

# Dokumentation für die Übungsaufgabe der Vorlesung Verteilte Systeme an der HS-Mannheim im SS2011

Kay Estelmann,  
Tobias Schröder,  
Thorsten Reitz,  
Thomas Zimmermann,  
Christoph Reiser,  
Jori Kern

Fakultät für Informatik  
Hochschule Mannheim  
Paul-Wittsack-Straße 10  
68163 Mannheim

**Zusammenfassung** Im Rahmen der Vorlesung Verteilte Systeme ist eine Implementierung des Spiels Buzzword-Bingo, oder umgangssprachlich auch Bullshit-Bingo, zu erstellen. Ein klassisches Beispiel für verteilte Systeme ist eine Webanwendung. Unsere Gruppe hat das Spiel als Webanwendung konzipiert. Dabei werden bewährte Technologien wie PHP, AJAX und eine MySQL Datenbank verwendet.

Im ersten Kapitel werden Spielregeln und Hintergründe zum Spiel Buzzword-Bingo erläutert. Kapitel 2 betrachtet die Entscheidungsfindung, warum eine Webanwendung eingesetzt wird. Danach wird auf den Aufbau des Systems und die Herausforderungen hinsichtlich der eingesetzten Technologien eingegangen. Der Aufbau der Klassen wird in Kapitel 3 erläutert. Anschließend wird in der Laufzeitsicht das Einloggen eines Spielers in ein bestehendes Spiel, sowie das Anlegen eines neuen Spiels näher betrachtet. Im Weiteren wird die eingesetzte Technologie sowie der Verlauf des Projekts evaluiert.

## 1 Aufgabenstellung

Aufgabe ist es, das genannte Buzzword-Bingo als verteilte Anwendung mittels Client und Server zu implementieren. Dabei ist den einzelnen Übungsgruppen offengelassen, wie genau das Spiel implementiert wird. Ob als natives Programm, Webanwendung oder App für ein Smartphone spielt dabei keine Rolle. Ebenso besteht Gestaltungsfreiraum, ob zum Beispiel der Client, der ein neues Spiel eröffnet, auch gleichzeitig der Server ist.

Die einzigen Anforderungen sind:

- Über den Client soll ein Spieler ein neues Spiel erstellen, oder einem bestehenden Spiel beitreten können
- Wird ein neues Spiel erstellt, soll der Spieler die Größe des Spielfelds auswählen können
- Der Spieler/Client, der ein Spiel eröffnet hat, steuert den Server
- Spieler sollen über ein Netzwerk miteinander spielen können
- Im Client soll man eine Liste der Mitspieler einsehen können
- Hat ein Spieler gewonnen, werden die Mitspieler benachrichtigt

Des Weiteren ist eine Dokumentation und eine Präsentation zur Implementierung zu erstellen.

## 2 Was ist Buzzword-Bingo

Die Erklärung von Wikipedia umfasst bereits alle Hintergründe und die wichtigsten Spielregeln des Buzzword-Bingo.

Buzzword-Bingo, in der späteren Verbreitung auch Bullshit-Bingo genannt, ist eine humoristische Variante des Bingo-Spiels, die die oft inhaltslose Verwendung von zahlreichen Schlagwörtern in Vorträgen, Präsentationen oder Besprechungen persifliert.

Statt Bingokarten mit Zahlen werden Karten mit Schlagwörtern (engl. buzzwords) benutzt. Im Gegensatz zum originalen Bingo, bei welchem die zu streichenden Zahlen aus einer Lostrommel gezogen werden, werden Wörter gestrichen, wenn sie genannt werden. Bei einer vollständig gefüllten Reihe, Spalte oder Diagonale soll der Spieler den Regeln nach aufstehen und (statt "Bingo") lautstark "Bullshit" (dt. Quatsch, Schwachsinn) rufen. Mit dem Spiel und diesem Ausruf wird gleichzeitig die übermäßige Verwendung von oft inhaltsleeren Schlagwörtern kritisiert. [1]

## 3 Projektorganisation

### 3.1 Software Entwicklungsprozess

In der Prozessauswahl haben wir uns in Anbetracht des terminlichen Drucks für einen agilen Prozess entschieden. Wir haben uns hierbei an Extreme Programming (XP) angelehnt. Auf Grund des angewandten Prozesses gab es keine Aufteilung in Rollen im klassischen Sinn. Durch die zeitliche Vorgabe haben wir weniger Ressourcen für die Dokumentation aufwenden müssen als auf die Umsetzung, wirklich dokumentiert festgehalten wurde zu Beginn nur die grobe Struktur. Einige Abläufe des Spiels mussten allerdings ausformuliert werden, da sie für die Implementierung wichtig waren. Das bedeutete aber auch, dass wir enger zusammenarbeiten mussten, was sich durch tägliche Treffen bewältigen lies und dadurch, dass alle wichtigen Entscheidungen mit dem gesamten Team getroffen wurden. Somit hatte jeder den gleichen Informationsstand. Auch innerhalb der Entwicklung hielten wir uns an XP und haben meist in Pair Programming entwickelt, was sich im Nachhinein als sehr wichtig herausstellte (siehe Kapitel Projektreview). Hierdurch lies sich das Team auch entsprechend der Kompetenzen und technologischen Fähigkeiten aufteilen. Die Entwicklung zielte gemäß XP auf lauffähige Teilkomponenten ab, was in unserem Fall in Form von Skripten geschah. Das Paradigma des Testdriven Development lies sich auf Grund der gewählten Technologien nicht einhalten, da keine Testroutinen für diese geschrieben werden können. Einzig Testfälle konnten spezifiziert werden, die durch Try and Error (siehe hierzu auch Kapitel Projektreview) ausgeführt wurden.

