Die Entwicklung von Psystatix

Informatikprojekt

am Institut für Informatik und angewandte Mathematik
der Universität Bern

von Frank Wettstein
und Niklaus Jordi

im Winter 2004/2005

Betreut durch:

Prof. Dr. Oscar Nierstrasz
Inst. für Informatik (IAM)
Neubrückstrasse 10
3012 Bern

M. Sc. Orla Greevy
Inst. für Informatik (IAM)
Neubrückstrasse 10
3012 Bern

Inhaltsverzeichnis:

1 Ziel des Dokuments

Ziel dieses Dokumentes ist es eine Übersicht über die Arbeit während der Vorlesung Praktikum in Softwareengeneering (PSE) und das anschliessende Projekt zu vermitteln. Dieser Text richtet sich an Informatikstudenten, welche noch kein grösseres Projekt bearbeitet haben, und soll ihnen einen Eindruck vermitteln, wie so etwas ablaufen kann, wo gewisse Schwierigkeiten liegen können, und wie man sie vermeiden kann.
Das Dokument ist folgendermassen aufgebaut: Im ersten Teil werden Ziele und Aufbau des Dokuments festgelegt. Der zweite Teil gibt eine Übersicht über die Funktionalität und den Domain des erstellten Programms, im dritten Teil wird erklärt, wie unser Programm aufgebaut ist und der vierte und fünfte Teil widmen sich unseren Erfahrungen während der Implementationsphase in PSE bzw. während der Weiterentwicklung als Informatikprojekt.
Dieser Bericht enthält ausser den sachlichen Beschreibungen noch einige persönliche Meinungen und Kommentare, welche kursiv gedruckt sind.

2 Was ist Psystatix

In der Vorlesung „Praktikum in Softwareengeneering“ (PSE) hatten wir die Aufgabe eine Web-Applikation zu entwickeln, welche es der psychiatrischen Abteilung des Inselspitals ermöglicht ihre Patienten und deren Status, d.h. ihr Krankheitsbild, zu erfassen. Das Programm sollte Daten über den Patienten und seinen Zustand in einer Datenbank speichern und einen formalisierten Bericht über den Patienten erstellen.
Um den Zustand eines Patienten zu beschreiben, sollen ausschliesslich Begriffe verwendet werden, welche das Programm über eine Datenbank zur Verfügung stellt. Jedem so ausgewählten Begriff wird durch den Arzt zusätzlich ein Gewicht zugeordnet, je nach dem wie stark das entsprechende Symptom ist.
Es soll auch möglich sein die Patientendaten und die verschiedenen Berichte über den Patienten abzufragen und möglicherweise nachzubearbeiten.
Ausserdem soll Psystatix als Lernprogramm dienen, worin die verschiedenen Symptome und deren Definitionen nachgeschlagen werden können. Dies beinhaltet eine Suchfunktion, um die entsprechenden Begriffe zu finden. Schliesslich soll es möglich sein den einzelnen Begriffen Texte, Videos und Audiofiles zur Dokumentation zuzuordnen.
Um die Anwendung plattformunabhängig und die Installation möglichst einfach zu machen, sollte ein Browser als Userinterface dienen und die Applikation nur auf einem Server laufen.
Diese Aufgabe wurde in Gruppen von 5 – 6 Personen bearbeitet. Insgesamt arbeiteten 6 Gruppen an diesem Projekt. Untereinander bestand somit eine Art Wettbewerb, da die Arbeiten am Schluss vom Kunden bewertet wurden.
Da wir im Rahmen von PSE nur die grundlegendsten Anforderungen implementieren konnten und das Inselspital an einer Weiterentwicklung interessiert war, haben wir uns entschlossen, unser Programm als Informatikprojekt zu vervollständigen. Dies geschah in einem zweier Team.

Abbildung 1: Startseite von Psystatix

3 Der Aufbau von Psystatix

3.1 Die Verwendeten Technologien

JSP:
Da die Webseiten unserer Applikation grösstenteils dynamisch erstellt werden müssen, benötigen wir dazu eine Technologie, welche dies ermöglicht. Weil die einzige Programmiersprache, die das ganze Team gemeinsam hatte, Java war, bot sich JSP an, da hier Javacode in HTML-Code eingefügt wird, um dynamische Veränderungen an den Webseiten zu realisieren. Ausserdem war es so möglich, ein Model in Java zu schreiben und einzubinden.

Tomcat:
Um die JSP Seiten anzeigen zu können benötigten wir einen Servlet-Container. Der Bekannteste war Tomcat. Dazu kam, dass dies ein Opensourceprojekt ist, und dass Tomcat auf fast allen Plattformen läuft. Weitere Informationen sind unter http://jakarta.apache.org/tomcat/ zu finden.

MySQL:
Bei der Datenbank hat der Kunde MySQL vorgeschlagen. Auch dies ist ein Opensourceprojekt. MySQL stellt die grundlegenden SQL-Funktionalitäten zur Verfügung, was für die meisten kleinen Projekte und Datenbanken durchaus ausreichend ist. Leider mussten wir feststellen, dass einige Optionen, welche in unserem Fall nützlich gewesen wären, fehlten. Beispielsweise wurden bei der Join-Funktion in älteren Versionen nicht alle Möglichkeiten unterstützt, so dass diese Dinge in Java gemacht werden mussten. Unter http://www.mysql.de/ können nähere Informationen gefunden werden.

CVS:
Ausserdem benutzten wir ein Versionsmanagementtool (CVS), welches es uns erleichterte, gleichzeitig an dem Programm zu arbeiten. Ein CVS-Server mit den nötigen Accounts wurde uns von den Assistenten der Vorlesung zur Verfügung gestellt.

Software	Version	Zweck
Java	1.4.1	Programmiersprache für Model
Tomcat	4.1 bzw. 5.0	Servlet-Container / Web-Server
MySql	4.1.9	Datenbank

Tabelle 1: Versionsnummern verwendeter Software

3.2 Alternative Techonlogien

Hibernate:
Hibernate ist ein Framework, welches die Datenbank mit dem Model verbindet. Es ermöglich eine automatische Abspeicherung von Objekten. Dies wäre eine Erleichterung gewesen, sobald wir uns einmal in das Framework eingearbeitet hätten. Wir waren jedoch der Meinung, dass es schon genug Neues zu lernen gab und dass die Datenbankzugriffe, welche auftreten würden, nicht allzu kompliziert waren. Deshalb dachten wir, dass der Aufwand, sich in ein

neues Framework einzuarbeiten, sich in diesem Fall nicht auszahlen würde. So haben wir uns entschlossen die SQL-Queries selbst zu programmieren.

EJB:
Zur Diskussion stand auch noch EJB, was das Einbinden der Datenbank erleichtert hätte, da hier die Synchronisation mit der Datenbank automatisch geschieht. Hier hat die Mehrheit entschieden, dass dies für unser Projekt zu kompliziert sei, und zuwenig Nutzen bringen würde.

Struts:
Ausserdem überlegten wir uns Struts zu nutzen, um unsere JSP-Dateien zu strukturieren. Struts ist ein Framework, welches es erleichtern soll ein MVC in JSP zu realisieren. Auch hier war ausschlaggebend, dass die meisten Teammitglieder der Meinung waren mit den bisher gewählten, neuen Technologie genug Arbeit haben, und wir dachten, wir könnten auch eine gute Strukturierung ohne Struts erreichen.

