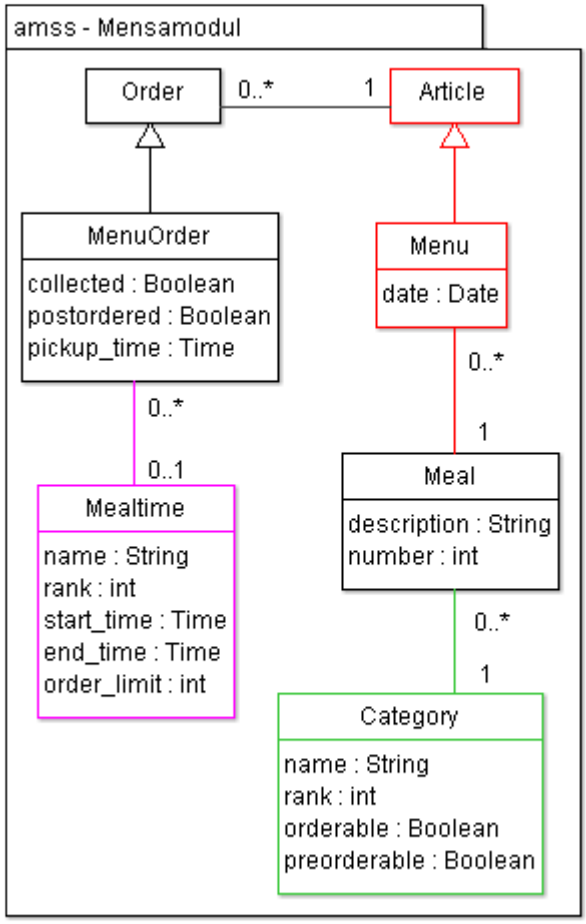

Abbildung 13: Klassendiagramm des Mensamoduls (aktueller Entwicklungsstand)

2.3.7.1.2. Konfiguration

In der Datei „config/config.yml“ können Einstellungen für die Anwendung vorgenommen werden. So lässt sie sich für verschiedene Kunden individuell anpassen. Wie geplant wird hier keine SQL-Lösung mehr benutzt, wodurch sich Einstellungen nicht zur Laufzeit verändern lassen.

Diese Implementierung basiert auf der Beispielimplementierung einer YAML-Konfigurationsdatei [29].

```yaml
#custom configuration
defaults: &defaults
  #BASIS STUFF
  basis_nav_heading: "Testschule"

  #MENSA STUFF
  #number of weeks the menu is shown for should be >0
  mensa_menu_show_weeks: 3

  #number of different prices for one menu
  mensa_number_of_different_prices: 2

  #OTHER MODULE STUFF

development:
  <<: *defaults

test:
  <<: *defaults

production:
  <<: *defaults
```

Abbildung 14: config.yml Konfigurationsdatei

2.3.7.1.3. Rechteverwaltung

Die Berechtigungen werden in einer Datei festgelegt und dann mittels „filter_access_to:all“ in den Controllern eingebunden.

```ruby
authorization do

  role :guest do
    has_permission_on :home, :to => [:index]
  end

  role :user do
    includes :guest

    has_permission_on :users, :to => [:update]
    has_permission_on :menu_orders, :to => [:create]
    has_permission_on :menus, :to => [:index]

  end
```

Abbildung 15: authorization_rules.rb Berechtigungsdatei (Ausschnitt)

Hier sieht man die Berechtigungen der Rollen „guest“ und „user“. Die Rolle „guest“ hat jeder Besucher, der keine andere Rolle besitzt. Dieser darf hier im HomeController die index Methode aufrufen, welche die Startseite nach der Anmeldung anzeigt.

Die Rolle „user“ enthält alle Rechte der guest-Rolle, sowie die Möglichkeit, das eigene Profil zu bearbeiten, Bestellungen zu tätigen und den Speiseplan anzuzeigen.

2.3.7.2. Was ist unerwartet hinzu gekommen

Beim Order-Model habe ich mich doch dafür entschieden, den Preis der Bestellung zu speichern, um sicherzustellen, dass dem Benutzer nur der vereinbarte Betrag in Rechnung gestellt wird.

Bei der Abrechnung kam sogar eine neue Klasse „Bill“ hinzu um diese Funktionalität zu bündeln und handhabbar zu machen. Eine Abrechnung besteht aus mehreren Bestellungen, die zu einem Modul gehören und sich in einem Abrechnungsintervall befinden.

2.3.7.3. Was ist noch nicht implementiert

Einige Dinge sind gewollt oder aus Zeitgründen noch nicht implementiert.

Als grundlegendes Strukturmerkmal dient die Modularisierung der Software. Hier habe ich noch nicht alle Möglichkeiten von Rails ausgeschöpft. So liegen alle Controller, Models und Views jeweils in einem gemeinsamen Ordner, egal zu welchem Modul sie gehören. Dieses Thema wird erst richtig akut, wenn es ein weiteres Modul neben dem Mensasystem gibt und es möglich sein soll, nur eins oder beide davon zu betreiben.

Die Klasse „Fehlermeldung“ im Basismodul habe ich nicht implementiert. Der Grund dafür ist nicht die Komplexität der Umsetzung, denn ich erwarte dabei keinerlei Schwierigkeiten, sondern meine Bedenken über den Sinn.

In der Implementierungsphase hatte ich die Idee für ein umfassenderes Meldungssystem. Dieses soll z. B. erfolgreiche (Ab-)Bestellvorgänge, fehlgeschlagene Loginversuche und vieles mehr speichern. Vielleicht könnte es dazu Meldungen in verschiedenen Kategorien, wie Warnung, Fehler und Erfolg, geben, die der Benutzer dann einsehen kann.

Der Speiseplan kann momentan zwar angelegt aber nicht mehr bearbeitet werden. Auch ist es nicht möglich, für andere Benutzer zu bestellen, wie es für die Verwaltung vorgesehen ist. Im alten System gab es eine Einstellung, wie lange vor der Essensausgabe man noch Essen bestellen und abbestellen darf. Diese ist momentan noch nicht implementiert, wodurch der Benutzer bereits abgeholte Essen noch abbestellen könnte und so der Zahlung entgeht. Das ist natürlich nicht einsatzfähig.

Aus Zeitgründen habe ich darauf verzichtet, das Anlegen von Benutzern oder das Einstellen des Speiseplans mittels csv-Dateien zu implementieren. Durch das Hochladen einer Datei sollten alle darin vorkommenden Datensätze im System gespeichert werden. Bis auf den Umgang mit einer Datei ist das aber keine grundlegend neue Funktionalität, weshalb sie die prinzipielle Eignung der „Ruby on Rails“-Anwendung nicht in Frage stellt.

2.4. Evaluation der neuen Software

Nachdem der Entwicklungsstand der Software nun eingefroren ist, möchte ich im Rahmen der Bachelorarbeit eine ähnliche Softwarequalitätsanalyse, wie für die alte Software durchführen und vergleichend auf Änderungen hinweisen.

Die hier gewonnenen Erkenntnisse helfen mir dann hoffentlich auch, den Erfolg oder Misserfolg der verschiedenen Ziele bewerten zu können.

2.4.1. Softwarequalität

2.4.1.1. Code Smells

Wie schon bei der alten Software möchte ich die neue Software auf Code Smells untersuchen. Dabei möchte ich wieder die gleichen Smells überprüfen.

„Duplicate Code“, „Large Method“ und „Duplicate Method“.

2.4.1.1.1. Duplicate Code

Bei der alten Software habe ich als Beispiel die Dateien „speiseplan.php“ und „speiseplan_bearbeiten.php“ aufgeführt (siehe 2.1.3.1.1), die beide den Speiseplan anzeigen aber einmal zum Bestellen und einmal zum Bearbeiten dienen. Als dritte Datei dazu gibt es die „speiseplan_eintragen.php“ um neue Menüs zum Speiseplan hinzuzufügen.

Im neuen System ist es zwar noch nicht möglich, den Speiseplan zu bearbeiten, aber es kann bereits ein Speiseplan eingetragen und danach bestellt werden. Diese Funktionalität ist nun in der gleichen Action (MenusController, Action index) realisiert und wird im View je nach Nutzerberechtigung unterschiedlich angezeigt.

```
1  <table width="600"><tr><th colspan="<%= week.size %>">Woche vom <%= week.first[:date].to s(:german) %>
2    <% form_for :menu_order, :url => { :controller => "menu_orders", :action => "create" }  do |f| %>
3    <tr>
4      <%= render :partial => "day", :collection => week %>
5    </tr>
6    <tr>
7      <td colspan="<%= week.size %>">
8        <%= f.submit "Bestellung abschicken" %>
9        <% end %>
10     </td>
11   </tr>
12   <% permitted_to? :create, :menus do %>
13     <tr>
14       <td colspan="<%= week.size %>">
15         <%= render :partial => "new", :locals => {:menu => @menu, :start_date => week.first[:date]} %>
16       </td>
17     </tr>
18   <% end %>
19 </table><br />
```

Abbildung 16: Wochenteilview des Speiseplans

In Zeile 12 wird hier überprüft ob der Benutzer das Recht zum Anlegen neuer Menüs besitzt und nur in diesem Fall ein Formular dafür angezeigt. Der Code zum Anzeigen des vorhandenen Speiseangebots wird in Zeile 4 für alle Benutzer gleichermaßen ausgeführt.