## Phasen der Entwicklung

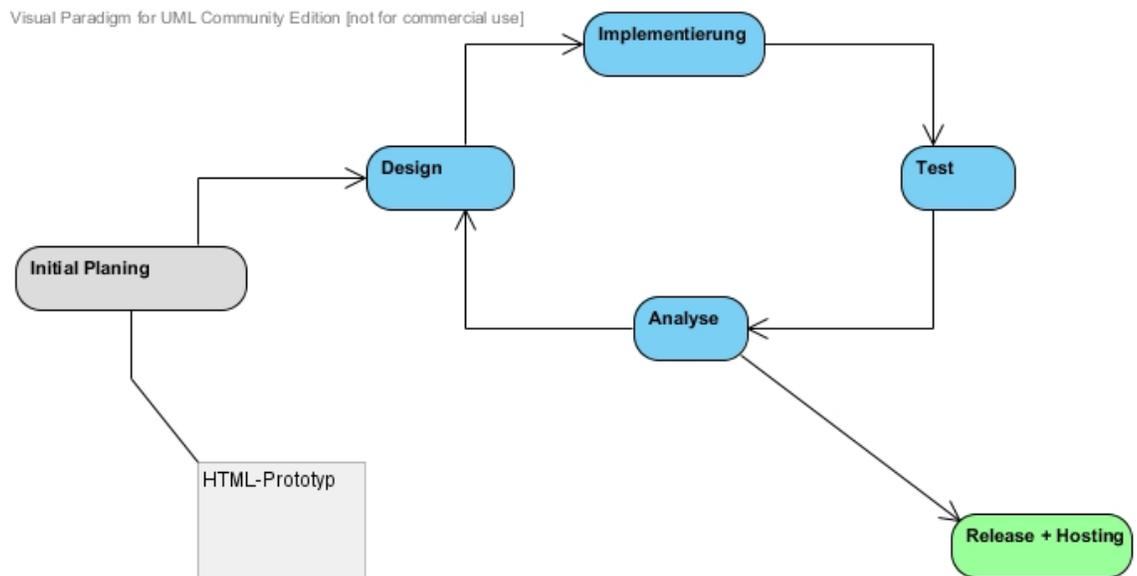


Abbildung 1. Grobes Schema der Prozessphasen

Der Implementierung ging eine kurze Phase der Anforderungsanalyse und des groben Designs voraus, aus der ein Prototyp der GUI entstand, auf dem die weitere Entwicklung beruhte. Dieser war für die Abmachungen zwischen den Scriptsprachen von enormer Bedeutung. In der Phase von Implementierung und Test wurde in mehreren Iterationen die Bestandteile inkrementell entwickelt. (Siehe Abbildung 1)

### 3.2 Konfigurationsmanagement

In diesem Abschnitt werden die Abmachungen hinsichtlich der eingesetzten Technologien beschrieben.

#### IDE

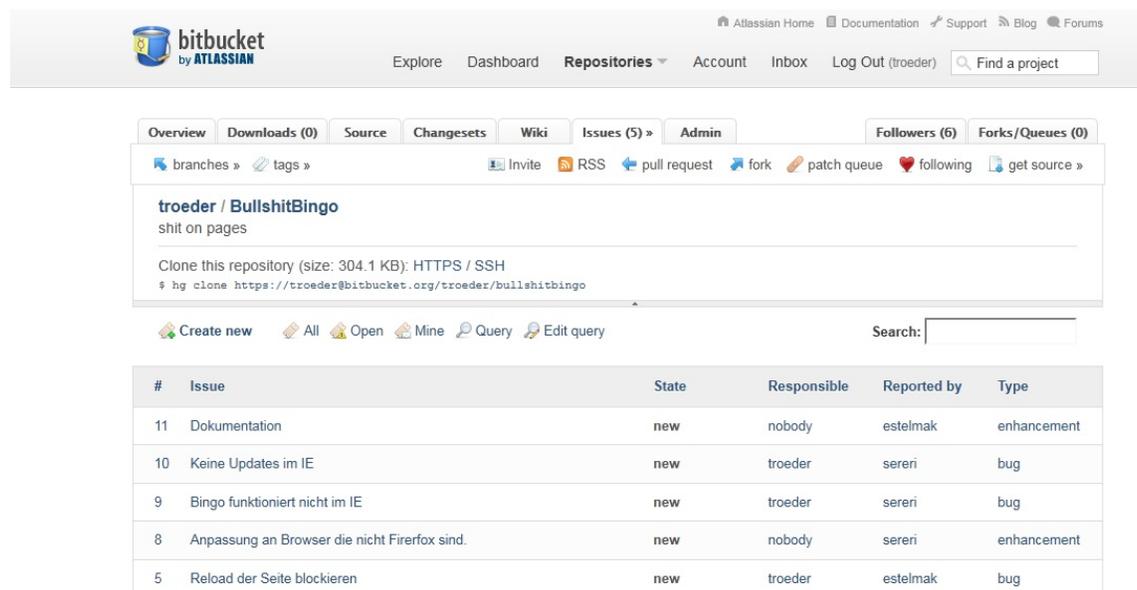
Für die Implementierung haben wir uns für die Verwendung von einer einheitlichen IDE entschieden und uns auf Netbeans [2] festgelegt, weil sich dadurch auch die bereits integrierte Versionsverwaltung nutzen lies.

## Versionsverwaltung

Als Versionsverwaltung wurde Mercurial verwendet. Mercurial ist, ähnlich Git und im Gegensatz zu Subversion, eine verteilte Versionsverwaltung mit einem lokalen und einem zentralen Repository. Das zentrale Repository hatten wir beim freien Anbieter bitbucket.org [3] angelegt. Die Kombination aus Netbeans und Mercurial ist bei den meisten Mitgliedern der Gruppe schon länger im Einsatz.

## Issue Tracker und Wiki

Bitbucket bietet den Nutzern zu jedem Repository ein eigenes Wiki und einen eigenen Issue Tracker (Abbildung 2) an, welche auch zum Einsatz kamen. Alle Definitionen, Anleitungen etc. wurden im Wiki vermerkt und sind dort nachzulesen. Es kann als eine Art Projekthandbuch angesehen werden, wenn auch nicht in der ausführlichen Form eines Handbuchs. Der Issue Tracker funktioniert ähnlich einem Bug Tracker, allerdings kann dem Report noch ein Typ zugewiesen werden. Wird ein Issue erstellt, kann dieser mit einem Kommentar versehen werden und eine Zuständigkeit für diesen angegeben werden. Die Meldung geht dann trotzdem an alle Beteiligten des Projekts per Email. Dieses Feature wurde von uns genutzt um auch einen Überblick über das zu haben, was noch zu tun war bzw. noch nicht funktionierte.



The screenshot shows the Bitbucket web interface for a repository named 'troeder / BullshitBingo'. The repository description is 'shit on pages' and it has a size of 304.1 KB. Below the description, there are options to 'Create new', 'All', 'Open', 'Mine', 'Query', and 'Edit query'. A search bar is also present. The main part of the interface is a table of issues.

#	Issue	State	Responsible	Reported by	Type
11	Dokumentation	new	nobody	estelmak	enhancement
10	Keine Updates im IE	new	troeder	sereri	bug
9	Bingo funktioniert nicht im IE	new	troeder	sereri	bug
8	Anpassung an Browser die nicht Firerfox sind.	new	nobody	sereri	enhancement
5	Reload der Seite blockieren	new	troeder	estelmak	bug

Abbildung 2. zeigt die Übersichtsseite des Issue Trackers

## **Zielpattform**

Da Buzzword Bingo mit HTML, PHP und Javascript von uns umgesetzt wurde, musste ein Server für die Entwicklungsphase gewählt werden. Die Entscheidung fiel auf einen jeweils lokal genutzten Apache Webserver, welcher im Paket von XAMPP [4] frei zur Verfügung steht. Es wurde in erster Linie für Firefox entwickelt, hier wurde das Debugging Tool FireBug [5] verwendet.