3.3 Die Architektur

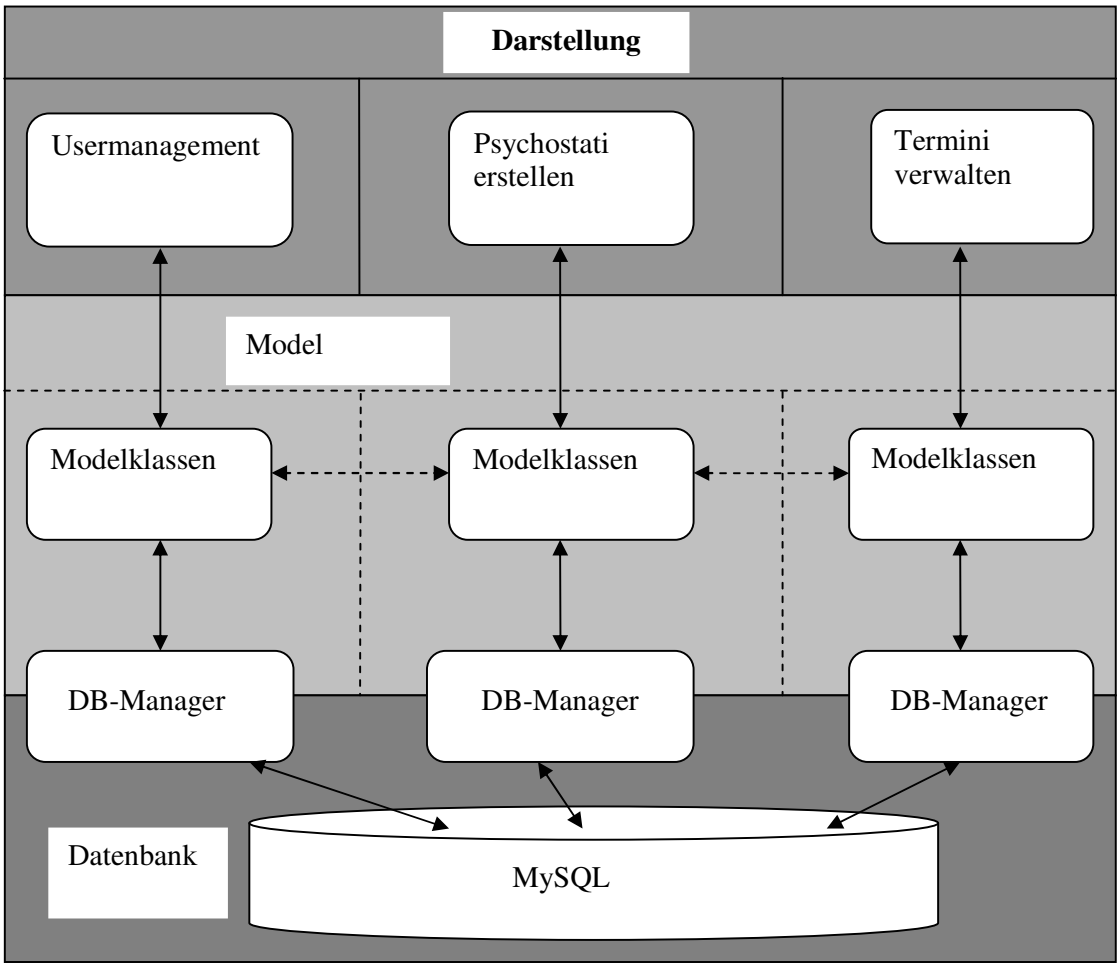


Abbildung 2: Architekturübersicht

Unser Programm ist in drei Schichten aufgebaut. Die erste Schicht bildet die Datenbank. Sie dient dazu die Daten von Usern, Patienten und deren Status zu speichern und zu verwalten. Ausserdem werden die Begriffe um den Zustand der Patienten zu beschreiben in der Datenbank gespeichert.
In der zweiten Schicht befinden sich die Java-Klassen, welche das Model bilden und die Zugriffe auf die Datenbank übernehmen. Diese Schicht war auch dazu gedacht, möglichst viel Javacode aufzunehmen, so dass die JSP-Dateien nicht allzu kompliziert und unübersichtlich würden.
Die dritte Schicht wird von den JSP-Dateien gebildet. Sie übernehmen die Darstellung und den Kontrollfluss.
Es hat sich jedoch herausgestellt, dass eine solch strikte Trennung zwischen den verschiedenen Schichten nicht realistisch war. Es war nicht möglich annähernd soviel Javacode im Model zu halten, wie erwartet. Zu viele Eventualitäten mussten in den JSP-Dateien selber berücksichtigt werden. Somit wurde das Model mehr oder weniger auf ein Interface zur Datenbank reduziert.
Die Datenbank selber bietet nicht immer ausreichend Möglichkeiten für Abfragen (MySQL unterstützt nicht alle SQL-Optionen), so dass die Queries, welche im Model ausgeführt werden, zum Teil sehr kompliziert wurden und manchmal sogar nachbearbeitet werden mussten.

3.4 Das Design

Die verschiedenen Schichten wurden in weitere Teilbereiche unterteilt. So wurden auf der JSP-Schicht Usermanagement, Patientenerfassung und Verwaltung der Termini zur Beschreibung der Symptome unterschieden (siehe Abbildung 2).
Dies spiegelt sich natürlich auch in den darunter liegenden Schichten wieder. Jedoch können hier die Grenzen nicht mehr so exakt gezogen werden, da Daten aus verschienen Teilbereichen an mehreren Stellen gebraucht werden. So haben zum Beispiel die Tabellen, welche die Userdaten in der Datenbank repräsentieren, kaum etwas mit dem Rest der Datenbank zu tun. Es gibt aber eine Verbindung zum Status eines Patienten, weil dort immer erfasst wird, wer den Patienten behandelt hat.
Unser Ziel war ein Model-View-Controller Konzept (MVC) umzusetzen, um wenigstens ein bisschen Ordnung in die sehr komplizierten JSP-Dateien zu bringen. Das Model befindet sich, wie schon erwähnt, in der zweiten Schicht. Die Aufgaben View und Controller wurden in JSP-Dateien implementiert.

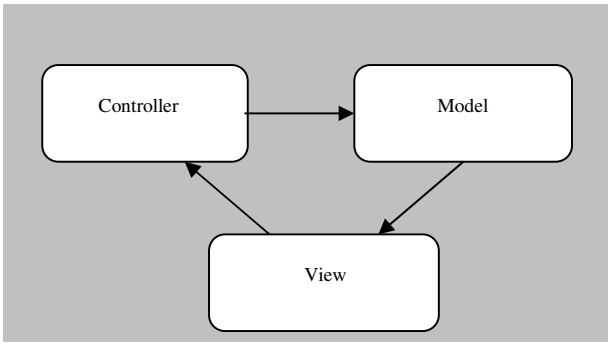


Abbildung 3: MVC-Pattern

Da es aber nicht möglich ist JSP direkt über Änderungen im Model zu informieren, ist das MVC-Konzept nicht vollständig umgesetzt. Die Unterscheidung zwischen View und Controller dient vor allem dazu, in den JSP-Dateien welche für die Darstellung zuständig sind, nicht zuviel Javacode zu haben. Das war jedoch nicht immer zu erreichen, da bei der Darstellung sehr viele Fälle berücksichtigt werden mussten, welche eine Unterscheidung erforderten, die ohne Javacode nicht zu realisieren war.
Da die Klassen, welche das Interface zur Datenbank bilden, an vielen Stellen gebraucht werden, jedoch eigentlich eine Instanz genügt um alle Anfragen zu bearbeiten, haben wir diese Klassen als Singelton implementiert. So sind sie im ganzen System erreichbar, und es gibt nicht unnötig viele Instanzen.
Ein weiteres Pattern, welches wir verwendet haben, ist das Composite-Pattern. Wir haben es benutzt, um die Verschachtelung von Termini und Kategorien zu implementieren.

3.5 Testing

Tests sind in einem grösseren Projekt eine sehr wichtige Hilfe. Man kann so feststellen, dass Methoden und Services von Objekten schon funktionieren, obwohl es dafür noch kein Userinterface gibt. Ausserdem ist es wichtig Fehler im Programm möglichst früh zu entdecken, um den Aufwand für eine Korrektur zu minimieren. Test helfen auch dabei sicherzustellen, dass bekannte Fehler nicht durch Änderungen erneut auftreten.

Um schon während der Implementation zu testen, eignen sich automatisierte Tests besonders gut. Wir kannten für die Tests an den JSP-Dateien kein Framework, und dachten auch, dass hier automatische Tests nicht so wichtig wären, da es im JSP-Teil hauptsächlich um grafische Dinge gehen sollte. Allerdings lohnen sich solche Tests eigentlich immer, und eine gute Möglichkeit dies zu realisieren wäre HTTP-Unit (siehe http://httpunit.sourceforge.net/) gewesen.
Um den Javacode in Model und Datenbankinterface zu testen, benutzten wir JUnit (siehe http://www.junit.org/index.htm). Im Model selber gibt es nur einzelne Methoden, welche komplexere Aufgaben erfüllen, und deshalb auch ausführlich getestet wurden. Der weitaus grösste Teil sind aber ziemlich einfache Methoden, welche keine komplizierte Logik enthalten. Diese wurden nur oberflächlich getestet.
Im Interface zur Datenbank waren die Tests noch wichtiger. Hier gibt es viele komplizierte Abfragen, und ausserdem zum Teil komplizierte Logik zum Aufbau von Datenstrukturen (z. B: zur Verwaltung der Begriffe zum Erstellen eines Psychostatus). Diese Methoden mussten ausführlich getestet werden. Dazu kam noch, dass in Java SQL-Statements einfache Strings sind, und deshalb der Compiler nicht hilft Syntaxfehler zu finden. Diese werden erst beim Ausführen des Codes ersichtlich, und dafür sind automatisierte Tests äusserst nützlich.

Am Ende einer Implementationsphase haben wir immer einen kurzen Test gemacht, bei welchem jedes Teammitglied alle bereits implementierten Funktionalitäten via Userinterface ausprobierte. Wir haben so noch einige Fehler gefunden. Jedoch ist es auf diese Weise unmöglich, alle denkbaren Ereignisse durchzuspielen, so dass lange nicht alle Fehler gefunden werden. Ausserdem lohnt es sich solche Tests zu automatisieren, da man die Zeit, welche man investieren muss um Tests zu schreiben, später durch einfacheres Testen längst wieder gutmacht.

Abschliessend wurde nach jeder Implementationsphase ein kurzer Test durch den Kunden gemacht. Hier wurde vor allem Wert darauf gelegt, die Veränderungen seit der letzten Vorführung zu zeigen, und festzustellen ob die Implementation den Wünschen des Kunden entsprach, oder ob allenfalls Korrekturen nötig waren.