Insgesamt gibt es nun keine Ansammlungen sehr ähnlichen Codes. Im MealtimesController wird in drei Methoden als erste Zeile ein Objekt mit einer ID geladen. Das könnte in eine eigene Funktion ausgelagert werden, die dann vor diesen Methoden aufgerufen wird.

```
@mealtime = Mealtime.find params[:id]
```

Abbildung 17: Erste Zeile der Methoden edit, update und destroy im MealtimesController

Ob Codedopplungen nun dauerhaft vermieden werden können ist aber noch nicht geklärt.

Einerseits habe ich noch keinerlei Gebrauch vom Konstrukt „Helper“ gemacht, welche eigentlich dazu dienen, häufig verwendeten Code zentral verfügbar zu machen. Andererseits gibt es noch nicht viel ähnliche Funktionalität, die dazu einlädt, den gleichen Code immer wieder zu verwenden.

Momentan sieht es gut aus, aber allgemeine Aussagen würde ich daraus nicht ableiten. Als Pluspunkt kann aber hier noch angemerkt werden, dass zum Editieren und Erstellen von Objekten einer Klasse meist ein Partial, d. h. ein View das nur als Teil anderer Views eingebunden wird, Verwendung findet, wodurch redundanter Code vermieden wird.

2.4.1.1.2. Large Method

Hier habe ich wieder manuell nach den größten Funktionen im Anwendungscode gesucht und diese aufgelistet. Während ich bei der letzten Evaluation nur die Länge ohne Quellcode dokumentiert habe, wollte ich hier auch überprüfen, ob lange Funktionen mit Kommentaren erklärt werden.

Controller / Action	**Länge mit Kommentaren**	**Länge ohne Komentare**
MenusController / create	42 Zeilen	37 Zeilen
BillController / create	30 Zeilen	27 Zeilen
MenusController / index	24 Zeilen	18 Zeilen
MenuOrdersController / create	20 Zeilen	19 Zeilen

Die beiden größten Funktionen haben durchaus eine stattliche Länge, die aber auch aus vielen Leerzeilen zur Lesbarkeit resultiert.

Bis auf die letzte Funktion in der Tabelle sind die Funktionen auch relativ gut kommentiert, zumindest überdurchschnittlich im Vergleich zum restlichen Programm. (siehe 2.4.1.2. Datenblatt Softwarequalität)

In der alten Software gab es auch im Bereich des Speiseplans die besonders großen Funktionen, die dort jedoch aus über 100 Zeilen Quellcode bestanden. Hier gab es also auf jeden Fall eine deutliche Verbesserung.

Das hängt sicher auch damit zusammen, dass in den PHP Funktionen die Ausgabe von HTML mit echo „HTML-Code“; als Quellcode gelten, während es in den Rails-Controllern keine Ausgabe gibt. Diese findet in den Views statt, welche nicht als Quellcode gelten und keine Funktionen sind.

2.4.1.1.3. Duplicate Method

Bei Duplicate Method suche ich zwei oder mehr Funktionen, die die gleiche Aufgabe erfüllen. Hier bin ich nur auf die update-Funktionen im UsersController gestoßen. Die Action „update“ dient dazu, den Benutzer seine eigenen Profildaten, also seine E-Mail-Adresse und sein Passwort, bearbeiten zu lassen. Mit „update_all“ verwaltet der Administrator Benutzer und darf auch deren Ausweisnummern und Stammdaten verändern.

```
def update
  @user = @current_user

  #only update email and password. other values are updated with update_all
  if @user.update_attribute(:email, params[:user][:email]) && @user.update_
    flash[:notice] = "Profil erfolgreich bearbeitet!"
    redirect_to edit_account_url
  else
    render :action => :edit
  end
end
```

Abbildung 18: UsersController / update-Action

```
def update_all
  if params[:id]
    @user = User.find params[:id]
  else
    flash[:error] = "Kein Benutzer ausgewählt!"
    redirect_to users_url
  end

  if @user.update_attributes(params[:user])
    flash[:notice] = "Änderungen erfolgreich gespeichert!"
    redirect_to users_url
  else
    render :action => :edit
  end
end
```

Abbildung 19: UsersController / update_all-Action

Die letzte if-else-Anweisung ist in beiden Methoden ähnlich und könnte für eine Zusammenlegung sprechen. Die Möglichkeit, den Zugriff auf Methodenebene zu verwalten, vereinfacht meiner Meinung nach den Ablauf aber ungemein und rechtfertigt zwei Methoden.

Sonst müsste diese Abfrage innerhalb der Methode durchgeführt werden, was den Vorteil vermutlich wieder zunichte machen würde. Diese Situation ändert sich, wenn die Übereinstimmungen noch stärker werden, falls beide Methoden an Komplexität gewinnen.

2.4.1.2. Datenblatt Softwarequalität

2.4.1.2.1. LOC + Kommentare

Die Lines of Code habe ich wieder mit zwei verschiedenen Werkzeugen gemessen um die Vergleichbarkeit zu gewährleisten. Dieses Mal kommen beide auf das gleiche Ergebnis.

Objekt	**Lines of Code ohne Kommentare**
ESV gesamt	2511
Tests	1397
App Folder	591
Controller	438
Models	129

Tabelle 5: Source Lines of Code (SLOC): gemessen mit SLOCCount entwickelt von David A. Wheeler
Aufruf durch „sloccount Ordner“

Objekt	**Lines of Code (Ruby)**	**Kommentarzeilen**	**Kommentaranteil**
ESV gesamt	2856	302	9,56%
Tests	1397	106	6,9%
App Folder	591	45	7,1%
Controller	438	33	7%
Models	129	11	7,9%

Tabelle 6: Lines of Code gemessen mit cloc entwickelt von Al Danial Copyright (c) 2006-2009, Northrop Grumman Corporation / Information Technology / IT Solutions
Aufruf durch „cloc-1.08.exe PfadZumOrdner“

Unter „ESV gesamt“ verstehe ich hier die Ordner „tests“, „app“, „db“ und „config“. Den ganzen Projektordner analysieren zu lassen macht keinen Sinn, da dort auch viele Rubyscripte liegen, die nicht von mir erstellt wurden.

Im Ordner „config“ liegt die Datei „authorization_rules.rb“, die für die Berechtigungen zuständig ist und die „environment.rb“, in welcher Plug-ins eingebunden und Einstellungen getätigt werden. Letztere Datei gibt es in jedem Rails Projekt und sie ist standardmäßig mit vielen Kommentarzeilen versehen, was bei meinem noch kleinen Projekt den gesamten Kommentaranteil in die Höhe zieht.

Im „db“-Ordner liegen die Migrations und das momentane Datenbankschema. Die Migrations enthalten ebenfalls von mir erstellten Quellcode und sollen daher auch mitgezählt werden.

Was auch auffällt ist, dass deutlich mehr Anwendungscode in den Controllern als in den Models steckt. Die Models bestehen meist nur aus einer Ansammlung von Validations.

Eigentlich setzt Rails auf das Prinzip „skinny Controller, fat Model“ [30]. Der Anwendungscode sollte also in den Models untergebracht werden.

Dass dies bei meinem System nicht der Fall ist, kann einerseits daran liegen, dass ich noch mit zu wenig verschiedenen Wegen auf die Daten zugreife und so kaum Funktionen in den Models benötigt habe. Durch eine größere Funktionsvielfalt könnte sich das also noch verändern.

Vielleicht habe ich aber auch Aufgaben in den Controllern erledigt, die in das Model verlagert werden sollten. Das kann nur ein erfahrener Rails-Entwickler entscheiden.

Wie man in der zweiten Tabelle gut sieht, ist der Kommentaranteil bei den von mir erstellten Dateien durchgängig etwa 7 %. Nach meiner Auswertung kam das alte System insgesamt zwar auch auf fast 7 % Kommentaranteil, allerdings mit unerklärlichen Resultaten in der „funktionen.php“-Datei im Basissystem. Realistischer ist die beim Modul AMSS errechnete Zahl von 2,2 % womit sich diese Kennzahl im neuen System deutlich verbessert hat.

2.4.1.2.2. Laufzeit

Jede Aufgabe wird hier 5-mal ausgeführt und der Durchschnitt als Zeit angegeben.