## **4 Lösung**

### **4.1 Technologieauswahl**

Schon kurz nach Erteilung der Aufgabe war uns klar, dass wir auf Websprachen als Technologie setzen würden. Dies lag zum einen an der Begrenztheit unseres Horizonts bezüglich Programmiersprachen, zum anderen an der vermeintlichen Einfachheit der Webtechnologien im Bereich der Server-Client Programmierung. So waren C++ und ähnliche Sprachen auf Grund unserer mangelnden Erfahrungen mit den Sprachen schnell ausgeschieden. Bei Java hielten wir die Server/Client Programmierung für verhältnismäßig kompliziert. Vor allem weil dies bei den Webtechnologien entfiel, fiel unsere Wahl auf eine Mischung aus PHP mit MySQL Anbindung, CSS gestyltem HTML und JavaScript. Ein weiterer Grund war eine hohe Plattformunabhängigkeit ohne große Anpassungen.

### **4.2 Technische Herausforderungen**

Das Problem bei der Programmierung interaktiver Inhalte mit PHP ist, dass der Datentransfer nur aus einer Richtung initiiert werden kann, durch eine Anfrage des Clients. Aktualisierungen des Spiels wie eine neue Runde können also nicht vom Server an die Spieler mitgeteilt werden, sondern müssen durch die Clients in einem festgelegten Intervall beim Server erfragt werden.

Ein weiterer Punkt ist die Verschiedenheit der Systeme, nur hier nicht in Bezug auf Betriebssysteme, sondern auf die verschiedenen Browser. Zwar wurden Standards für die Darstellung von HTML und die Verarbeitung von JavaScript schon vor über 10 Jahren verabschiedet, trotzdem verhalten sich nicht alle Browser gleich oder stellen den selben Quellcode gleich dar. Zwar ändert sich das mit den allerneuesten Versionen der Webbrowser, diese sind allerdings leider nicht als weitverbreitet zu betrachten.

### **4.3 Vor- und Nachteile**

Es gibt verschiedene Vorteile beim Zurückgreifen auf PHP & Co als Programmierumgebung für dieses Projekt. Ein Punkt wäre wie schon erwähnt das Entfallen der Serverentwicklung, da die Server - Client Kommunikation bereits über das

HTTP Protokoll gegeben ist. Ein reines Ablegen der Dateien auf einem Server der PHP unterstützt reicht somit aus um Daten auszutauschen. Der Lernaufwand innerhalb des Teams war eher gering. Die meisten waren mit PHP bereits vertraut, HTML, CSS und SQL konnten bereits alle. Nur der Kenntnisstand in Hinsicht auf JavaScript war im Team nicht ausgeprägt, reichte allerdings aus um das Projekt zufriedenstellend abzuschließen.

Durch die Verwendung von HTML als Basis war jede Plattform, für die ein Webbrowser existiert mit dem Projekt kompatibel, zumindest in der Theorie. Denn hier beginnen die Nachteile bereits. Zwar ist heutzutage die Darstellung in allen Browsern ungefähr die gleiche, doch schon bei der Gestaltung können minimale Unterschiede auftreten, die dann allerdings wiederum nicht unerhebliche Probleme bereiten. Bei der Java- bzw. ECMAScript Unterstützung auf der anderen Seite sind die Unterschiede noch immer erheblich und führen teilweise wie im Falle des Internet Explorer (sogar in der aktuellsten Version 8) zu spielverhindernden Fehlern.

Die Webarchitektur des Spiels selbst erzeugte aber auch Probleme ihrerseits. Durch die Verwendung des nur in eine Richtung sendenden PHP musste clientseitig JavaScript eingesetzt werden um Rückmeldungen zu senden und Teile der Seite zu aktualisieren ohne die Seite im Gesamten neu zu laden und somit den Spielstand zu verlieren oder andere Probleme zu provozieren. Auch die Angelegenheit des Schließens des Fensters stellte eine komplizierte Aufgabe dar. Das Testen der Funktionen und später des Spiels auf Fehler war ohnehin eine schwierige Sache. Oft war nicht einmal sofort klar in welcher Programmiersprache überhaupt das Problem lag, da Fehler in Browsern meist nur versteckt wenn überhaupt ausgegeben werden, ein Debug-Modus nicht vorhanden ist. Die einzigen Abhilfen in solchen Fällen war die Ausgabe von Informationsstrings, das Firefox Add-On Firebug und der logische Verstand des Programmierers.

Generell würden wir uns, dieselbe Aufgabe noch einmal gegeben, wohl eher nicht mehr für die Verwendung von 5 verschiedenen Sprachen gleichzeitig entscheiden, vor allem wenn unsere Kenntnisstände vollkommen unterschiedlich sind so wie sie es hier waren.

#### **4.4 Andere Technologien**

Außer per Webtechnologie hätte das Projekt auch mit anderen Technologien bzw. Programmiersprachen umgesetzt werden können. Im Folgenden werden einige von diesen beschrieben und erklärt warum wir sie nicht eingesetzt haben

**C++ / C#** Die Verwendung hätte eine Verwendung auf vielen Desktop Betriebssystemen wie Windows, OSX und Linux durch den Einsatz verschiedener Compiler erlaubt, da alle Teammitglieder noch nicht mit diesen Technologien gearbeitet haben, war dies in Anbetracht der kurzen Zeit für die Umsetzung keine Alternative. Auch die Verwendung auf Mobiltelefonen wäre wohl nicht möglich gewesen.

**Java** Der Einsatz von Java wäre wohl noch die beste Alternative gewesen. Dafür sprachen unser Kenntnisstand und die relative Einfachheit der Programmierung. Allerdings wäre eine JVM auf dem ausführenden Gerät benötigt worden, was wiederum die Anzahl möglicher Spieler verringert und einige Mobiltelefone ausschließt. Im Gegensatz zu den verwendeten Webtechnologien hätte hier wie bei C++/C# ein Server programmiert werden müssen, was auf Grund des Zeitlimits ein weiterer Grund dagegen war.

**Mobile Plattformen** Mobiltelefone waren nur sehr kurz im Gespräch, zum einen wegen der noch immer recht geringen Verbreitung von sogenannten Smartphones, die die nötige Auflösung bieten würden. Zum anderen verwendet jede Plattform unterschiedliche Programmiersprachen oder Architekturen, eine einheitliche Programmierungsumgebung ist also nicht gegeben. Was ein weiterer positiver Nebeneffekt der Webtechnologie ist. Damit kann das Spiel mittels passenden Stylesheets schnell an mobile Plattformen angepasst werden.