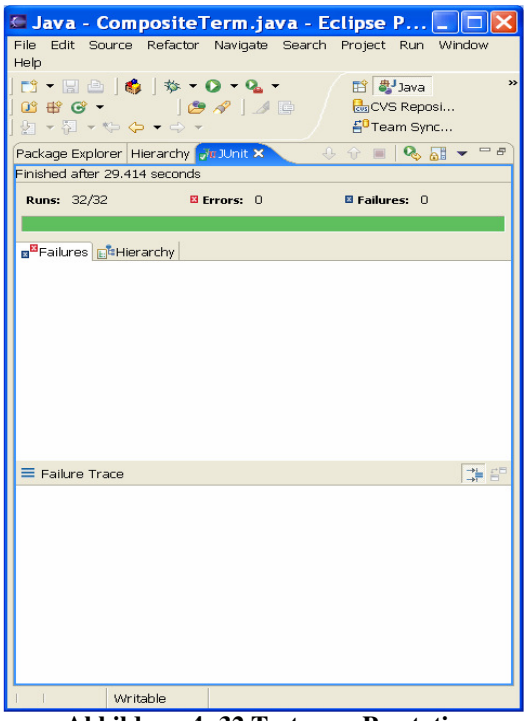


Abbildung 4: 32 Tests von Psystatix

3.6 Dokumentation

Wie bei jedem Projekt wäre es äusserst sinnvoll gewesen, alle Entwicklungsschritte ausführlich zu dokumentieren. Allerdings fehlte in unserem Team ein bisschen die Motivation solche Dokumentationen zu erstellen. So haben wir an Dokumentation nur das Notwendigste gemacht. Dies war folgendes: Ein UML-Klassendiagramm des Models, ein ER-Diagramm der Datenbank und in regelmässigen Abständen ein Statusreport. Ausserdem mussten diverse Präsentationen vorbereitet werden. Der Statusreport sollte eigentlich einmal pro Woche erstellt und im Internet verfügbar gemacht werden. Allerdings waren wir hierbei nicht sehr konsequent. Zu Beginn wussten wir gar nicht, dass dies verlangt war. Später nahmen wir immer die alten Statusreports, und veränderten ein paar Kleinigkeiten, aber wirklich aufschlussreich waren diese Rapporte nie. Wenn wir dies allerdings seriöser gemacht hätten, hätten diese Rapporte einiges über den Verlauf unseres Projekts und den Aufwand zur Implementation von einzelnen Requirements aussagen können.
Eine ausführlichere Dokumentation wäre sicherlich bei der Weiterentwicklung sehr hilfreich gewesen, und hätte uns den Einstig wesentlich erleichtert.
Ursachen für mangelnde Dokumentation gibt es viele. Ein Punkt ist sicherlich das Dokumentation häufig erst gemacht wird, nachdem der entsprechende Teil des Programms fertig gestellt wurde. Zu diesem Zeitpunkt fehlt allerdings dann häufig die Zeit. Dazu kommt, dass für den Kunden die Dokumentation natürlich keinerlei Priorität hat, da er nur das

Programm und seine Funktionalitäten sieht. Aus diesem Grund wird vom Kunden auch nicht unbedingt eine ausführliche Dokumentation verlangt.
Um dies zu verbessern, ist es sinnvoll ein Teammitglied für die Dokumentation verantwortlich zu machen, welches immer wieder auf notwendige Dokumentationen hinweisst. Ausserdem sollte in der Planung nicht nur Zeit für die Implementation der Requirements, sondern auch für die Dokumentation, berücksichtigt werden. Ein sehr wichtiger Punkt wäre, dem Kunden auch die erstellten Dokumente zu verkaufen, d.h. ihm zu erklären warum es sinnvoll ist Zeit in eine gute Dokumentation zu investieren, und ihm mitteilen, welche Dokumente man erstellen will. So könnte erreicht werden, dass die Dokumentation auch aus Sicht des Kunden eine gewisse Bedeutung und Professionalität hat. Ausserdem gibt es Möglichkeiten gewisse Dokumentationen automatisch zu erstellen z.B Javadoc, CVS-Einträge usw. Dies sollte unbedingt so gut wie möglich genutzt werden.

4 Die Entwicklung während PSE

4.1 Vorbereitungsphase

4.1.1 Bildung des Teams

Im Unterschied zu ESE (Einführung in Software Engineering), wo die Teams durch die Assistenten gebildet worden waren, hatte man in PSE bei der Teambildung freie Wahl. Gegenseitiges Kennen und Sympathie waren bei der Teamgründung die entscheidenden Kriterien. Dies war auch bei unserer Gruppe der Fall: Der Kern des Teams(4 Personen) hatte bereits in ESE zusammengearbeitet, die beiden neuen Mitglieder (Vera und Christian) sind aus Sympathiegründen dazugestossen. Der Nutzen davon war eine erleichterten Kommunikation innerhalb der Gruppe und ein lockerer Umgang zwischen den einzelnen Teammitgliedern. Ein weiterer Vorteil wäre es gewesen, die Teams so zusammenzustellen, dass eine möglichst heterogene Mischung unterschiedlicher, sich ergänzender Fähigkeiten zu Stande kommt. Dieses eher technisch orientierte Kriterium spielte aber in den meisten Teams eine untergeordnete Rolle.
Unsere Gruppe setzte sich aus den folgenden Mitgliedern zusammen:

- Vera Fischer (Informatik im Hauptfach)
- Christian Schmid (Informatik im Hauptfach)
- Niklaus Jordi (Informatik im Hauptfach)
- Frank Wettstein (Informatik im Hauptfach)
- Anja Federer (Mathematik im Hauptfach)
- Sebastian Stampfli (Wirtschaft im Hauptfach)

Des Weiteren erhielten wir mit Philip Horwath, einem Studenten aus einem höheren Semester, einen Coach, der uns während des Projektes begleiten und mit Rat zur Seite stehen sollte.

Frank: Die Kommunikation und zwischenmenschlichen Beziehungen bei einer Teamarbeit dürfen nie unterschätzt werden. Dies gilt insbesondere bei Informatikprojekten, wo aufgrund der Komplexität ein erhöhter Kommunikationsbedarf besteht. Ein schlechtes Arbeitsklima führt schnell zu Frustrationen und sinkender Motivation, was sich schliesslich auch negativ auf das Produkt niederschlägt! In unserem Team hatten wir in dieser Hinsicht keinerlei Probleme. Es gab jedoch Gruppen, die sich selbst vor dem Kunden stritten. Auffallend war, dass die Kommunikation in Teams, in der die Teilnehmer unterschiedliche Programmierkenntnisse hatten, deutlich schwieriger war als in Gruppen mit einem homogenen Know-how.

4.1.2 Entscheidungsfindung

Gerade zu Beginn des Projektes mussten viele Entscheide getroffen werden. Es galt eine Technologie zur Umsetzung unseres Projektes auszuwählen und die damit verbundenen Arbeiten untereinander aufzuteilen. Solche Entscheidungen fällten wir während Teamsitzungen, wo man gegenseitig Vor- und Nachteile einzelner Technologien besprach und auswertete. Waren sich nicht alle Mitglieder einig, galt das Mehrheitsprinzip. Eine Ausnahme dessen war die Entscheidung über die Anbindung der Datenbank. Hier wurden nur die Leute einbezogen, die auch tatsächlich damit zu tun hatten. Dies erklärt sich damit, dass die anderen Teammitglieder sich nicht über das Thema informiert hatten, und auch die

Konsequenzen nicht mittragen mussten, da sie nicht an der Implamentation des Interfaces zur Datenbank beteiligt waren.
Anhand unserer Vorgehensweise bei der Entscheidungsfindung lässt sich ablesen, dass es in unserer Gruppe keinen eigentlichen Teamleader gab. Alle Mitglieder waren somit gleichberechtigt. Unser Coach hatte bei Entscheidungen nur einen beratenden Einfluss, überhaupt war unsere Gruppe äussert autonom und nur selten auf die Hilfe des Coaches angewiesen.

Frank: Mit der Wahl von Seiten mit reinem JSP ohne weitere Frameworks hatten wir uns für einen Mittelweg entschieden: Eine Implementation mit PHP war zwar womöglich die technisch einfachste Variante, erschien uns aber als zu wenig mächtig und modular, da es mit PHP kein eigentliches Model gibt wie die Java-Beans in JSP. Eine Implementierung mit JSP inklusive zusätzlicher Frameworks wie EJB oder Struts schien uns zwar als die technisch eleganteste Variante, jedoch auch als die anspruchsvollste. Da wir der Meinung waren mit dem neuen JSP genug Arbeit zu haben und EJB für ein kleines Projekt wie unseres eigentlich übertrieben war – unser Coach sagte dazu oft: „EJB ist wie mit Kanonen auf Spatzen schiessen“ - entschieden wir uns schliesslich zwar nicht einstimmig, jedoch zumindest demokratisch einwandfrei für reines JSP mit dem MVC2-Modell.