Aufgabe	**Ruby on Rails**	**PHP (Messung von oben)**
Startseite (nicht angemeldet)	0,02058 s	0,14259 s
Anmelden (Anzeigen der Startseite nach Anmeldung)	0,03174 s	0,26527 s
Benutzer anlegen	0,07489 s	0,24167 s
Menü eintragen	0,43720 s	0,21666 s
Menü bestellen	0,60782 s	0,22180 s

Tabelle 7: Seitenaufbauzeiten im Vergleich: RoR mit Mongrel Server, PHP mit Apache 2

Bei den vier unteren Funktionen können die tatsächlichen Zeiten beim „Ruby on Rails“-System noch höher liegen. Das liegt daran, dass nach der Aufgabe zu einer anderen Action weitergeleitet wird und meine Zeitmessung sich immer nur auf die Berechnung der letzten Action bezieht.

Hier wird also vermutlich nur der Aufwand für die Speiseplan- oder die Benutzerlistenanzeige gewertet. Das erklärt aber nicht, warum ein Menü zu bestellen länger dauert als das Menü eintragen, da beide schlussendlich den Speiseplan anzeigen.

Angesichts dieser doch sehr mäßigen Messwerte, die noch dazu auf unterschiedlich stark gefüllten Datenbanken erhoben wurden, muss man diese sehr vorsichtig auswerten. Da die Ergebnisse sich aber um Größenordnungen von denen des alten Systems unterscheiden, lassen sich meiner Meinung nach trotzdem Aussagen treffen.

Die einfache Anzeige der Startseite geht im neuen System deutlich schneller. Der Aufwand für das Routing, wenn er denn von dieser Messung erfasst wird, scheint also nicht ins Gewicht zu fallen.

Die Anzeige der einfachen Startseite nach der Anmeldung geht ähnlich schnell und auch die

Benutzerliste anzuzeigen dauert nicht viel länger. Hier hat das PHP-System immer noch das Nachsehen, wobei das auch ein Messfehler sein kann.

Deutlich ist hingegen das Ergebnis bei den unteren beiden Aufgaben. Die Anzeige des Speiseplans ist in PHP viel schneller als in meinem neuen System, das bei einem relativ leeren Speiseplan mit der Anzeige von drei Wochen schon in etwa eine halbe Sekunde benötigt und damit eine wahrnehmbare Verzögerung verursacht.

Die Anzeige vieler ineinander geschachtelter Partials scheint hier viel Zeit zu brauchen. Wenn sich das Ergebnis mit mehr Menüs und Bestellungen noch etwas verschlechtert, muss die Performance mit Refaktorierung verbessert werden.

Meine Vermutung nach diesen Ergebnissen ist, dass die Railsanwendung wegen des Frameworks nicht prinzipiell langsamer ist als eine PHP-Anwendung, aber die Anzeige komplexerer Seiten zu Geschwindigkeitseinbußen führen kann. Die Performance muss bei der Weiterentwicklung also kritisch im Blick behalten werden und vielleicht mit Lasttests noch detaillierter erhoben werden.

Natürlich kann auch an den Servereinstellungen noch einiges optimiert werden um die Geschwindigkeit zu verbessern.

2.4.1.2.3. Testabdeckung

Mit TDD war es nicht schwer, die Testabdeckung des alten Systems von 0 % zu übertreffen.

Die Models und Controller sind nun zum überwiegenden Teil mit Tests abgedeckt. An manchen Stellen habe ich vielleicht etwas mehr Quellcode erstellt, als für den Test tatsächlich notwendig gewesen wäre. Die Testabdeckung sollte aber fast 100 % betragen.

Die Views und Berechtigungen sind, wie bereits weiter oben erwähnt, nicht mit Tests abgedeckt.

2.4.1.3. Fazit der Softwarequalitätsveränderung

Bis auf die Laufzeit haben sich die hier erhobenen Metriken durch den Reengineeringprozess deutlich verbessern lassen. Die Ursachen dafür sind sehr vielfältig.

Gerade bei den Code Smells kommt sicher die Grundstruktur des Railsprojekts zum Tragen. Präsentation und Anwendungslogik nicht zu trennen wird dadurch fast aufwändiger als es zu tun, wodurch die Qualität natürlich sofort steigt.

TDD resultiert automatisch in einer hohen Testabdeckung für die Bereiche, auf die es angewendet wird. Dafür habe ich wahrscheinlich mit einer verlängerten Entwicklungszeit bezahlt.

Der größere Kommentaranteil ist weder Ruby on Rails noch der testgetriebenen Entwicklung zuzurechnen. Meiner Meinung nach schreckt die testgetriebene Entwicklung sogar etwas vom Kommentieren ab, da die Überlegung, wie etwas umgesetzt wird, in den Tests zum Teil schon vorweggenommen wird. Dadurch ist die Dokumentation dieses Überlegungsprozesses in Kommentaren dann etwas unnatürlich. Diese Empfindung spricht etwas gegen die Qualität meiner Tests, da gute Tests unabhängig von den inneren Abläufen der getesteten Funktion sein sollten.

3. Fazit

Am Ende meiner Bachelorarbeit möchte ich noch ein Fazit ziehen. Zuerst blicke ich auf die selbst gesetzten Ziele zurück und bewerte, ob diese erreicht wurden oder nicht.

Dann gehe ich in meinem persönlichen Fazit auch noch darauf ein, welche Erfahrungen ich durch diese Arbeit gemacht habe und was mir das Ganze gebracht hat.

3.1. Rückblick auf die Ziele

3.1.1. im Bereich Softwarequalität

- **Verbesserung der Struktur durch objektorientierte Programmierung**
  Eine Strukturverbesserung ist schwer zu messen. Durch die Objektorientierung greife ich nun aber beim Zugriff auf den aktuellen Benutzer auf Methoden im User-Model zurück und habe diese Funktionalität so gebündelt an einer Stelle. Die Verschränkung zwischen den verschiedenen Klassen hat dadurch abgenommen.
- **Zuverlässigkeit und geringe Fehlerrate durch eine höhere Testabdeckung**
  Die höhere Testabdeckung wurde erreicht. Ob sich die Zuverlässigkeit dadurch verbessert und im Betrieb tatsächlich weniger Bugs auftreten, kann ich nicht beurteilen. Nach allen Erfahrungswerten ist aber davon auszugehen, dass ein (gut) getestetes Programm weniger Defekte enthält.
- **Wartbar- und Erweiterbarkeit**
  Das hätte ich gerne getestet. Leider war das in diesem Rahmen nicht mehr realisierbar.
  Die Erweiterbarkeit stellt sich bisher aber sehr positiv dar. In meinem Entwicklungsprozess habe ich ja Klasse nach Klasse hinzugefügt. Natürlich war das einfach, da diese in der Planung bereits vorhanden waren.
  Doch auch die ungeplante Klasse „Bill“ für die Abrechnung ließ sich sehr schnell implementieren. Eine größere ungeplante Veränderung musste die Struktur aber noch nicht aushalten. Ob sich die Wartbarkeit also verbessert hat, bleibt offen.
- **Verständlichkeit des Quellcodes durch Kommentare und Programmierstil**
  Ruby gilt als Sprache, die sehr einfach zu lesen und intuitiv zu verstehen ist. Daher war ein Erfolg dieses Ziels mit der Plattformwahl bereits fast sicher. Durch die Erhöhung des Kommentaranteils im Vergleich zu meiner früheren Leistung im Modul AMSS betrachte ich dieses Ziel als erreicht .

3.1.2. im Bereich Funktionalität

- **Die Bedienung soll weniger komplex sein**
  Das ist sehr schwer zu bewerten. Durch den geringeren Funktionsumfang ist die Bedienung momentan sicher übersichtlicher. Allerdings habe ich das Layout dem alten System angepasst, wodurch sich hier weniger verändert hat. Dieses Ziel halte ich inzwischen für schlecht gewählt, da mir die Bedeutung nicht ganz klar wird und es so für mich nicht messbar ist.

- **Die Benutzerführung soll vereinheitlicht werden**
  Das Untermenü innerhalb des Controllers wurde in ein horizontales Menü am Seitenkopf verlagert. Fast jede Klasse wird über ihre Indexfunktion als Liste der vorhandenen Objekte angezeigt. Neben den Attributen gibt es Links zum Bearbeiten und Löschen. Das ist einheitlicher als früher.
- **Ungenutzte Funktionalität soll entfernt werden**
  Funktionalität weg zu lassen ist ein sehr leicht erfüllbares Ziel. Schwieriger wird es werden, möglichst viele Kundenwünsche zu erfüllen ohne wieder zu einer Vielzahl selten oder gar nie benutzter Funktionen zu kommen.
- **Das System soll bessere, mehr und einheitlichere Rückmeldungen an den Benutzer liefern**
  Ich glaube nicht, dass das System mehr oder informativere Rückmeldungen an den Benutzer gibt. Ausgenommen davon ist die von Rails automatisch zur Verfügung gestellte Anzeige, welche Werte in einem Formular zum Fehler beim Speichern des Objekts geführt haben. Alle anderen Fehler- und Erfolgsmeldungen erscheinen nun am Anfang der Seite und müssten in einem verbesserten Stylesheet noch mehr hervorgehoben werden. Die Einheitlichkeit ist halb erfüllt, die anderen Punkte sind noch offen und können vielleicht im Zusammenhang mit dem angedachten Meldungssystem realisiert werden.