**AIR Applikation** Die Adobe AIR Umgebung wäre auch plattformunabhängig gewesen und hätte sich an der Webtechnologie orientiert. Jedoch kam dies auf Grund des Zeitmangels und dem Mangel an Know-How in diesem Bereich nicht zum Einsatz. Mittels der AIR Umgebung hätte das Problem der unterschiedlichen Interpretation in verschiedenen Browsern vermieden werden können.

## 5 Design und Topologie

Unsere Software implementiert ein webbasiertes Bingospiel welches mit mehreren Spielern spielbar ist. Da wir das Spiel in einem Webbrowser ausführen ist es weitestgehend plattformunabhängig. Im folgenden soll die Funktionsweise der Software näher beschrieben werden.

Beim Design der Software haben wir uns für eine Model-View-Controller Architektur entschieden. Dieses teilt unsere Software in 3 Hauptelemente. Unsere Motivation zur Verwendung dieses Entwurfsmusters liegt in der hohen Flexibilität und Übersichtlichkeit, welche durch die MVC Architektur begünstigt wird.

Das Model verwaltet die im Programm verwendeten Objekte und stellt deren Methoden zur Verfügung. Die Objekte in unserer Software beinhalten Informationen aus der Datenbank, in welcher Informationen zu laufenden Spielen gespeichert sind.

In der View wird die Ausgabe der Daten verwaltet. Hier befinden sich hauptsächlich die Javascripte und CSS Dateien. Das visuelle Design unserer Software wird in der View gesteuert. Unsere View teilt sich auf 3 Seiten auf.

Das sind: Startseite, Spielseite und eine Seite um das Spiel zu verlassen.

Der Controller fungiert als Vermittler zwischen View und Model. Zudem beinhaltet er einen Database Controller, welcher den Objekten ermöglicht auf die SQL Datenbank zuzugreifen. Über den sogenannten Caller läuft die AJAX Kommunikation zwischen Client (Webbrowser) und unserem Server. Hier werden in einem kurzen Intervall Requests vom Client gesendet, welche den aktuellen Punktestand abfragen. Im Falle eines Bingos wird überprüft ob der Spieler tatsächlich ein Bingo erzielt hat und dann wird ein Bingo-Request gesendet, welches dem Server signalisiert das ein Spieler ein Bingo erzielt hat und eine neue Runde gestartet werden soll.

Ausserhalb des MVC haben wir noch eine weitere Datei namens index.php. Diese stellt so etwas wie eine main dar, welche den Programmablauf startet. Die Index Seite wählt je nach gesetzten Variablen die passende View aus und dient als Einstiegspunkt für das Spiel. Ist keine Variable gesetzt, wird die Start-View geladen. Ist die Post Variable "login=1" gesetzt wird die Game-View geladen. Ist "login=0" gesetzt wird die Leave Seite angezeigt, um das Spiel zu verlassen.

## 5.1 Komponenten

### Models Game-Model

Das Game-Model beinhaltet alle Funktionen, die ein Spiel benötigt. Desweiteren wird mit dem Konstruktor ein neues Spiel in die Datenbank eingetragen, bzw. alle notwendigen Daten aus der Datenbank geladen. Das Game-Modell beinhaltet folgende public Attribute, die einfach ausgelesen und somit leicht in den Views verwendet werden können. Dass die Attribute public sind ist hier eher unkritisch zu sehen, da PHP als Serverscript läuft und somit der Benutzer keinen Einblick erhält.

- ID: Die eindeutige ID des Spiels in der Datenbank
- Name: Der Name den das Spiel vom Ersteller bekommen hat
- Players: Ein Array mit Datenbankobjekten der Spieler. Die Objekte werden automatisch von PHP erstellt und decken sich lediglich durch die Attribute mit unserem Spieler Model.
- Leader: Hier wird die ID des Spiel-Erstellers gespeichert, wenn dieser ein Spiel verlässt wird es beendet und aus der Datenbank gelöscht.
- Round: Hier wird die aktuelle Spielrunde gespeichert. Jedes mal wenn ein Spieler ein Bingo hat, wird die Runde erhöht.
- Gridsize: Mit Gridsize ist die Größe des Spielfeldes gemeint. Hierbei wird die Gesamtzahl an Feldern gespeichert und nicht die Anzahl Felder pro Reihe bzw. Spalte. Ist das Spielfeld also 5x5 groß, wird hier der Wert 25 gespeichert. Mit dieser Variable wird die benötigte Anzahl an Buzzwords aus der Datenbank geladen.

Das Model bietet folgende Funktionen:

- refreshPlayers(): Damit werden die aktuellen Spieler aus der Datenbank geladen und in das Players-Array geschrieben.
- increaseRound(gameID, playerId): Diese Funktion erhöht die Runde und den Punktestand beim entsprechenden Spieler. Dies ist eine statische Funktion, sodass sie einfach aus dem Caller aufgerufen werden kann. Deshalb wird als Parameter eine ID des Spiels benötigt, um beim richtigen Spiel die nächste Runde zu starten. Der Parameter playerId wird benötigt, um bei dem Spieler, der Bingo gedrückt, hat den Punktestand zu erhöhen.

### **Player-Model**

Das Player-Model beinhaltet alle Funktionen und Attribute zu einem einzelnen Spieler, desweiteren werden in den Attributen alle Variablen gespeichert, um sie einfach in den Views ausgeben zu können. Das Modell beinhaltet folgende public Attribute:

- ID: Eindeutige ID des Spielers in der Datenbank
- Name: Spielername, den sich der Spieler beim Einloggen gegeben hat
- Score: Aktueller Punktestand des Spielers. Gibt an, wie oft er schon ein Bingo hatte
- Game: ID des Spiels, an dem der Spieler teilnimmt.

Das Model bietet folgende Funktionen:

- setGame(GameID): Falls beim Erstellen des Spielers das Spieleobjekt noch nicht existiert, kann hiermit die SpielerID in die Datenbank geschrieben werden und somit der Spieler in das Spiel eingeloggt werden.
- delete(): Prüft ob ein Spieler der Ersteller eines Spiels ist, falls ja wird das Spiel ebenfalls aus der Datenbank gelöscht. In jedem Fall wird der Spieler aus der Datenbank gelöscht.
- checkName(Name, GameID): Statische Funktion, um zu überprüfen, ob der Name bereits in dem angegebenen Spiel existiert.

**View:** In den Views haben wir für einfacheres Arbeiten mit JavaScript das JQuery [6] Framework eingesetzt.

**Startseite:** Der HTML Code der Startseite wird von der start.php generiert. Dieser HTML Code wird zusätzlich von der start.css formatiert und gestaltet. Die Funktionen dieser Seite sind in der start.js implementiert. Die View beinhaltet das Loginformular, um ein neues Spiel zu erstellen bzw. sich in ein bestehendes Spiel einzuwählen. Wählt ein Spieler ein bestehendes Spiel aus der Liste, wird mittels AJAX-Request eine Liste aller Spieler mit dem aktuellen Punktestand geladen und angezeigt.