4.1.3 Erste Analyse: Sammlung der Anforderungen und Ableiten der User-Stories

Nach einem ersten Kontakt mit dem Kunden, der innerhalb einer einstündigen Information aller Teams stattgefunden hatte, blieben viele Fragen offen. Die Ausführungen des Kunden waren noch etwas unpräzise und für uns als Psychologielaien zum Teil unverständlich. Uns war somit sofort klar, dass die erste Schwierigkeit beim Entwickeln unseres Programms darin bestand, herauszufinden, was für Anforderungen es zu erfüllen hatte. Wir setzten uns deshalb proaktiv mit der für unser Team zuständigen Ansprechperson, Herrn Aebi, in Verbindung um die Anforderungen zu klarifizieren. Laut Lehrbuch wäre es nun die Aufgabe des Kunden gewesen User Stories niederzuschreiben. Diesem fehlte jedoch die dafür notwendige Zeit und Kenntnis. Deshalb haben wir die Stories nach eigenem Gutdünken vorbereitet. Das daraus entstandene Dokument ist im Anhang aufgeführt. Während den darauf folgenden Implementationsphasen haben wir dieses Dokument laufend ergänzt und korrigiert.

Niklaus: In der Startphase eines Projektes ist eine enge Zusammenarbeit mit dem Kunden unumgänglich. Es empfiehlt sich die Stories mit dem Kunden zusammen zu erarbeiten, so kann der Zeitverlust durch Fehlinterpretationen der Aufgabenstellung minimiert werden. Es ist aber wichtig dem Kunden zu genau zu erklären, was von ihm erwartet wird, und weshalb diese Arbeit wichtig ist.

4.1.4 Erstes Design: CRC-Session

Als eine erste Version der User Stories erstellt war, ging es in einem zweiten Akt darum das Skeleton unseres Programms aufzusetzen. Dazu versammelten wir uns zu einer gemeinsamen CRC-Session. In einem mehrstündigen Prozess identifizierten wir Klassen und deren Verantwortungen. Wir verwendeten dazu kleine Karten, die wir mit dem Namen der soeben gefundenen Klasse, deren Verantwortung und deren Partner beschrifteten. Das Schlussresultat dieser Arbeit ist im Anhang in Form eines Klassendiagrammes zu sehen. Obwohl wir während der Implementationsphasen einzelne Designänderungen vornahmen, erwies sich

dieses Grundgerüst als äussert solid. Schon jetzt war uns klar, dass im Model selbst nicht viel Funktionalität enthalten sein würde, und dass es nur relativ wenige und kleine Klassen geben würde.

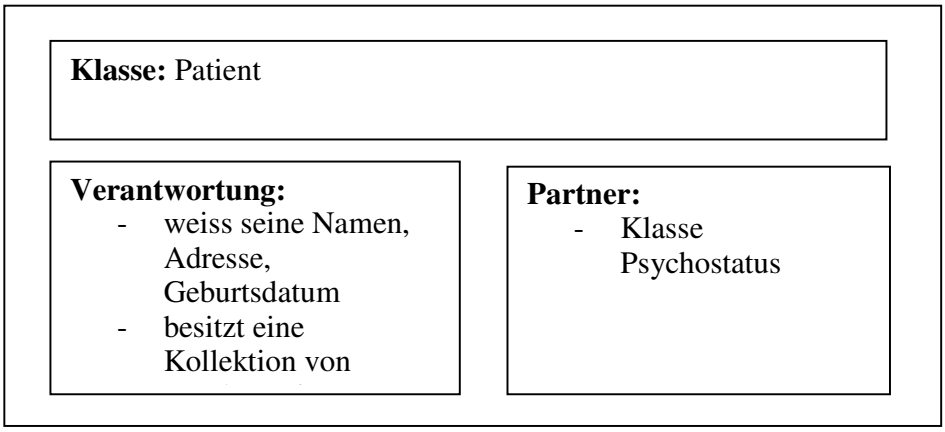


Abbildung 4: Beispiel einer CRC-Karte

4.1.5 Aufteilung der Arbeiten

Wir teilten unsere Gruppe in drei Teilbereiche entsprechend der Architekturebenen auf. Dies sollte die Einarbeitung erleichtern, da sich jeder nur in die neuen Technologien einarbeiten musste, die seine Ebene betrafen. Die Aufteilung sah wie folgt aus:

Name	Arbeitsbereich
Christian Schmid	Datenbank, Interface
Niklaus Jordi	Datenbank, Interface
Anja Federer	Model
Vera Fischer	Darstellung
Sébastien Stampfli	Darstellung
Frank Wettstein	Darstellung

Tabelle 2: Aufteilung der Arbeitsbereiche

Dies spiegelt den Aufwand wieder, den wir in den verschiedenen Schichten erwarteten. Es stellte sich jedoch bald heraus, dass unsere Erwartungen nicht der Realität entsprachen. Die Verbindung mit der Datenbank gab zwar zu Beginn reichlich Arbeit, als aber die grundlegenden Zugriffsmethoden einmal geschrieben waren, änderte sich hier nur noch wenig. Im Model gab es von Beginn weg eigentlich nicht genug Arbeit um eine Person voll zu beschäftigen. Dies führte dazu, dass wir die Gruppe nach wenigen Wochen bereits umorganisieren mussten. Neu war nur noch Christian mit der Datenbank beschäftigt, Anja entwickelte sich zum Allrounder, indem sie Aufgaben im Model und in der Darstellung übernahm und sich auch ums Testing sowie administrative Tätigkeiten kümmerte. Die vier anderen Gruppenmitglieder beschäftigten sich indessen hauptsächlich mit der relativ aufwändigen Darstellung. Diese Umstrukturierung war jedoch nicht einfach, da sich beispielsweise Niklaus, der sich zuvor mit der Datenbank beschäftige hatte, innert kürzester Zeit HTML- und JSP-Kenntnisse aneignen musste. Des Weiteren musste er sich in den bereits erstellten Code einarbeiten, um zu verstehen, wie das System aufgebaut war. Diese Umstellung kostete ihn viel zusätzliche Zeit.

Deshalb ist es ratsam die einzelnen Arbeitsgebiete schon zu Beginn genau zu analysieren, damit eine möglichst zutreffende Aufteilung der Arbeiten ermöglicht und so eine spätere Umorganisation des Teams überflüssig wird. Ausserdem raten wir, falls doch eine neue Aufteilung der Arbeitskräfte erforderlich ist, diese sofort durchzuführen, da der Aufwand umso grösser wird, je länger man damit zuwartet.
Bei der Aufteilung haben wir auch darauf achten müssen, dass Leute aus unserem Team, die nicht Informatik im Hauptfach hatten, weniger Zeit in Psystatix investieren können.

Niklaus: Da ich in allen drei Schichten mitgearbeitet hab und noch während der Implementationsphase das Arbeitsgebiet wechselte, musste ich mich immer wieder in neue Dinge einarbeiten. Dies war für mich sehr schwierig, da ich nicht nur neue Technologien erlernen musste, sondern auch den bereits erstellten Code. Um zu sehen wie gewisse Probleme in JSP gelöst wurden, musste ich immer wieder Sourcecode der anderen Teammitglieder zu Rate ziehen, und manchmal aufgrund von Zeitdruck auch duplizieren.

Frank: Ich selber habe von Beginn an bis zum Schluss auf der Darstellungsseite gearbeitet, sprich die JSP-Seiten von der View und dem Controller editiert. Dass dieser Teil derart aufwändig war, liegt an mancherlei Gründen. Hauptgrund war sicher, dass JSP, HTML und CSS für die meisten von uns relativ neu waren und wir uns zuerst in die verschiedenen Gebiete einarbeiten mussten. Hinzu kommt, dass wir uns von Anfang an viel Mühe gegeben haben, den Webseiten ein ansprechendes Design zu geben. Das Webseitenlayout ist jedoch etwas, das viel Detailpflege benötigt und somit Zeit in Anspruch nimmt. Wer bereits mit den oft widerspenstig anmutenden Eigenschaften gewisser Browser punkto Seitenrendering Erfahrungen gemacht hat, weiss wovon ich spreche. Als weitere Gründe für den hohen Aufwand für View und Controller sind ausserdem die schlechte Lesbarkeit und Testbarkeit von JSP-Dateien aufzuführen. Ein anderer Bremsfaktor war der umständliche Entwicklungszyklus: Keiner von uns hatte sich die Mühe genommen, lokal auf seinem PC sowohl einen Tomcatserver wie auch einen SQLServer einzurichten. Als Folge mussten wir nach jeder Änderung die JSP- Dateien zuerst via CVS einchecken, anschliessend mit einem Anttarget das eingecheckte File auf den Tomcatserver an der Uni laden, unter Umständen den Tomcat neustarten (weil die Version 4.1 geänderte Beans nicht automatisch nachladen konnte), um schliesslich dann in unserem Browser genervt festzustellen, dass wir ein Semikolon vergessen haben.