3.1.3. im Bereich Plattformwechsel

- **Wechsel der Programmiersprache von PHP auf Ruby**
- **Wechsel von selbst geschriebenem PHP-Projekt zu Ruby on Rails als Framework**
- **Wechsel von Script auf objektorientierte Programmierung**
- **Einführung einer MVC (Model-View-Controller) Struktur**

Die „Ziele“ in diesem Bereich wurden offensichtlich erreicht. Mit der erfolgreichen Durchführung des Reengineerings in Ruby on Rails erledigen sich diese Punkte automatisch.

3.1.4. Erfolgsbewertung

Ziele die ich erreichen musste, habe ich tatsächlich erreicht. Bei den meisten anderen Zielen habe ich vor Beginn der Bachelorarbeit erwartet, diese durch das Reengineering erreichen zu können.

Da viele dieser Ziele schwer zu überprüfen sind, gehe ich nun meist davon aus, diese wie erwartet erreicht zu haben. Das ist natürlich eine sehr subjektive Sicht und kann keine allgemeingültigen Aussagen über die Auswirkungen eines Reengineerings machen.

Einige wenige Ziele hätte ich verfehlen können und habe sie nach meiner Einschätzung, belegt zum Beispiel mit dem erhobenen Kommentaranteil, tatsächlich erreicht. Das waren vermutlich die besten Ziele, da sie überprüfbar sind und nicht erreicht werden mussten.

3.2. Persönliches Fazit

Von der Bachelorarbeit hatte ich mir zwei Dinge erhofft.

Erstens eine neue Herausforderung und die ernsthafte und realitätsnahe Beschäftigung mit Ruby on Rails in einer nützlichen Anwendung. Und zweitens die Notwendigkeit, auf mein im Studium erarbeitetes Wissen zurückzugreifen und dieses in die Arbeit einbringen zu können.

Eine Herausforderung war diese Arbeit ohne Zweifel.

Obwohl ich einen klaren Zeitplan hatte, geriet ich gleich am Anfang durch organisatorische Missverständnisse und die Klausurphase an der Uni in einen Rückstand. Wenn man diese Verzögerung in Betracht zieht, konnte ich mich im weiteren Verlauf auf meine Einschätzungen, wie lange ich für gewisse Aufgaben brauchen würde, verlassen.

Der Modellierungsprozess war sehr anstrengend, da es mir nicht gelungen ist, konkrete Teilaufgaben zu definieren, die sich einfach erreichen lassen. Im Gegensatz dazu lief das begleitende Schreiben der Bachelorarbeit meist sehr angenehm ab. Mit kurzen Stichpunkten konnte ich schnell Ideen für Absätze und die Struktur festhalten und diese nachher einzeln ergänzen.

Bei der Implementierung fiel es mir immer schwer, zuerst die Tests zu schreiben, die denen für andere Controller und Models oft sehr ähnelten. Die Tatsache, dass ich mehr Testcode als Anwendungscode geschrieben habe, (wenn man die Views nicht mitzählt) ist mir schon früh negativ aufgefallen. Tests als etwas Selbstverständliches zu sehen, muss ich noch verinnerlichen.

Die Analyse der neuen Software war dann sehr schnell erledigt, da die Werkzeuge bereits herausgesucht und bekannt waren. Leider konnte ich hier keine aufwändigeren Versuche mehr durchführen.

Auch meine zweite Erwartung, auf Studieninhalte zurückgreifen zu können, hat sich erfüllt.

Die in den diversen „Algorithmen und Programmierung“ erlernten Programmierkenntnisse waren sicher hilfreich bei der Implementierung, auch wenn ich die meisten weiterführenden Kurse nicht direkt verwenden konnte.

Nicht ganz zufällig konnte ich meine Kenntnisse im Bereich Softwaretechnik einbringen, auf die diese Aufgabe natürlich ein Stück weit zugeschnitten war.

Zu guter Letzt halfen mir meine Railsvorkenntnisse aus dem Proseminar Webtechnologie, eine realistische Einschätzung des Aufwands und der Möglichkeiten von Rails zu tätigen.

So hinterlässt die Arbeit das befriedigende Gefühl, dass sich die Studieninhalte tatsächlich anwenden lassen und ich bei mir einen Fortschritt in meinen Fähigkeiten als Informatiker feststellen kann.

Mit der neuen Software bin ich ziemlich zufrieden, auch wenn noch viel Arbeit zu erledigen ist, bis diese das alte System tatsächlich ersetzen kann. Ob es sich rentiert, diese Arbeit zu investieren und den Sprung vom PHP-System auf das „Ruby on Rails“-System wirklich umzusetzen, möchte ich mit etwas Abstand beurteilen.

Außerdem ist an dieser Entscheidung natürlich mein Kollege Sven Seeberg ebenfalls beteiligt.

4. Anhang

4.1. Danksagung

Ich danke meinem Betreuer Prof. Dr. Lutz Prechelt dafür, dass er sich bereit erklärt hat diese Arbeit zu betreuen und mir dadurch die freie Gestaltung meiner Bachelorarbeit erlaubt hat.

Außerdem danke ich Prof. Dr. Adrian Paschke für die Zusage als Zweitgutachter zu dienen.

Für die Unterstützung bei der Durchführung dieser Bachelorarbeit danke ich meinem langjährigen Programmier- und Geschäftspartner Sven Seeberg, der immer ein offenes Ohr für meine Einfälle hat und als Realitätstest für neue Ideen unerlässlich war.

Für die schnelle Hilfe bei Problemen mit dem von ihm verwalteten Plug-in „validates_timeliness“ danke ich Adam Meehan und damit auch stellvertretend allen anderen Plug-inentwicklern, die die Ruby on Rails Community so bereichern.

Dank gilt auch meiner Mutter Angela Feistenauer, die mit ihren Hinweisen diese Ausarbeitung merklich verbessert hat. Alle Fehler und Unzulänglichkeiten gehen natürlich allein auf meine Kappe.

4.2. Eidesstattliche Erklärung

Ich versichere hiermit, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben.

Zitate aus fremden Arbeiten sind als solche kenntlich gemacht.

Diese Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 22. April 2010

Armin Feistenauer

4.3. Glossar

In diesem Glossar möchte ich kurz einige Begriffe erklären, die in der vorliegenden Bachelorarbeit verwendet werden und vielleicht nicht jedem Leser ein Begriff sind.

Controller Im MVC-Muster ist der Controller die Steuerungsinstanz welche Benutzereingaben entgegen nimmt und zwischen Präsentation (View) und Model vermittelt.

ESV Kürzel für elektronisches Schulverwaltungssystem eine von Sven Seeberg und Armin Feistenauer entwickelte Webanwendung.

IDE Steht für integrated development environment bzw. integrierte Entwicklungsumgebung im Deutschen und bezeichnet eine Anwendung für die Softwareentwicklung.

Migration Migrations sind Klassen, die in Rails dazu dienen Veränderungen an der Datenbankstruktur durchzuführen. Mit ihnen kann man z. B. neue Tabellen erstellen und bestehende Tabellen verändern, sowie mit Daten befüllen.

Model Das Model (deutsch Modell) ist Teil des MVC-Musters und für die Verwaltung der und den Zugriff auf die Daten verantwortlich.

MVC Abkürzung für das Model-View-Controller Architekturmuster welches zwischen Daten (Model), Präsentation (View) und Steuerung (Controller) unterscheidet.

Partial Ist ein View, das nur innerhalb von anderen Views angezeigt wird. Dient dazu Präsentationscode wiederzuverwenden.

Repository Im Zusammenhang mit Software handelt es sich dabei um einen versionsverwalteten Speicher für digitale Daten.

Reverse Engineering bezeichnet die Rekonstruktion eines Modells aus einem fertigen Softwaresystem.

TDD Steht für Test Driven Development und lässt sich mit „testgetriebene Entwicklung“ übersetzen. Bei diesem Programmieransatz werden zuerst die Tests und dann der Anwendungscode geschrieben.

Validation ist in Rails eine Funktion, die Eigenschaften eines Objekts sicherstellt. Rails bietet mehrere validations um Attribute zu überprüfen und die Speicherung nicht validierter Objekte zu verhindern.

View bezeichnet die Präsentationsschicht im MVC-Muster und in Rails eine Präsentationsdatei mit Ruby- und/oder HTML-Code.