**Spielseite:** Auf der Spielseite wird das Spielfeld, sowie eine Spielerliste dargestellt. Generiert wird diese in der caller.php. Formatierungen sind in game.css hinterlegt. Das wichtigste JavaScript ist BullshIT.js, hier sind alle AJAX-Requests, die im Spiel benötigt werden, implementiert. Kommt ein Spieler in die Game View werden über diese die entsprechenden Models angesprochen, um ein Spiel bzw. den Spieler anzulegen. Mittels JavaScript wird überprüft, ob ein geklicktes Bingo richtig ist, falls ja wird der entsprechende Request gestartet. Die AJAX-Requests gehen immer an den Caller Controller.

**Leavesite:** Auf die Leave View kommt ein Spieler, wenn er das Spiel über den Button "Verlassen" beendet. Die Leave View veranlasst das Player Model dazu den Spieler zu löschen. Hier bekommt jeder Spieler seinen Punktestand beim Verlassen des Spiels angezeigt und wird persönlich verabschiedet. Da hier keine Interaktion mit dem Benutzer geschieht wird kein JavaScript benötigt.

**Controller Buzzword Controller:** Der Buzzword Controller liest die Schlagwörter für das Spielfeld aus der Datenbank und gibt diese zurück. Zur Kommunikation mit der Datenbank wird der Datenbank Controller verwendet.

**Caller:** Der Caller dient als Schnittstelle für die AJAX-Requests. Alle Requests gehen an diesen Controller. Mittels der gesetzten Variablen wird entschieden welche Aktion durchgeführt werden soll. Über den periodischen Request, der nur die SpieleID beinhaltet wird die Spielerliste ausgelesen, sowie die aktuelle Runde und die jeweiligen Punkte der Spieler. Diese werden als HTML zurückgegeben. Sind UserID, GameID und Bingo gesetzt, wird für den entsprechenden Spieler die Punktzahl sowie die Runde im Spiel erhöht. Die restlichen Spieler bekommen dann beim nächsten periodischen Request mitgeteilt, dass eine neue Runde gestartet ist und es werden neue Buzzwords geladen. Neue Buzzwords werden zurückgegeben, wenn die Variable getWords gesetzt ist.

**Database Controller:** Der Datenbank Controller dient zur einfacheren Kommunikation mit der MySQL Datenbank. Dies ist mit PHP schon recht leicht möglich, jedoch muss man sich immer um den Verbindungsauf- und abbau kümmern. Der Controller übernimmt diese Aufgabe. Er bietet 2 statische Methoden, denen jeweils nur die SQL Strings übergeben werden.

- returnQuery(SQL String) ist für SQL Abfragen gedacht, die ein Resultat zurückliefern. Die Funktion gibt ein Array von Objekten zurück. Die Objekte haben dann jeweils die Tabellenfelder als public Attribut, so dass diese leicht angesprochen werden können.
- runQuery(SQL String) ist für SQL Abfragen, die kein Ergebnis liefern wie Update oder Delete. Die Funktion gibt lediglich true oder false zurück, je nach dem ob die SQL Abfrage ohne Fehler durchlaufen wurde oder nicht.

Die Abfragen mussten in zwei Funktionen umgesetzt werden, da eine einzige Funktion bereit zu stellen, zu Fehlern bei Abfragen ohne Rückgabe führte, da Datensätze erwartet wurden.

## 5.2 Laufzeitsicht

In der Laufzeitsicht haben wir uns für das Einloggen eines Spielers in ein bestehendes Spiel (siehe Abbildung 3) und das Erstellen eines neuen Spiels (siehe Abbildung 4) als wichtige Prozesse entschieden.

Zunächst fordert der Client die Index Seite unseres Bingo an, in dem er die entsprechende URL in seinem Browser aufruft. Die Index Seite prüft, ob die Login Variable gesetzt ist. Da diese zum Anfang nicht existiert, wird die Start View zurückgegeben. Die Start View liest vor der Rückgabe noch die bestehenden Spiele aus der Datenbank aus und fügt diese in eine Liste ein. Beim User erscheint nun die Login Seite. Wird ein Spiel aus der Liste gewählt, so wird per Ajax ein Request an den Caller gesendet, der die eingeloggten Spieler, die Spielrunde, sowie die aktuellen Punktestände zurückliefern soll. Dazu fragt der Caller die Datenbank nach den benötigten Informationen ab. Das Ganze wird wie die Spielerliste als HTML zurück an den Client gesendet und unter dem Login Formular angezeigt. Dies kann sich beliebig oft wiederholen, solange der Benutzer ein Spiel aus der Liste wählt. Ist das Formular ausgefüllt und der Spieler klickt auf Los wird das Formular nach einer Überprüfung an die Index-Seite geschickt. Da nun die Login Variable gesetzt ist, wird die Game View geladen. Diese bringt das Spieler Modell dazu einen neuen Spieler zu erstellen. Das Model trägt alle Informationen in die Datenbank ein und ruft die ersten Buzzwords für den Spieler ab. Die Informationen stehen nun im Modell. Die Game View enthält auch alle Daten des Spiels, um eine Spielerliste anzuzeigen. An den Client wird nun das Spielfeld und die Spielerliste mit allen am Spiel beteiligten Usern und deren Punktestand angezeigt.

Zu Beginn fordert der Client die Index Seite an, in dem er die entsprechende URL in seinem Browser aufruft. Die Index Seite prüft, ob die Login Variable gesetzt ist. Da diese zum Anfang nicht existiert wird die Start View zurückgegeben. Die Start View fordert beim Laden eine Liste aktiver Spiele aus der Datenbank an. Diese werden als Liste für das Formular angezeigt. Der Benutzer lässt die Liste mit den Spielen unverändert auf Neues Spiel stehen, gibt stattdessen einen Namen für das Spiel an und wählt die Spielfeldgröße aus. Diese Daten werden an die Index-Seite nach einer Überprüfung gesendet. Diese prüft wiederum, ob die Login Variable gesetzt ist. Diesmal ist die Variable gesetzt und es wird die Game View geladen. Mit den Variablen Spielfeldgröße, Username und Spielname werden nun die entsprechenden Models angesprochen. Zunächst wird ein neuer Spieler ohne Spiel angelegt. Das Spieler Model trägt alle Spielerdaten in die Datenbank ein und ruft die ersten Buzzwords aus der Datenbank ab. Danach wird über das Game Model ein neues Spiel in die Datenbank eingetragen, die