4.2 Implementation

4.2.1 Erster Prototyp

Um zu testen, ob sich die von uns ausgewählten Technologien verbinden lassen und wunschgemäss funktionieren, implementierten wir ein Login, welches auch für die spätere Applikation benötigt wurde. Beim Login werden sämtliche Technologien verwendet. Dargestellt wird das Ganze von einer JSP-Datei, welche via einer Javaklasse auf die Datenbank zugreift, um zu überprüfen, ob der eingegebene Userename existiert, das Passwort stimmt und welche Zugriffsberechtigungen der User hat.

Als erstes mussten wir Tomcat und MySQL installieren. Bereits hier stiessen wir auf erste technische Probleme. Zuerst gab es beim Entpacken von Tomcat einen Fehler, welcher dazu führte, dass das Programm nicht laufen wollte. Als wir dieses Problem endlich behoben hatten, bekamen wir Schwierigkeiten mit der Konfiguration von Tomcat. Wir brauchten ja für

unsere Applikation einen eigenen Ordner, welcher von Tomcat gefunden werden musste, und einen eigenen Applikationspfad. Auch dies gelang uns nach einigem unnötigen Aufwand schliesslich (eigentlich ist das Einrichten nicht so schwer, wenn man Tomcat ein bisschen kennt). Bei MySQL war das Problem weniger das Programm zu installieren, sondern eher die Verbindung von Java auf die Datenbank zu realisieren. Hier brauchten wir eine Bibliothek (welche schon nicht ganz einfach zu finden war), und ausserdem musste sie in das richtige Verzeichnis des Tomcat kopiert werden (hier trat wieder unsere Unkenntnis von Tomcat zu Tage). Ausserdem war der Zugriff auf die Bibliothek nicht ganz einfach, so dass wir ihn aus einem Beispiel übernehmen mussten.

Zu guter letzt funktionierte das Login jedoch einwandfrei. Wir mussten allerdings schon bald einen kleinen Dämpfer hinnehmen: Wir hatten den Kunden vorgängig nicht klar darüber informiert, dass es bei diesem ersten Prototypen in erster Linie darum ging, die verschiedenen Technologien miteinander zu verknüpfen. Er erwartete daher bereits erste Funktionalitäten und ein ansprechendes Design. Da aber Psystatix mit Ausnahme des Logins noch nichts konnte, zeigte er sich ein wenig über unsere scheinbar kleinen Fortschritte enttäuscht.

Anfangsschwierigkeiten beim Installieren und Verknüpfen unbekannter Programme lassen sich kaum umgehen. Es ist jedoch wichtig aus den Fehlern zu lernen, so dass die Installation später schneller vonstatten geht. Bei der Erstellung eines ersten Prototyps muss man darauf achten dem Kunden zu erklären, was man damit bezweckt. So können Enttäuschungen auf beiden Seiten verhindert werden.

Frank: Auch wenn unser erster Prototyp nur eine bescheidene Funktionalität aufwies, so hatte er für unsere Gruppe dennoch einen wichtigen psychologischen Effekt: Er demonstrierte uns, dass wir fähig waren, die ausgewählten Technologien einzusetzen und miteinander zu verbinden. Dadurch wurde das Risiko des Scheiterns unseres Projektes drastisch reduziert.

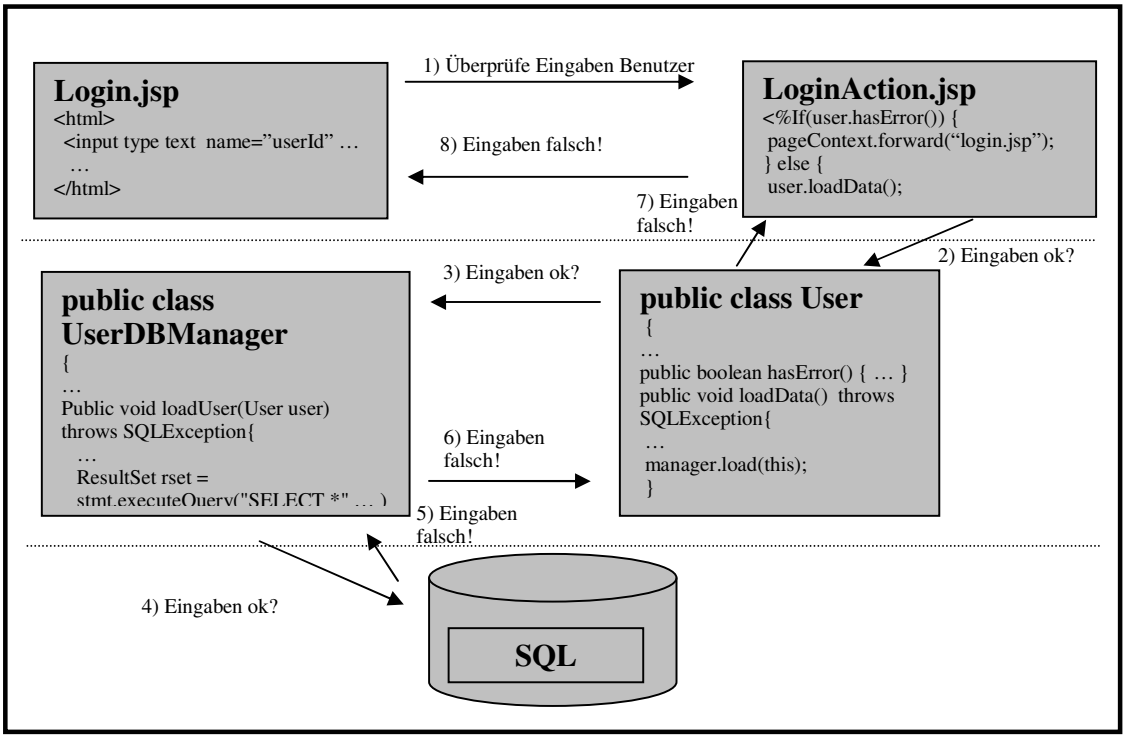


Abbildung 5: 3-Schichten-Architektur unseres Prototypen

4.2.2 Der Iterationszyklus

Die einzelnen Iterationen liefen immer nach dem gleichen Schema ab. Die in der Vorbereitungsphase erarbeiteten User Stories schrieben wir einzeln auf kleine Karten. Wir teilten ihnen unterschiedliche Prioritäten zu und versuchten den Arbeitsaufwand für die Stories einzuschätzen. Da wir wussten, dass man in der Regel zu tiefe Schätzungen abliefert, haben wir den Aufwand absichtlich recht hoch eingeschätzt. Damit lagen wir meistens recht gut. Schon vorher hatte jedes Teammitglied angegeben wie viele Stunden pro Woche es in Psystatix investieren konnte bzw. wollte. Daraus ergab sich die Gesamtarbeitszeit, welche wir dem Kunden pro Iteration zur Verfügung stellten. Nun wurden mit dem Kunden zusammen die Stories ausgewählt, welche wir implementieren sollten. Hier wurde darauf geachtet, dass unser Arbeitszeitbudget nicht überschritten wurde, ansonsten konnte sich der Kunde die zu implementierenden Stories relativ frei auswählen. Natürlich gab es zwischen einigen Stories Abhängigkeiten, welche auch berücksichtigt werden mussten. Danach teilten wir die Arbeiten unter uns auf, damit jeder mit der Implementation der ihm zugeordneten Aufgaben beginnen konnte. Allerdings waren unsere Einschätzungen bei der Aufteilung der Aufgaben nicht immer ganz korrekt, so dass einige Teammitglieder wesentlich mehr zu tun hatten als andere. Auch kam es vor, dass man auf die Implementation einer Story warten musste, bevor man überhaupt mit der eigenen beginnen konnte. Dies führte während der Implementation doch immer wieder zu Wartezeiten

Bevor das Team mit der Implementation der gewünschten Features startet, sollte daher ein möglichst genauer Zeitplan erstellt werden, welcher festlegt, wer welche Arbeit bis wann fertig stellt. So kann die Arbeit besser verteilt und Leerlauf vermieden werden.

Vor der Validierung durch den Kunden fand am Ende des Iterationszyklus immer eine kurze Testphase statt, wo wir nebst dem Abspielen der Unittests auch das Userinterface interaktiv testeten. Jedoch fehlte uns oft die Zeit für ausführliches Testen. Im Anschluss daran führten wir dem Kunden die neuen Features unseres Programms vor. Dieser entschied dann, ob er mit den implementierten Stories zufrieden war oder ob allenfalls Verbesserungen nötig waren.