4.4. Literaturverzeichnis

[1] Chikofsky, E. and Cross, J.: Reverse Engineering and Design Recovery: A Taxonomy. (IEEE Software), 1990, S.13-18.

[2] Seeberg S. und Feistenauer A.: Elektronische Schulverwaltung, 2009 http://www.esv-on.de/?page_id=22

[3] Mäntylä M.: Bad Smells in Software – a Taxonomy and an Empirical Study, 2003, http://www.soberit.hut.fi/sems/shared/deliverables_public/mmantyla_thesis_final.pdf

[4] Wheeler D.: SLOCCount, http://www.dwheeler.com/sloccount/

[5] Danial Al: Cloc, Copyright (c) 2006-2010, Northrop Grumman Corporation / Information Technology / IT Solutions, http://cloc.sourceforge.net/

[6] Clarke Gavin: Why is Ruby on Rails so darn slow? (The Register), 2008, http://www.theregister.co.uk/2008/04/21/bray_ruby_rails/

[7] Booch G, Rumbaugh J, Jacobson I.: The unified modelling Language User Guide, Second Edition (Addison Wesley), 2005, S. 103-131

[8] Jeckle M, Rupp Ch., Hahn J., Zengler B., Queins S.: UML2 glasklar (Hanser), 2004, S. 31-99

[9] ArgoUML „open source UML modeling tool“, http://argouml.tigris.org/

[10] Prechelt L., Vorlesung Softwaretechnik an der Freuen Universität Berlin, http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2010/22_Usecases.pdf

[11] Olszowka Ch.: The Ruby Toolbox – Know your options!, 2010, ruby-toolbox.com

[12] Olson R.: restful-authentication, 2008,2009, http://github.com/technoweenie/restful-authentication

[13] Bachir J. J., Johnson B.: authlogic „A simple model based ruby authentication solution“, 2008-2010, http://github.com/binarylogic/authlogic

[14] Bartsch S.: declarative_authorization „An authorization Rails plugin...“, 2008-2010, http://github.com/stffn/declarative_authorization

[15] Hellesoy A.: Cucumber – Making BDD fun, http://cukes.info/

[16] Chelimsky D.: Rspec, the original BDD Framework for Ruby, http://rspec.info/

[17] Beck K.: Test Driven Development: By Example (Addison-Wesley Longman), 2002

[18] Wolf H. (it-agile GmbH): Was ist Testgetriebene Entwicklung?, http://www.it-agile.de/wasisttdd.html

[19] Johann M.: Mit der IDE von der IDEE zur Realisierung (RailsWay Magazin 1.2009), 2009, S. 33-37

[20] Oracle Corporation: Netbeans IDE 6.8, © 2010, http://netbeans.org

[21] Git version control System, http://git-scm.com/about

[22] Chen N.: Convention over Configuration, 2006, http://softwareengineering.vazexqi.com/files/pattern.html

[23] Bachir J. J., Johnson B.: authlogic „A simple model based ruby authentication solution“, 2008-2010, http://github.com/binarylogic/authlogic_example

[24] Meehan A.: validates_timeliness „Date and time validation plugin for Rails 2.x“, 2008-2010, http://github.com/adzap/validates_timeliness

[25] Thomas D., Heinmeier Hansson D.: Agile Web Development with Rails (Second Edition), 2007, S. 344-348

[26] Thomas D., Heinmeier Hansson D.: Agile Web Development with Rails (Second Edition), 2007, S. 349-352

[27] Bates R.: Testing without Fixtures, 2007, http://railscasts.com/episodes/60-testing-without-fixtures

[28] FSF (Free Software Foundation): GNU General Public License, 29.06.2007, http://www.gnu.org/licenses/gpl-3.0.html

[29] Bates R.: YAML Configuration File, 2007, http://railscasts.com/episodes/85-yaml-configuration-file

[30] Buck J.: Skinny Controller, Fat Model, 2006, http://weblog.jamisbuck.org/2006/10/18/skinny-controller-fat-model