Laufzeitsicht – User in aktivem Spiel anmelden

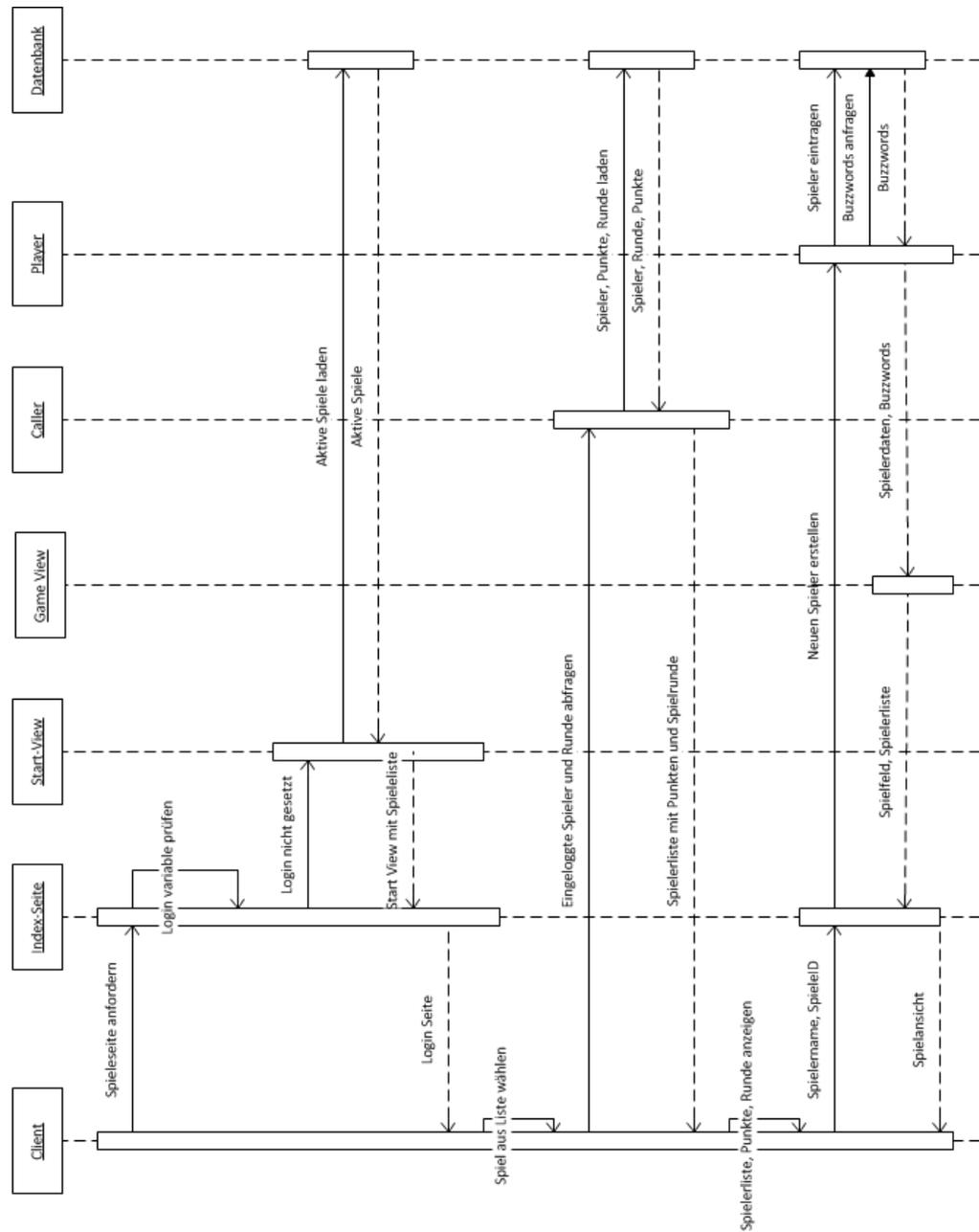


Abbildung 3. zeigt die Laufzeitsicht zum Einloggen in ein bestehendes Spiel

## Laufzeitsicht – Neues Spiel erstellen

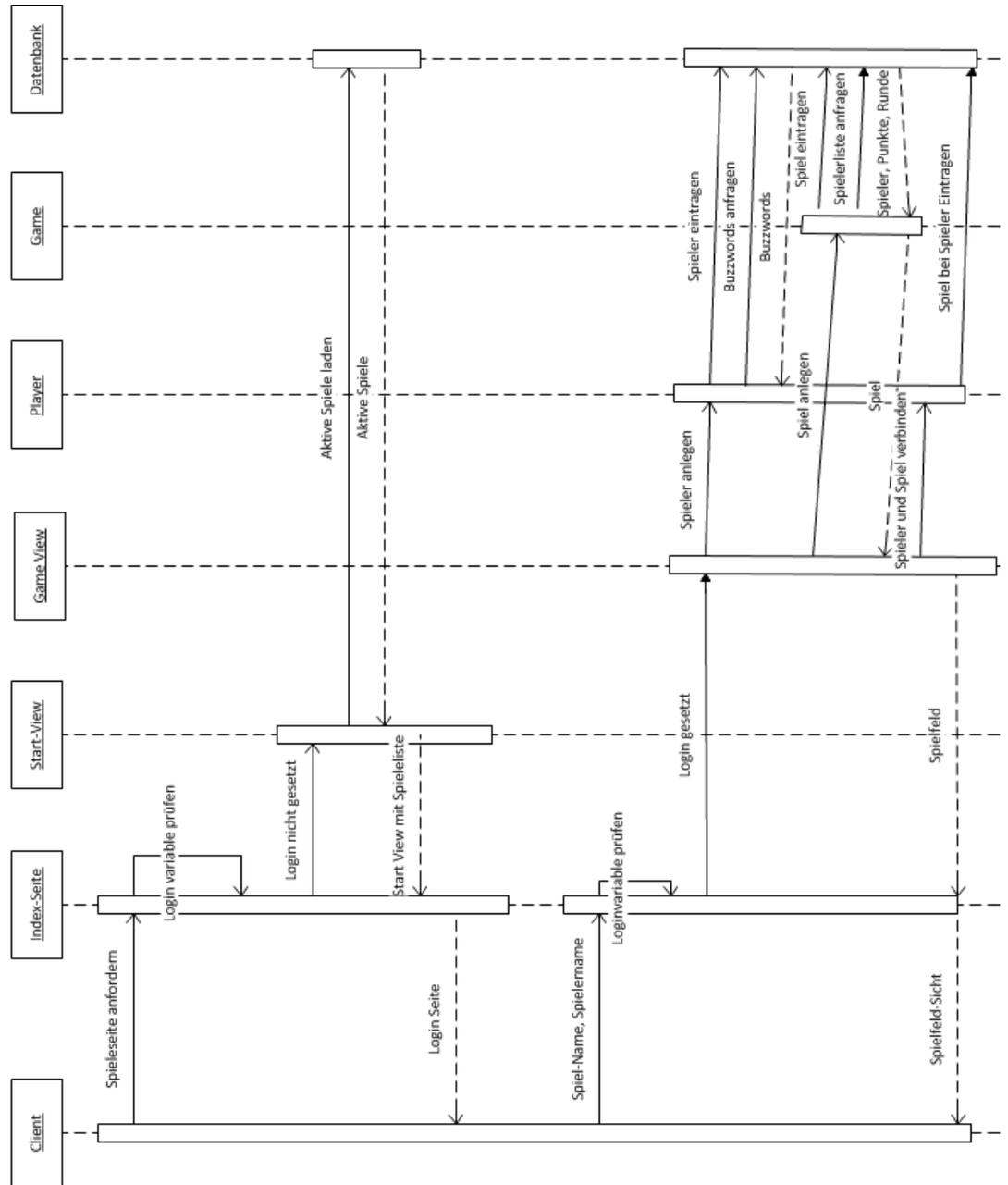


Abbildung 4. zeigt das Anlegen eines neuen Spiels

zurückgelieferte ID wird über das Spieler Model beim Spieler in der Datenbank hinterlegt. Nun bekommt der Benutzer das Spielfeld und die Spielerliste angezeigt. Bis weitere Spieler diesem Spiel beitreten, wird er erst einmal der einzige Benutzer sein.

### 5.3 Verteilungssicht

Die Verteilungssicht (Abbildung 5) zeigt auf, wie die einzelnen Komponenten verteilt sind.

- Webserver
  - **Beschreibung:** Heutzutage ist es üblich die Datenbank auf der gleichen Serverhardware laufen zu haben. Damit ist die Datenbank nur lokal vom Server ansprechbar. Auf dem Webserver werden unsere PHP Skripte abgelegt. Diese kommunizieren serverintern mit der Datenbank. Über Internet kann auf die PHP Skripte zugegriffen werden bzw. bei Aufruf des Spiels wird die Index Seite zurückgegeben.
  - **Leistungsmerkmale:** Der Webserver muss PHP unterstützen, um die Skripte interpretieren zu können. Desweiteren wird eine MySQL Datenbank auf dem Server benötigt, um die Buzzwords abzulegen und Änderungen im Spiel speichern zu können, da die Skriptobjekte von PHP nur zur Skriptlaufzeit leben, diese endet jedoch mit Ausliefern der Seite an den Client.
- User Client
  - **Beschreibung:** Der Userclient besteht aus einem internetfähigen Gerät mit einem Browser, über diesen kann er die URL des Spiels aufrufen und sich an dem Spiel anmelden. Die Kommunikation mit dem Server geschieht über das Internet.
  - **Leistungsmerkmale:** Der Browser des Users muss JavaScript unterstützen, da nur damit der reibungslose Ablauf des Spiels mit schneller Interaktion möglich ist.