4.2.3 Die einzelnen Iterationen

Die Dauer der einzelnen Iterationen konnten wir selber festlegen. Da der Kunde nur wenig Zeit hatte, und wir dachten, es wäre für uns einfacher, wenn wir uns die Zeit möglichst selber einteilen könnten, begannen wir mit einer vierwöchigen Iterationsdauer. In der ersten Iteration ging es vor allem darum grundlegende Features des Programms zu implementierten. Dazu gehörten einige Verbesserungen am bereits bestehenden Login, das Usermanagement, das Erfassen von Patienten, das Auswählen der Termini sowie einige Dinge, die nichts direkt mit dem Programm zu tun hatten, aber uns die Arbeit erleichtern sollten, wie beispielsweise ein automatisches Neuaufsetzen der Datenbank.
Wir mussten leider feststellen, dass wir mit einer vierwöchige Iterationsdauer nur schlecht umgehen konnten. Zu vieles verschoben wir am Anfang auf die nachfolgenden Wochen, zu gemächlich sind wir ins Projekt gestartet. Die Validierung war ja noch fern, und wir dachten, dass wir alles rechtzeitig fertig stellen könnten. Dies führte schliesslich dazu, dass drei Tage vor der Validierung nur ein Bruchteil der verlangten Features implementiert war. Wir sahen uns gezwungen den Stand unseres Projektes im Statusreport auf gelb (=gefährdet) zu setzen. Nach zwei langen Tagen und kurzen Nächten haben wir die termingerechte Implementation der gewünschten Stories doch noch zu Stande gebracht. Eine derartige Hektik hätten wir natürlich ganz leicht durch besseres Einteilen der Arbeit verhindern können. Als weitere

Massnahme entschlossen wir uns zukünftig nur zweiwöchige Iterationen durchzuführen. Dadurch wurde unser Arbeitspensum überschaubarer. Falls uns wieder keine optimale Arbeitseinteilung gelingen sollte, hatten wir zwar vor der Validierung wieder ein bisschen Stress, aber dann wenigstens nicht mehr im gleichen Ausmass.
In der zweiten und dritten Iteration haben wir vor allem an einem Interface für die Eingabe und Verwaltung der Termini, welche für die Beschreibung der Symptome eines Patienten zur Verfügung stehen sollen, gearbeitet.

Niklaus: Besonders am letzten Tag der ersten Iteration hatten wir sehr viel zu tun, um die Deadline noch einzuhalten. Die Methoden in der Datenbank und im Model waren weitgehend vorhanden, aber den Aufwand in der Darstellungsschicht hatten wir völlig falsch eingeschätzt. Ich habe dann versucht in der Darstellungsschicht auszuhelfen, musste aber feststellen, dass ich nur die einfachsten Aufgaben übernehmen konnte, da ich mich kaum in JSP eingearbeitet hatte, und ich auch nicht wusste wie die verschiedenen Seiten verbunden waren. Deshalb war ich nur sehr begrenzt eine Hilfe.

4.3 Der krönende Abschluss

Leider war ein Semester zu kurz, um alle gewünschten Anforderungen implementieren zu können. Das Potenzial, welches in diesem Projekt steckte, konnte nur annähernd ausgeschöpft werden. Somit blieben weiterhin viele Wünsche des Kunden offen, so z.B. eine Möglichkeit zur Verlinkung der Termini mit Dokumentationsdateien (Texten, Audiodateien, Videos). Der Kunde zeigte sich mit unserem Produkt jedoch sehr zufrieden und verlieh unserem Psystatix gar den ersten Platz. Als Dankeschön erhielten wir eine Flasche Champagner und die Möglichkeit innerhalb einer Projektarbeit an Psystatix weiterzuarbeiten. Von dieser Möglichkeit machten Niklaus und Frank schliesslich ein Jahr später Gebrauch. Doch dazu später mehr.

Frank: Der Hauptgrund, dass unser Produkt dem Kunden gefallen hat, liegt wohl daran, dass wir von Beginn an produktbezogen gearbeitet haben. Wir verzichteten bewusst auf technische „state-of-the-art"-Lösungen (z.B. EJB verbunden mit Struts) um uns ganz auf die Implementierung und Darstellung von Features konzentrieren zu können.

4.4 Rückblick

4.4.1 Die Zusammenarbeit mit dem Kunden

Es ist allgemein bekannt, dass bei Informatikprojekten hie und da Schwierigkeiten bei der Kommunikation mit dem Kunden auftauchen. Dies war auch bei uns der Fall.
Im Grossen und Ganzen muss jedoch gesagt werden, dass unser Kunde sehr verständnisvoll war und uns ein angenehmer, umgänglicher Partner während der Implementation war. Dass er nicht aus der Informatikbranche kam, war ausserdem in gewissen Situationen eher ein Vorteil, da ein informatikbewandter Kunde unsere Arbeitsweise und die Qualität unseres Produktes besser hätte beurteilen und somit auch besser hätte kritisieren können!
Es folgt nun eine Auflistung der drei grössten Schwierigkeiten, denen wir begegnet sind. Wir versuchen für jedes Problem eine mögliche Lösung aufzuzeigen.

- *„Changing requirements“:* Oft kam es vor, dass sich die Anforderungen an das Programm änderten, und zwar manchmal auch nachdem ein Feature vollständig implementiert und validiert war. Ein Beispiel dafür sind die verschiedenen Zugangsberechtigungen für User. Zu Beginn sollten alle User die gleichen Berechtigungen haben. Nach der ersten Iteration wurde uns mitgeteilt, dass wir noch zwischen normalen Usern und Oberärzten unterscheiden müssen, weil für jeden Psychostatus ein Arzt und ein Oberarzt zuständig waren, und ausserdem nur Oberärzte neue User und Termini eingeben durften. Noch etwas später wurde uns gesagt, dass User die keine Berechtigung mehr haben, nicht einfach gelöscht werden dürfen, weil die Informationen über den Verfasser eines Psychostatus möglicherweise auch weiterhin noch gebraucht werden. So hatten wir also zum Schluss nicht nur eine Art von Usern, sondern deren drei.
  Lösunsansatz: Um die Häufigkeit von sich verändernden Anforderungen zu reduzieren, empfiehlt sich die Ausarbeitung eines schriftlichen Vertrages, in dem man von Beginn an mit dem Kunden die gewünschten Anforderungen spezifiziert. Beide Parteien sind anschliessend verpflichtet sich an diesen Vertag zu halten.
- *Unterschiedliche Ansprechpartner:* Jeder Ansprechpartner hatte unterschiedliche Erwartungen und Anforderungen an das Programm. So bekamen wir nicht nur verschiedene, sondern teilweise auch widersprüchliche Anweisungen. Zum Beispiel sollte unser Programm dafür sorgen, dass sich der behandelnde Arzt jede Kategorie von Termini mindestens einmal anschaut. Um dies zu gewährleisten, stellten wir an den Psychostatus die Anforderung, dass aus jeder Kategorie ein Terminus ausgewählt werden musste. Als wir dies aber dem Kunden vorführten, war derjenige, der uns diesen Auftrag erteilt hatte, zwar sehr zufrieden, die restlichen Ansprechpartner jedoch fanden diesen Zwang, aus jeder Kategorie einen Terminus auszusuchen, unpraktisch.
  Lösungsansatz: Es ist wichtig, dass man von Beginn an verschiedene Ansprechpartner bei der Spezifikation der Anforderung miteinbezieht, um so früh unkohärente Anweisungen entdecken und direkt mit dem Kunden aus dem Weg räumen zu können. Auch für die verschiedenen Ansprechpartner wäre es wichtig, sich untereinander zu besprechen um widersprüchliche Anweisungen zu vermeiden.
- *„Customer doesn’t know what he wants“:* Zu den bekannten Problemen gehört, dass der Kunde nicht so genau formulieren kann, was er eigentlich will. Vor allem kennt er sich kaum oder gar nicht in Informatikbegriffen aus. Dies führt häufig zu Missverständnissen, welche manchmal zu falschen oder nicht gewollten Implementationen von Requirements führen, was später korrigiert werden muss. Das ist natürlich mit einem zusätzlichen Aufwand verbunden. *Lösungsansatz*: Es ist sehr wichtig in engem Kontakt mit dem Kunden zu stehen und auf kontinuierlichen Feedback zu bestehen. Man sollte immer wieder bestrebt sein klarzustellen, wie man einen Auftrag verstanden hat und die Implementation immer wieder in kurzen Abständen vorführen, um mögliche Fehlinterpretationen früh zu erkennen, und so den Aufwand bei der Verbesserung gering zu halten.