4.5. Sonstige Artefakte

4.5.1. Klassendiagramme

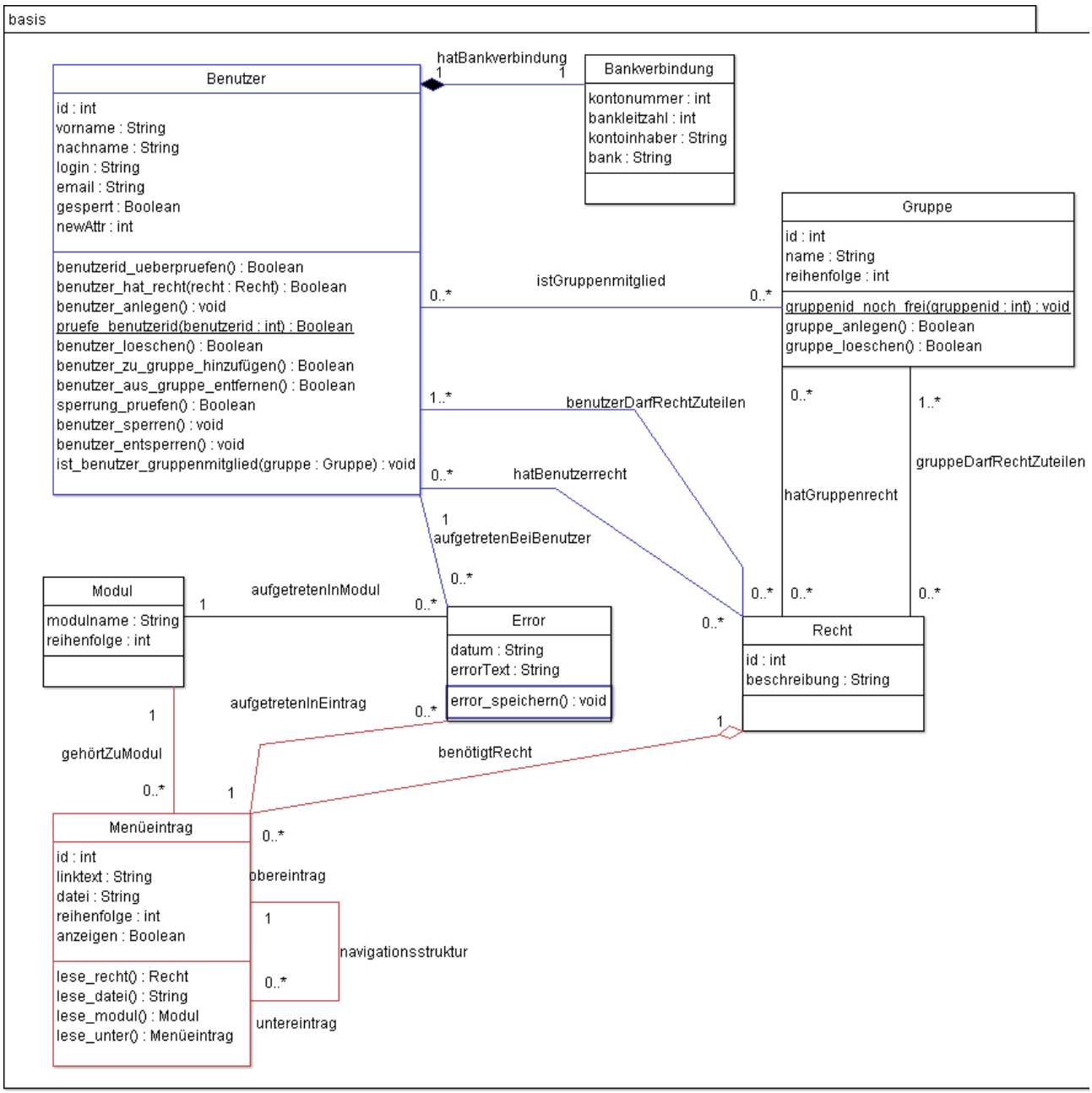

Abbildung 20: Klassendiagramm Basis mit Operationen

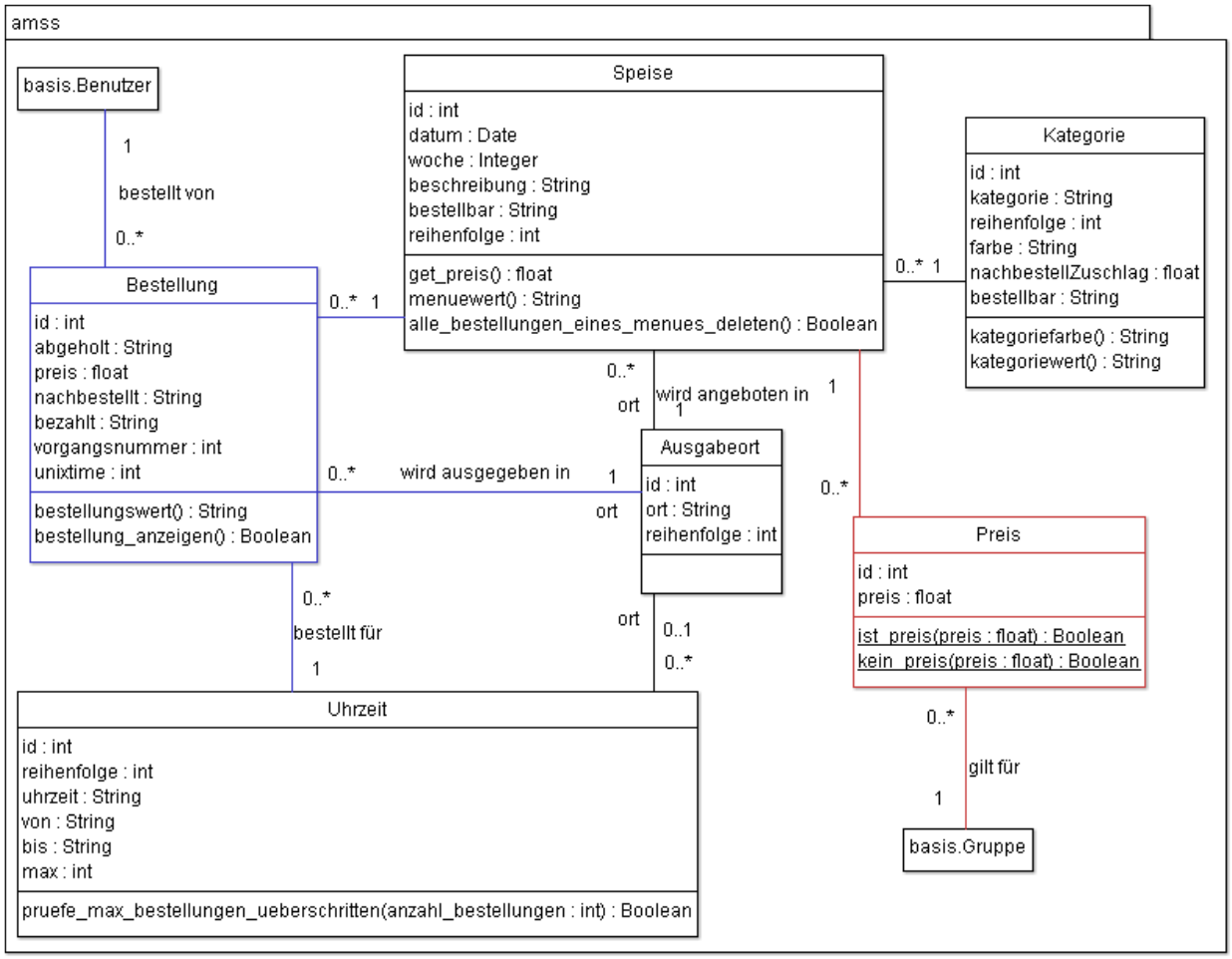

Abbildung 21: Klassendiagramm des AMSS-Moduls mit Operationen

4.5.2. Use Cases

Mensabestellungen erfassen und abrechnen

Hauptakteur: Caterer
Anwendungsbereich: Mensasystem
Niveau: Projektüberblick
Beteiligte und Interessen:

Caterer	Anzahl und Zusammensetzung der Bestellungen erfahren um diese in der Mensa auszugeben
Mensagast	Bestellung aufgeben und Speise bekommen
Mensaverwaltung	Bezahlung der Catererrechnung anhand der Mensagastbestellungen durchführen

Voraussetzung: Benutzer sind im System angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall: Caterer erfährt wie viele Essen bestellt wurden und wird bezahlt
Mensagast kann Essen bestellen und erhält dieses am Ausgabetag
Haupt-Erfolgsszenario:

1. Caterer trägt den Speiseplan ein (Speiseplan einstellen)
2. Mensagast bestellt (Essen bestellen) für ihn eingestellte Speisen
3. Caterer liest Anzahl der Bestellungen (Bestellungen einsehen) im System
4. Caterer gibt bestellte Essen aus
5. Mensaverwaltung rechnet Ausgabetage ab (Essen Abrechnen)
6. Mensaverwaltung bezahlt den Caterer

Erweiterungen: ---

Essen bestellen

Hauptakteur: Mensagast
Anwendungsbereich: Mensasystem
Niveau: Überblick
Beteiligte und Interessen:

Mensagast	Bestellung aufgeben und Speise bekommen
Caterer	Anzahl und Zusammensetzung der Bestellungen erfahren um diese in der Mensa auszugeben

Voraussetzung: Mensagast ist angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall:
Bestellung des Mensagasts wird gespeichert und Gast kann Essen abholen. Caterer sieht Bestellung des Mensagasts.
Haupt-Erfolgsszenario:

1. Mensagast navigiert zum Speiseplan
2. Mensagast wählt Essen aus (Essen auswählen)
3. Caterer gibt am Ausgabetag dem Mensagast das bestellte Essen aus
4. Essen wird durch Abrechnung bezahlt (Essen Abrechnen)

Erweiterungen: ---

Anzahl der bestellten Essen erfahren

Hauptakteur: Caterer
Anwendungsbereich: Mensasystem
Niveau: Überblick
Beteiligte und Interessen:

Caterer	Anzahl und Zusammensetzung der Bestellungen erfahren um diese in der Mensa auszugeben Anschließend möchte er für diese Bestellungen bezahlt werden.
Mensagast	Bestellung aufgeben und Speise bekommen

Voraussetzung: Mensagast ist angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall:
Bestellung des Mensagasts wird gespeichert und Gast kann Essen abholen. Caterer sieht Bestellung des Mensagasts.
Haupt-Erfolgsszenario:

1. Caterer stellt Speiseplan ein.
2. Mensagäste bestellen Essen (Essen bestellen)
3. Caterer sieht für einen Tag die Bestellungen ein (Bestellungen einsehen)
4. Caterer liefert Anhand dieser Daten die Essen
5. Caterer wird bezahlt

Erweiterungen: ---

Benutzer und Gruppen verwalten

Hauptakteur: Administrator
Anwendungsbereich: Schulverwaltungssystem (basis)
Niveau: Überblick
Beteiligte und Interessen:

Administrator	legt Benutzer und Gruppen an, löscht und verändert diese, sowie deren Rechte.

Voraussetzung: Administrator ist angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall:
Benutzer und Gruppen sind wie vom Administrator gewünscht im System eingetragen und können nur auf für sie freigegebene Systemfunktionen zugreifen
Haupt-Erfolgsszenario:

1. Administrator navigiert zur Verwaltung
2. Administrator verwaltet Gruppen (Gruppe anlegen)
3. Administrator verwaltet Benutzer (Benutzer anlegen)
4. Administrator legt Berechtigungen fest (Berechtigung setzen)

Erweiterungen: ---

Bestellungen abrechnen und für Benutzer bestellen

Hauptakteur:	Verwalter	
Anwendungsbereich:	Mensasystem	
Niveau:	Überblick	
Beteiligte und Interessen:	Verwalter	möchte Geld für die Bestellungen von den Mensagästen bekommen und diese beim Bestellvorgang unterstützen
	Mensagast	möchte Einsicht in zu erwartende Kosten haben um Rechnungen nachvollziehen zu können
Voraussetzung:	Verwalter ist angemeldet Speiseplan ist eingetragen	
Mindestzusicherung:	---	

Zusicherung im Erfolgsfall:

Jedes bestellte Essen im ausgewählten Zeitraum ist erfasst und für jeden Benutzer eine Rechnung oder eine Lastschrift erstellt.
Geld wird vom Mensagast eingezogen.
Mensagast kann Rechnung einsehen

Haupt-Erfolgsszenario:

1. Verwalter bestellt für Benutzer (Für Mensagast bestellen), falls dieser dazu nicht in der Lage ist. (optional)
2. Verwalter führt eine Abrechnung für einen gewissen Zeitraum durch (Essen Abrechnen)
3. Verwalter lässt entstehende Beträge vom Konto der Mensagäste abbuchen
4. Mensagast kann Rechnung einsehen

Erweiterungen: ---

Benutzer anmelden

Hauptakteuer: Benutzer
Anwendungsbereich: Schulverwaltungssystem
Niveau: Benutzerziel
Beteiligte und Interessen: Benutzer möchte im System auf Funktionen zugreifen
System muss absichern, dass nur berechtige Personen auf die Funktionen und Daten des Systems zugreifen
Voraussetzung: Benutzer kennt gültige Logindaten und ist im System eingetragen
Mindestzusicherung: Benutzer wird informiert warum er sich nicht anmelden kann
Zusicherung im Erfolgsfall:
Benutzer ist im System angemeldet
Benutzer kann auf geschützte (Anmeldung erforderlich) Funktionen zugreifen
Haupt-Erfolgsszenario:

1. Benutzer besucht die Startseite des Systems
2. Benutzer trägt seine Logindaten in die vorgesehenen Felder
3. Benutzer sendet die Daten ab
4. System validiert die Logindaten
5. Benutzer wird zur internen Startseite weitergeleitet und über den Erfolg des Logins informiert

Erweiterungen:

2a) Benutzer hat keine Daten
2a1) Benutzer geht zur Verwaltung und meldet sich an / holt sich Daten

4a) Benutzerdaten sind nicht korrekt
4a1) System meldet Fehler an Benutzer und geht zurück zur Startseite

4b) Benutzer ist gesperrt
4b1) Benutzer wird über Sperre informiert

Essen auswählen

Hauptakteur:	Mensagast	
Anwendungsbereich:	Mensasystem	
Niveau:	Benutzerziel	
Beteiligte und Interessen:	Mensagast	möchte Essen bestellen um dies am Ausgabetag abzuholen
	Caterer	möchte Vorbestellungen haben um Essen anzuliefern
Voraussetzung:	Mensagast angemeldet Speiseplan eingetragen	
Mindestzusicherung:	Mensagast sieht Speiseplan	

Zusicherung im Erfolgsfall:

Bestellung ist im System gespeichert und für Caterer und Mensagast sichtbar

Haupt-Erfolgsszenario:

1.) Mensagast navigiert zum Speiseplan
2.) System zeigt den Speiseplan mit den für den Mensagast momentan bestellbaren Speisen
3.) Mensagast wählt sein Menü aus
4.) Mensagast schickt Bestellung ab
5.) Speiseplan meldet Erfolg der Bestellung und zeigt diese im Speiseplan an

Erweiterungen:

2a) Das System zeigt einen leeren Speiseplan, da es nichts vorzubestellen gibt
2a1) Mensagast bricht die Aktion ab

Essen abbestellen

Hauptakteur:	Mensagast	
Anwendungsbereich:	Mensasystem	
Niveau:	Benutzerziel	
Beteiligte und Interessen:	Mensagast	möchte Essen abbestellen, da er es nun doch nicht Essen möchte
	Caterer	möchte Vorbestellungen haben um Essen anzuliefern und muss sich auf Bestellungen verlassen können
	Mensasystem	sichert Caterer Rahmenbedingungen zu und setzt dieses gegenüber dem Mensagast durch
Voraussetzung:	Mensagast angemeldet Speiseplan eingetragen	
Mindestzusicherung:	Mensagast sieht Speiseplan und seine Bestellungen	

Zusicherung im Erfolgsfall:

Bestellung ist aus dem System gelöscht und für den Caterer nicht mehr sichtbar
Bestellung wird dem Mensagast nicht berechnet

Haupt-Erfolgsszenario:

1. Mensagast navigiert zum Speiseplan
2. Mensagast wählt Bestellungen aus, die er zurücknehmen möchte
3. Mensagast schickt Löschwunsch ab
4. System löscht Bestellung aus dem System
5. System meldet Erfolg
6. Mensagast sieht den Speiseplan ohne seine gelöschten Bestellungen

Erweiterungen:

2a) Mensasystem zeigt keine Bestellungen an, da Mensagast keine abgegeben hat
2a1) Mensagast bricht Aktion ab

3a) Mensasystem verweigert Löschvorgang, da dies zu kurzfristig ist
3a1) Mensasystem informiert den Mensagast darüber

Rechnung einsehen

Hauptakteur: Mensagast
Anwendungsbereich: Mensasystem
Niveau: Benutzerziel
Beteiligte und Interessen: Mensagast möchte kontrollieren welche Speisen ihm in Rechnung gestellt werden
Mensasystem speichert Bestellungen und Abrechnung
Voraussetzung: Mensagast angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall:
Mensagast sieht welche seiner Bestellungen ihm in Rechnung gestellt wurden
Haupt-Erfolgsszenario:

1. Mensagast navigiert zu den Rechnungen
2. Mensagast wählt eine der vorhandenen Rechnungen aus
3. Mensasystem zeigt eine Liste der Bestellungen, die zu dieser Rechnung beigetragen haben an
4. Mensagast kann Zusammensetzung des Gesamtbetrags nachvollziehen

Erweiterungen:
2a) Mensasystem zeigt keine Rechnungen, wenn keiner vorhanden sind

Speiseplan einstellen

Hauptakteur:	Caterer	
Anwendungsbereich:	Mensasystem	
Niveau:	Benutzerziel	
Beteiligte und Interessen:	Caterer	benötigt Information über Anzahl der Essen die geliefert werden müssen. Dazu stellt er den Speiseplan online um Bestellungen zu ermöglichen.
	Mensagast	benötigt Information über Speisen
Voraussetzung:	Caterer angemeldet	
Mindestzusicherung:	---	

Zusicherung im Erfolgsfall:

Speiseplan ist im System gespeichert und Mensagast kann Bestellungen aufgeben

Haupt-Erfolgsszenario:

1. Caterer navigiert zum Speiseplan
2. Caterer trägt Speise ein
3. Caterer schickt Formular ab
4. Mensasystem speichert Speise im Plan
5. Speise kann von Mensagästen bestellt werden

Erweiterungen:

2a) Caterer lädt Liste von Speisen in das System
2a1) Mensasystem trägt alle Speisen ein

4a) Mensasystem meldet Fehler im Formular
4a1) Mensasystem leitet Caterer zum Speiseplan zurück

Bestellungen einsehen

Hauptakteur:	Caterer	
Anwendungsbereich:	Mensasystem	
Niveau:	Benutzerziel	
Beteiligte und Interessen:	Caterer	benötigt Information über Anzahl der Essen die geliefert werden müssen.
	Mensagast	möchte sein vorbestelltes Essen geliefert bekommen
	Verwalter	muss den Caterer für die Lieferung bezahlen und möchte dafür die Mensagäste belasten
Voraussetzung:	Caterer angemeldet	
Mindestzusicherung:	---	

Zusicherung im Erfolgsfall:

Caterer erfährt die Anzahl der Bestellungen zu diesem Zeitpunkt.

Haupt-Erfolgsszenario:

1. Caterer navigiert zu den Bestellungenlisten
2. Caterer wählt Ausgabetag aus für den er die Bestellungen sehen will
3. Mensasystem zeigt eine Liste mit allen Speisen und der Anzahl der Bestellungen für diese an
4. Caterer liefert diese Anzahl und gibt sie aus

Erweiterungen:

3a) Mensasystem zeigt eine Liste mit allen Speisen sowie verschiedenen Ausgabeorten und oder Uhrzeiten und wie viele Bestellungen für diese Teilbereiche getätigt wurden

Benutzer anlegen

Hauptakteur: Administrator
Anwendungsbereich: Schulverwaltungssystem
Niveau: Benutzerziel
Beteiligte und Interessen: Administrator möchte einem neuen Benutzer ermöglichen im System Funktionen auszuführen

Voraussetzung: Administrator angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall:
Benutzer ist im System vorhanden und kann sich anmelden

Haupt-Erfolgsszenario:

1. Administrator navigiert zur Benutzerverwaltung
2. Administrator trägt die Daten des neuen Benutzers ein
3. Administrator wählt aus in welchen Gruppen der neue Benutzer Mitglied ist
4. Administrator sendet die Daten ab
5. Schulverwaltungssystem validiert die Daten und legt den Benutzer an
6. Schulverwaltungssystem meldet Erfolg an den Administrator

Varianten:

2a) Administrator lädt eine Liste von Benutzern in System
2a1) Schulverwaltungssystem legt jeden Benutzer einzeln an

6a) Schulverwaltungssystem erstellt eine Ergebnisdatei mit dem Passwort des Benutzers
6a1) Administrator lädt Ergebnisdatei herunter
6a2) Administrator informiert Benutzer über sein Passwort

Erweiterungen:

5a) Schulverwaltungssystem kann die Daten nicht validieren
5a1) Schulverwaltungssystem informiert Administrator über Fehler in der Eingabe
5a2) Administrator geht zurück zu Schritt 2

Gruppe anlegen

Hauptakteur: Administrator
Anwendungsbereich: Schulverwaltungssystem
Niveau: Benutzerziel
Beteiligte und Interessen: Administrator möchte Benutzer in Gruppen sammeln um für diese Rechte zu setzen oder diese gemeinsam zu verwalten

Voraussetzung: Administrator angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall:
Neue Gruppe ist im System vorhanden
Benutzer können zu dieser Gruppe hinzugefügt werden
Gruppe kann Rechte bekommen

Haupt-Erfolgsszenario:

1. Administrator navigiert zur Gruppenverwaltung
2. Administrator trägt neuen Gruppennamen ein
3. Administrator sendet Formular ab
4. Schulverwaltungssystem validiert die Einmaligkeit des Gruppennamens und erzeugt diese
5. Schulverwaltungssystem informiert Administrator über Erfolg
6. Administrator kann Benutzer dieser Gruppe hinzufügen