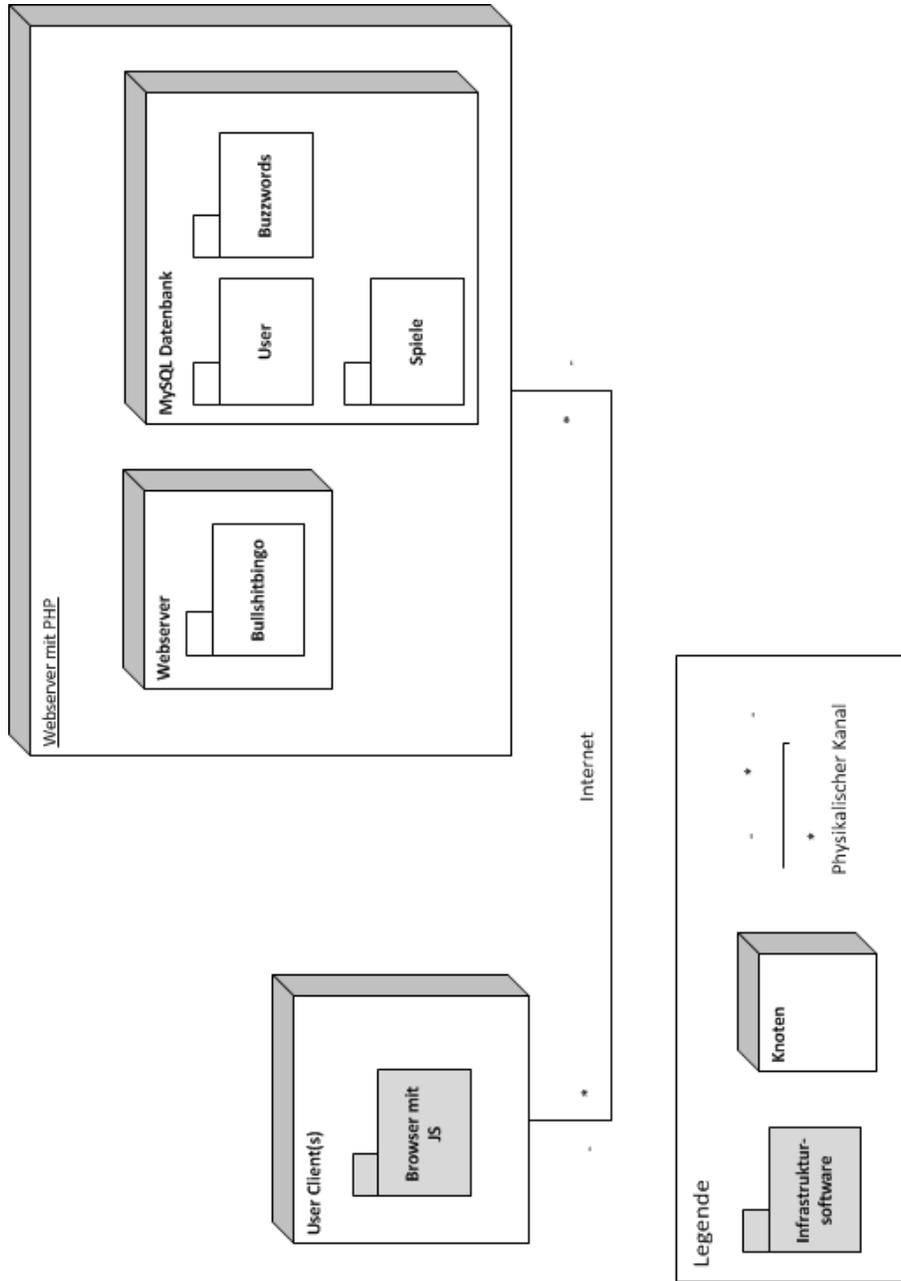


Abbildung 5. zeigt die Verteilung der Komponenten des Bingospiels

## 6 Projektreview

### 6.1 Interaktion im Team

Die Designentscheidungen fielen alle recht schnell und einstimmig. Auch die Entscheidung, eine einheitliche Entwicklungsumgebung zu verwenden, um Inkompatibilitäten im Versionsmanagement vorzubeugen, fielen wir schon früh im Projekt. Der allgemeine Umgang miteinander war immer aufgeschlossen und offen, Unstimmigkeiten gab es keine, lediglich das Refactoring von fremdem Quellcode führte manchmal zu Unklarheiten. Häufiges Pair- bzw. auch manchmal Group-Programming hat das Arbeitsklima positiv beeinflusst und die Fehlerzahl reduziert oder die Fehlersuche zumindest vereinfacht und erträglicher gemacht. Leider war die Einteilung in Arbeitsmodule aufgrund des geringen Projektumfangs nur sehr grob, deswegen war der Entwicklungsprozess an manchen Stellen etwas unkoordiniert. Daher gab es auch nur wenige klare Verantwortlichkeiten innerhalb des Projektes. Die lose Zeitplanung stellte kein Problem dar, da wir uns immer wieder zu Terminen zur Absprache oder Implementierung verabredeten und so nie in Zeitverzug gerieten.

### 6.2 Technische Aspekte

Das Projekt ist, obwohl der objektive Umfang eher gering ist, relativ komplex geworden. Die Komplexität rührt daher, dass bei der gewählten Umsetzung viele Technologien ineinandergreifen (PHP, JavaScript, HTML, CSS, SQL). Dies, sowie die Tatsache, dass auf diese Weise gezwungenermaßen viele kleine Dateien entstehen, führt dazu, dass das Projekt schwerer zu überblicken ist. Dem wurde mit einer simplen Model-View-Controller-Architektur sowie einer strukturierten Verzeichnishierarchie entgegengewirkt.

Die Fehlersuche und insbesondere das Testen wurden stark durch die verwendeten Technologien erschwert:

Durch die vielen verwendeten Technologien war es manchmal schwer nachzuvollziehen, wo der aufgetretene Fehler zu suchen war.

Weiterhin ist das Debugging in JavaScript nur schwer möglich, da viele Befehle in String-Form vorliegen und daher z.B. Tippfehler erst zur Laufzeit erkannt werden können. Eine vorherige Syntaxprüfung ist innerhalb von Strings bekanntermaßen nicht möglich. Zusätzlich verschlucken die meisten Browser Fehlermeldungen, anstatt sie anzuzeigen, sodass durch spezielle Browserplugins [5] teilweise Abhilfe geschaffen werden musste.

Das verwendete Model für den Client barg einige Tücken:

Durch die Vielfalt an Browsern und deren Eigenheiten bei der Interpretation von HTML, CSS und JavaScript war es schwierig, das Userinterface browserübergreifend gleich zu gestalten. Als Gegenmaßnahme verwendeten wir das JQuery Framework, um zumindest die Probleme der JavaScript-Interpretation

einzudämmen.

Gerade mit Microsofts Internet Explorer gab es massive Probleme. Hier wurden nicht nur die CSS Dateien unerwartet interpretiert, sondern es kam bzw. kommt auch zu äußerst merkwürdigen Problemen in der Javascript Ausführung (und sogar in der HTML Darstellung).