4.4.1.1 Die Zusammenarbeit im Team

Unser Team bestand aus sehr unterschiedlichen Personen, welche verschiedene Arbeitsweisen hatten und auch unterschiedlich viel Zeit in das Projekt investieren konnten. Ein Hauptunterschied lag hierbei zwischen Personen, welche Informatik als Hauptfach und denjenigen welche Informatik als Nebenfach belegt hatten. Es war von Anfang an klar, dass die zwei Personen in unserem Team, für welche Informatik ein Nebenfach war, nicht so viel

Zeit aufwenden konnten wie die anderen Teammitglieder. Dies führte aber kaum zu Problemen, da jeder zu Beginn selber entschied, wie viel Zeit er für Psystatix aufwenden konnte oder wollte. Dies wurde dann in der Gesamtarbeitszeit berücksichtigt, welche dem Kunden pro Iteration zur Verfügung gestellt wurde.
Ein wesentlich grösseres Problem in unserer Gruppe war, dass verschiedene Prioritäten gesetzt wurden. Jeder hatte eine eigene Vorstellung davon, was wichtig war, und für alle war ihre Vorstellung so einleuchtend, dass kaum darüber diskutiert wurde. Dies führte später zu einigen Konflikten, weil manche Features anscheinend nicht so weit waren, wie erwartet, da verschieden an die Arbeit herangegangen wurde. Aber alles in allem konnten diese Konflikte immer gütlich beigelegt werden und störten die Arbeit nicht wesentlich.
Ein sehr wichtiger Aspekt bei einer Teamarbeit ist natürlich die Kommunikation. Wir nutzten viele unterschiedliche Kommunikationskanäle, was vielfach von Vorteil war. Da wir mehrheitlich alle alleine zu Hause arbeiteten, war die Kommunikation via Instant Messanger sehr hilfreich, weil auftretende Probleme mit Schnittstellen oft sofort diskutiert und behoben werden konnten. Natürlich wurden auch Emails rege genutzt, vor allem wenn es darum ging Dokumentationen zu erstellen oder um Präsentationen vorzubereiten. Ausserdem fand fast wöchentlich ein Meeting statt, wo grundsätzliche Probleme und die weitere Vorgehensweise eingehend besprochen wurden. Ein Faktor, der nicht unterschätzt werden darf war, ist, dass wir viele gemeinsame Vorlesungen besuchten und so Gelegenheit hatten allfällige Komplikationen zu besprechen.
Da wir uns bereits kannten und gut miteinander auskamen, hatten wir einen sehr guten Teamgeist, der die Zusammenarbeit wesentlich erleichterte.
Etwas unbefriedigend war allerdings die Kooperation mit unserem Coach. Wir wollten eigentlich zuerst selbst versuchen die Aufgaben zu lösen, und erst auf den Coach zurückgreifen, falls wir nicht mehr weiter kommen würden. Leider hat das nicht immer ganz nach Wunsch geklappt. Unser Coach war meistens da, wenn wir ihn nicht unbedingt brauchten, wenn wir aber mal eine Frage hatten, war er manchmal schwer erreichbar. Wir haben es ihm aber auch nicht unbedingt leicht gemacht. Unser Coach wurde äusserst selten von uns über den aktuellen Stand der Dinge informiert, und im Allgemeinen war die Kommunikation mit ihm weit weniger gut als teamintern. Ein gutes Beispiel dafür waren die Tage vor der ersten Validierung der Requirements mit dem Kunden. Wir waren dem Zeitplan weit hinterher. Unser Coach wusste davon aber nichts. Er schaute sich die Sache etwa zwei Tage vor der Validierung an und merkte, dass das so nicht klappen würde. Er machte sich einige Sorgen, sprach uns aber nicht darauf an. Wir legten dann noch eine extra Schicht ein und konnten so alles rechzeitig fertig stellen. Allerdings, hätten wir unseren Coach über unsere Schwierigkeiten und unsere “Lösung” dafür informieren sollen.

4.4.2 Extreme Programming

Rückblickend kann gesagt werden, dass wir in unserer Arbeitsweise viele Aspekte von „extreme Programming“ übernommen haben. Bereits die Anwendung von sich wiederholenden Interationszyklen mit einer Planungs-, Design-, Implementations- und Testphase ist ein zentrales Element von XP. Es folgt nun eine Tabelle der einzelnen Phasen mit XP-typischen und XP-atypischen Vorgehensweisen.

Phase	XP-typisch	XP-atypisch
Planung	- Schreiben von User Stories - Kurze Releases - Anwendung von Iterationen - Regelmässige Teammeetings	- Spezialisierung (Idee von XP jedoch ist, dass jeder sich im gesamten Code auskennt, also Generalismus)
Design	- Benützung von CRC-Karten - „Keep it simple stupid“ - Benützung von Prototypen	- Refactoring (wurde aus Zeitgründen nur selten gemacht)
Implementation	- „Continuos Integration“ via CVS	- Der Kunde ist schwer erreichbar. - keine Verwendung von „Coding Rules“ in den jsp Dateien. - Kein „Pair Programming“
Testing	- Unit Tests für Java Code - Bevor Code released wird, müssen alle Tests bestanden werden - Acceptance Test vom Kunden nach jeder Iteration	- Keine Tests für JSP-Dateien - Keine Tests gemacht nach Auffinden von Bugs

Tabelle 3: XP-typische/XP-atypische Aspekte

4.4.3 Vergleich mit anderen Teams

Da wir bei der Entwicklung unserer Web-Applikation nicht alleine standen, sondern in „direkter Konkurrenz“ zu 5 weiteren Teams, wurden automatisch Vergleiche zu den anderen Programmen gemacht. „Wie weit sind die anderen Gruppen?“, „Was haben sie besser gemacht als wir?“, „Welche Technologien verwenden sie?“ waren nur einige der Fragen, die hie und da auftauchten und uns schliesslich auf die Homepage der anderen Gruppen leiteten. Viel Zeit für Vergleiche blieb jedoch nicht, da wir bereits mit unserem Projekt alle Hände voll zu tun hatten. Grob gesehen können die Zielsetzungen der Gruppen in 2 Kategorien eingeteilt werden: Rund die Hälfte der Gruppen war darin bestrebt während des Projektes möglichst viel Neues erlernen zu können. Das Erarbeiten von Frameworks wie Struts, EJB oder Hibernate stand somit im Vordergrund. Bei der anderen Hälfte, zu der wir gehörten, war es das Ziel ein möglichst „gutes“ Produkt abzuliefern. Die Lernphase sollte also möglichst kurz sein, damit man sich direkt der Implementation widmen kann. Gruppe 5 beispielsweise entschied sich für PHP und konnte, da sich mehrere Gruppenmitglieder bereits damit auskannten, direkt loslegen. So erstaunt es nicht, dass die Gruppen, die eine „einfachere“ Technologie ausgewählt hatten, am Anfang am schnellsten vorankamen und am Schluss, obwohl die anderen Gruppen aufgeholt hatten, den Kunden am besten von ihrem Produkt überzeugen konnten.

5 Die Weiterentwicklung als Projekt

5.1 Der Wiedereinstieg

5.1.1 Kontaktaufnahme mit dem Kunden

Schon gegen Ende von PSE gab uns der Kunde zu verstehen, dass er an einer Weiterentwicklung von Psystatix interessiert war. Da es aus unserer Sicht noch einige Verbesserungsmöglichkeiten gab, entschlossen wir uns, das Programm als Projekt weiterzuentwickeln. Da wir aber in den folgenden Semestern viel zu tun hatten, verging fast ein Jahr, bis wir uns wieder damit beschäftigten.
Als wir die Bestätigung hatten, dass wir Psystatix im Rahmen eines Projektes weiterentwickeln konnten, nahmen wir erneut Kontakt mit dem Kunden auf. Bei einem ersten Treffen ging es darum, zu sehen was genau unser Programm schon konnte, und wo Verbesserungen nötig waren. Anschliessend besprachen wir, welche neuen Requirements zu implementieren waren. Hier hatte der Kunde hauptsächlich zwei Wünsche: Die Möglichkeit zur Verlinkung der Termini und eine automatisch Diagnosegenerierung (s. unten).

5.1.2 Einrichten der neuen Infrastruktur

Um an dem Projekt weiterzuarbeiten, mussten wir nun auf einem Server MySQL, Tomcat und CVS einrichten. Da wir nun bereits Erfahrungen mit diesen Programmen hatten, ging dies viel leichter als noch während PSE. Ausserdem gab es inzwischen eine neue Version von Tomcat (Tomcat 5.0), welche leichter zu handhaben war.

5.1.3 Psystatix Wiederaufsetzen

Nachdem die Grundprogramme installiert waren, setzten wir Psystatix wieder auf. Dabei stiessen wir auf einige Schwierigkeiten. Wir mussten feststellen, dass wir nach einem Jahr Pause nicht mehr wussten wie unser Programm im Detail funktionierte. Dadurch entstanden, vor allem aus der Verbindung mit der Datenbank, einige Probleme. Um dies zu verhindern ist eine ausführliche Dokumentation äusserst hilfreich.

5.1.4 Beginn der Programmierarbeiten

Da seit PSE einige Zeit vergangen war, gestaltete sich der Anfang der Programmierarbeiten gar nicht so einfach. Nicht nur, dass wir den Code der Teammitglieder, welche nicht mehr dabei waren, verstehen mussten, auch unser eigener erschien uns fremd. Deshalb verging einige Zeit bis wir uns wieder eingearbeitet hatten und uns im Programm wieder auskannten und zurechtfanden. Diesen Aufwand zur Einarbeitung lässt sich kaum vermeiden, jedoch kann er durch gut kommentierten Code erheblich verringert werden. Ausserdem ist es wichtig die Zeit für die Einarbeitung in der Planung zu berücksichtigen.

5.2 Neue Features von Psystatix

5.2.1 Verlinkung mit Dokumentationsdateien

Da Psystatix auch als Lernprogramm genutzt werden sollte, hatte die Realisierung einer Möglichkeit zur Verlinkung der Termini mit Dokumentationsdateien erste Priorität für den Kunden. Es sollte möglich sein für jeden Terminus eine Text-, Audio- und/oder Videodatei aus einem Pool auszuwählen und dem Terminus zuzuordnen. Ein Link zu dieser Datei soll

dann in der Liste der Termini angezeigt werden, so dass die Dokumentationen im Browser angesehen werden können. Dies soll es angehenden Psychiatern erleichtern sich mit den Termini und deren Anwendung vertraut zu machen.