Erweiterungen:
4a) Schulverwaltungssystem entdeckt vorhandene Gruppe mit demselben Namen
4a1) Administrator wird informiert und kann einen anderen Gruppennamen eintragen

Berechtigung setzen

Hauptakteur: Administrator
Anwendungsbereich: Schulverwaltungssystem
Niveau: Benutzerziel
Beteiligte und Interessen: Administrator möchte Benutzern und Gruppen ermöglichen gewisse Funktionen auszuführen und andere Benutzer davon abhalten

Voraussetzung: Administrator angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall:
Alle Mitglieder der Gruppe, bzw. der ausgewählte Benutzer können alle Funktionen ausführen, für die das Recht benötigt wird

Haupt-Erfolgsszenario:

1. Administrator navigiert zur Rechteverwaltung
2. Administrator wählt eine Gruppe oder einen Benutzer aus
3. Schulverwaltungssystem zeigt eine Liste mit allen Rechten an und ob der Benutzer/die Gruppe diese inne hat und/oder diese verteilen kann
4. Administrator verändert die Berechtigungen der Benutzer/Gruppe
5. Administrator sendet das Formular ab
6. Schulverwaltungssystem speichert die neue Rechtezuteilung
7. Benutzer können nun mit diesen Rechten arbeiten

Erweiterungen:
6a) Administrator ist nicht berechtigt ein Recht zu verteilen, welches er verändern will
6a1) Schulverwaltungssystem meldet ihm diesen Fehler und verändert die Rechte nicht

Essen Abrechnen

Hauptakteur:	Verwalter	
Anwendungsbereich:	Mensasystem	
Niveau:	Benutzerziel	
Beteiligte und Interessen:	Verwalter	möchte von den Mensagästen Geld für die bestellten Essen bekommen um den Caterer zu bezahlen
	Caterer	möchte alle Essen bezahlt bekommen
	Mensagast	möchte wissen wofür er bezahlt und nur bezahlen was er bestellt hat

Voraussetzung: Verwalter angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall:
Verwalter hat Lastschriften, mit denen er die Beträge vom Konto der Mensagäste einziehen lassen kann
Mensagast kann sich über Zusammensetzung der Rechnung informieren

Haupt-Erfolgsszenario:

1. Verwalter navigiert zur Essensabrechnung
2. Verwalter wählt Zeitraum für den abgerechnet wird und welche Benutzer davon betroffen sein sollen
3. Verwalter füllt einen aussagekräftigen Verwendungszweck aus
4. Verwalter stellt ein auf welches Konto der Betrag überwiesen werden soll
5. Verwalter schickt das Formular ab
6. Mensasystem rechnet die Bestellungen für die Benutzer zusammen und erstellt Rechnungen für jeden Benutzer
7. Mensasystem meldet Erfolg der Aktion
8. Verwalter kann diese Rechnungen herunterladen
9. Mensagast kann seine Rechnung einsehen

Erweiterungen: ---

Für Mensagast bestellen

Hauptakteur:	Verwalter	
Anwendungsbereich:	Mensasystem	
Niveau:	Benutzerziel	
Beteiligte und Interessen:	Verwalter	muss Mensagast unterstützen, wenn dieser sich nicht selbst anmelden kann. (Internetausfall, Krankheit, …)
	Mensagast	möchte eine Bestellung aufgeben, kann dies aber selbst nicht tun

Voraussetzung: Verwalter angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall:
Bestellung für Mensagast ist im System gespeichert
Haupt-Erfolgsszenario:

1. Verwalter navigiert zum „Bestellen für Mensagast“
2. Verwalter wählt Datum aus für das bestellt werden soll
3. Verwalter wählt Mensagast aus für den bestellt werden soll
4. Verwalter sendet Formular
5. Mensasystem zeigt Speiseplan für diesen Mensagast an diesem Tag
6. Verwalter wählt Speise(n) aus und wählt Uhrzeit und Ausgabeort wenn gefragt
7. Verwalter sendet Formular ab
8. Mensasystem speichert Bestellung für den ausgewählten Mensagast
9. Mensasystem bestätigt Verwalter Erfolg der Aktion

Erweiterungen: ---

Benutzer sperren

Hauptakteur: Verwalter
Anwendungsbereich: Schulverwaltungssystem
Niveau: Benutzerziel
Beteiligte und Interessen: Verwalter möchte Zugriff des Benutzers auf das System verhindern

Voraussetzung: Verwalter angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall:
Benutzer kann sich nicht mehr am System anmelden
Benutzer wird beim Anmeldeversuch über Sperrung informiert
Haupt-Erfolgsszenario:

1. Verwalter geht zum Benutzer sperren
2. Verwalter wählt Benutzer aus, den er sperren möchte
3. Verwalter sendet Formular ab
4. Schulverwaltungssystem speichert Sperre des Benutzers
5. Schulverwaltungssystem meldet Verwalter Erfolg der Sperrung des Benutzers
6. Benutzer kann sich nicht mehr anmelden

Erweiterungen: ---

Mensasystem konfigurieren

Hauptakteur:	Verwalter	
Anwendungsbereich:	Mensasystem	
Niveau:	Benutzerziel	
Beteiligte und Interessen:	Verwalter	möchte das Mensasystem so konfigurieren, dass der Caterer den Speiseplan eintragen kann und die notwendigen Informationen zum Essen liefern erhält
	Caterer	möchte das System mit möglichst wenig Aufwand benutzen
Voraussetzung:	Verwalter angemeldet	
Mindestzusicherung:	---	
Zusicherung im Erfolgsfall:	Konfiguration des Mensasystems ist gespeichert und kann vom Caterer benutzt werden	

Haupt-Erfolgsszenario:

1. Verwalter geht zur Mensasystemverwaltung
2. Verwalter wählt welche Einstellung er anpassen will (Kategorien, Schichten, Ausgabeorte)
3. Mensasystem zeigt vorhandene Einstellungen an
4. Verwalter ändert aktuelle Einstellungen oder fügt neue hinzu
5. Verwalter sendet Formular mit den Änderungen
6. Mensasystem speichert diese Änderungen
7. Mensasystem zeigt Erfolg und neue Einstellungen an

Erweiterungen: ---

Mensasystem konfigurieren

Hauptakteur: Verwalter
Anwendungsbereich: Mensasystem
Niveau: Benutzerziel
Beteiligte und Interessen:
- Verwalter: möchte das Mensasystem so konfigurieren, dass der Caterer den Speiseplan eintragen kann und die notwendigen Informationen zum Essen liefern erhält
- Caterer: möchte das System mit möglichst wenig Aufwand benutzen

Voraussetzung: Verwalter angemeldet
Mindestzusicherung: ---
Zusicherung im Erfolgsfall:
Konfiguration des Mensasystems ist gespeichert und kann vom Caterer benutzt werden
Haupt-Erfolgsszenario:

8. Verwalter geht zur Mensasystemverwaltung
9. Verwalter wählt welche Einstellung er anpassen will (Kategorien, Schichten, Ausgabeorte)
10. Mensasystem zeigt vorhandene Einstellungen an
11. Verwalter ändert aktuelle Einstellungen oder fügt neue hinzu
12. Verwalter sendet Formular mit den Änderungen
13. Mensasystem speichert diese Änderungen
14. Mensasystem zeigt Erfolg und neue Einstellungen an

Erweiterungen: ---

4.5.3. Laufzeitmessungen

Aufgabe	Schnitt	Zeit 1	Zeit 2	Zeit 3	Zeit 4	Zeit 5
startseite	0,14259	0,14145	0,14330	0,14409	0,14462	0,13947
anmelden	0,26527	0,33014	0,39971	0,19863	0,19943	0,19842
benutzer eintragen	0,24167	0,27660	0,23856	0,23162	0,23540	0,22616
sp eintragen	0,21666	0,26353	0,20484	0,20565	0,20902	0,20027
sp bestellen	0,22180	0,24948	0,21853	0,20539	0,22495	0,21064

Abbildung 22: Zeitmessung am 23.02.2010 auf 85.214.103.233/esv-stable

Aufgabe	Schnitt	Zeit 1	Zeit 2	Zeit 3	Zeit 4	Zeit 5
startseite	0,02058	0,02315	0,02090	0,02006	0,01944	0,01935
anmelden	0,03174	0,03300	0,03161	0,03183	0,03118	0,03110
benutzer ein	0,07489	0,06627	0,08777	0,06710	0,08598	0,06736
sp ein	0,43720	0,25030	0,31642	0,56156	0,39648	0,66126
sp bestellen	0,60782	0,42517	0,47702	0,71398	0,70765	0,71530

Abbildung 23: Zeitmessung am 13.04.2010 auf 85.214.103.233 ror-esv revision 28