Besser wäre gewesen, eine eigene Client-Applikation zu schreiben, die unabhängig von Fremdsoftware ist (beispielsweise auf Basis von Java).

### 6.3 Lessons Learned

Web- und insbesondere Browserbasiertheit erzeugt aufgrund der Abhängigkeit von Browsern und deren Interpretation der Inhalte viel Aufwand, der nichts mit dem eigentlichen Ziel zu tun hat. Den entstehenden zeitlichen Overhead hätte man schon zu Beginn einkalkulieren und eventuell andere Lösungen in Betracht ziehen können.

Eine einheitlichere Technologielandschaft vereinfacht das Gesamtmodell, da viel weniger Schnittstellen nötig sind.

Das Unterteilen in Arbeitspakete /zu schreibende Programmteile, zumindest in grober Form, lohnt sich auch bei kleineren Projekten, da so zumindest immer ein eindeutiger Ansprechpartner bei Problemen existiert.

Die Vorteile von Pair-Programming rechtfertigen den zusätzlichen Einsatz von Teammitgliedern, gerade wenn diese die Schnittstellen zwischen den einzelnen Modulen spezifiziert haben.

Schlussendlich bleibt anzumerken, dass eine Lösung mit der uns sehr gut bekannten Sprache Java einfacher gewesen wäre:

Zum einen beherrschen alle Teammitglieder Java und müssen sich nicht erst Wissen aneignen. Zum anderen ist die Technologie viel homogener, da Client und Server beide in Java implementiert sind. Mit Java fällt die Trennung von Client und Server einfacher, falls man sich für eine Client-Desktopapplikation und einen zentralen Server entscheidet.

Ein zusätzlicher Vorteil wäre gewesen, dass die optische Repräsentation des Spiels durch die Plattformunabhängigkeit einheitlich ist.

Außerdem ist das Testen viel besser zu automatisieren und auch klarer.

## 7 Schlusswort

Das Spiel Bullshit-Bingo wurde im Rahmen der Vorlesung Verteilte Systeme als Webanwendung realisiert. Hierzu wurden ein Client (browserseitiges Script) und serverseitige Scripte implementiert. Es kamen die Technologien PHP, Javascript inkl. Ajax und JQuery, CSS und MySQL zum Einsatz. Gerade diese heterogenen Technologien brachten große Herausforderungen in der Implementierung für das Team mit sich. Diese konnten zwar beherrscht werden, sie und der damit verbundene Overhead wären aber bei einem anderen Ansatz sicher nicht so enorm ins Gewicht gefallen. Trotz allen technologischen Herausforderungen und Hürden ist es uns aber gelungen ein spielfähiges Multiplayer-Bullshit-Bingo unter Verwendung der gängigen Webtechnologien zu realisieren.

## Literatur

1. Wikipedia Artikel zu Buzzword-Bingo, Stand 31.03.2011 bei Erstellen der Dokumentation:  
<http://de.wikipedia.org/wiki/Buzzword-Bingo>
2. Offizielle Seite der Netbeans IDE:  
<http://netbeans.org/>
3. Bitbucket Repository unseres Projekts:  
<http://bitbucket.org/troeder/bullshitbingo>
4. Offizielle Homepage des XAMPP Projekts:  
<http://www.apachefriends.org/de/xampp.html>
5. Offizielle Homepage des Firebug Projekts:  
<http://getfirebug.com/>
6. Offizielle JQuery Homepage:  
<http://jquery.com/>