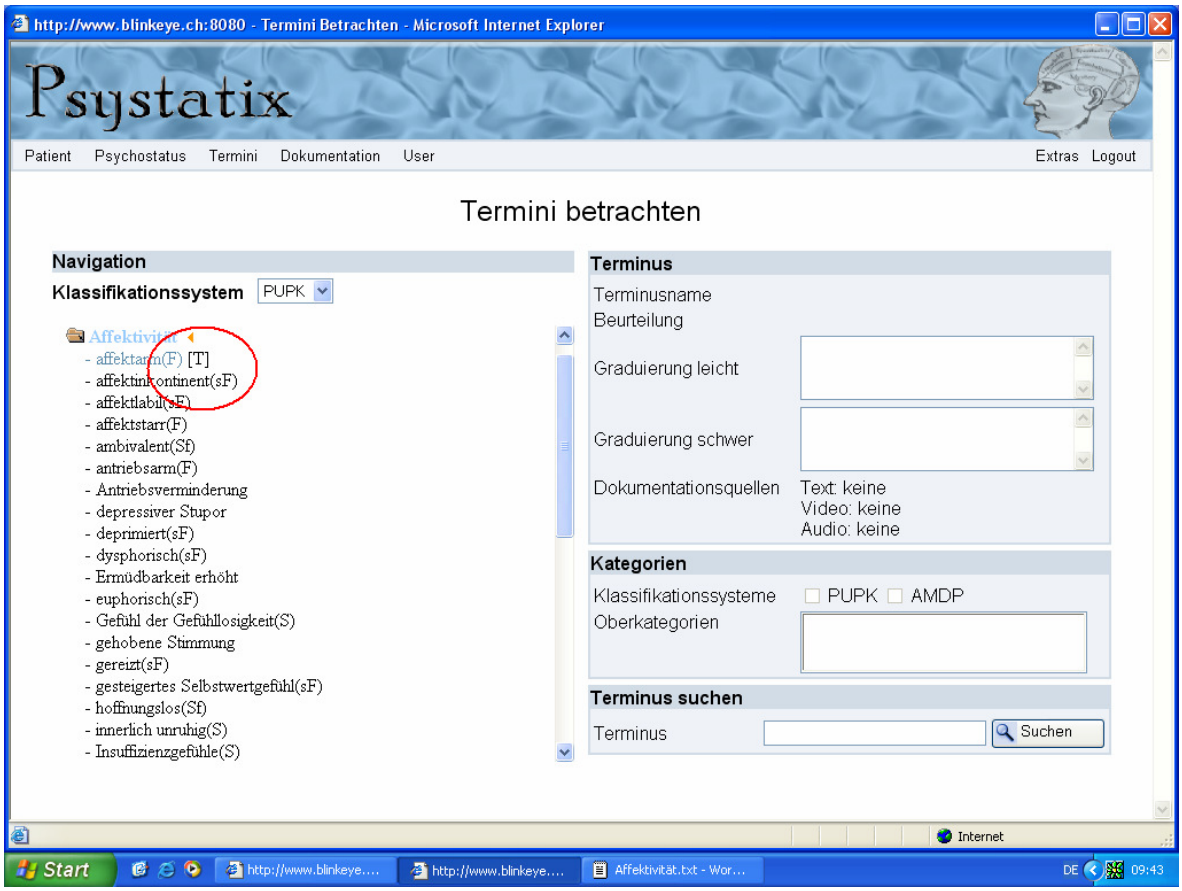


Abbildung 6: Anzeige von Dokumentationslinks

5.2.2 Management von Dokumentationsdateien

Die Dateien welche zur Dokumentation der Termini dienen sollten, müssen natürlich auch verwaltet werden können. Deshalb realisierten wir ein einfaches Portal um Dateien auf den Server zu laden, damit sie später als Dokumentationsquellen benutzt werden können. Ausserdem stellt Psystatix eine Möglichkeit zur Verfügung Dateien, welche nicht mehr gebraucht werden, zu löschen.

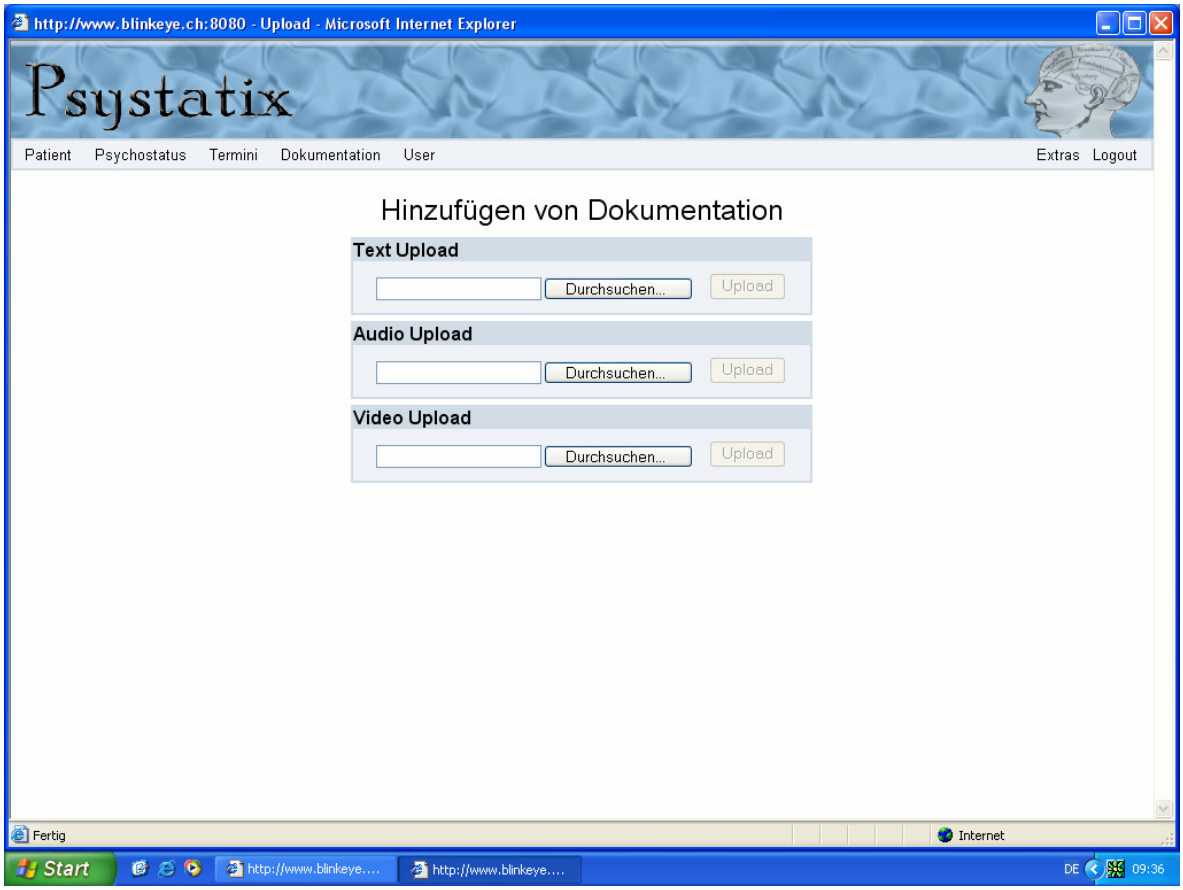


Abbildung 7: Portal zum Hochladen von Dateien

5.2.3 Export von Psychostati

Ein neues Feature von Psystatix sollte die Möglichkeit sein, einen Psychostatus in ein gängiges Format zu exportieren, so dass die Psychostati auch in anderen Programmen verwendet werden können. Aus Gründen der Einfachheit, und da keine komplizierte Formatierung gewünscht war, entschieden wir gemeinsam mit dem Kunden eine Exportfunktion in eine .txt-Datei zu erstellen.

5.2.4 Änderungen am Menu

Das alte Menu, welches wir in PSE entwickelt hatten, erwies sich als schwer skalierbar, und unübersichtlich. Ausserdem gab es einige Mängel im logischen Aufbau des Menus. Aus diesen Gründen haben wir auf Wunsch des Kunden ein neues Menu entwickelt, welches überschaubarer und für mögliche spätere Weiterentwicklung leichter anzupassen ist.

5.2.5 Grafische Verbesserungen

Einige der Seiten, welche noch aus PSE stammten waren etwas überladen und unübersichtlich. Ausserdem hätten einige der neuen Features auf Seiten gehört, auf welchen kein Platz mehr vorhanden war. Deshalb haben wir diese Seiten neu unterteilt. Im Weiteren

haben wir dem gesamten Programm ein neues Layout verliehen. Dies soll die einzelnen Seiten übersichtlicher machen und Psystatix mehr Struktur geben.

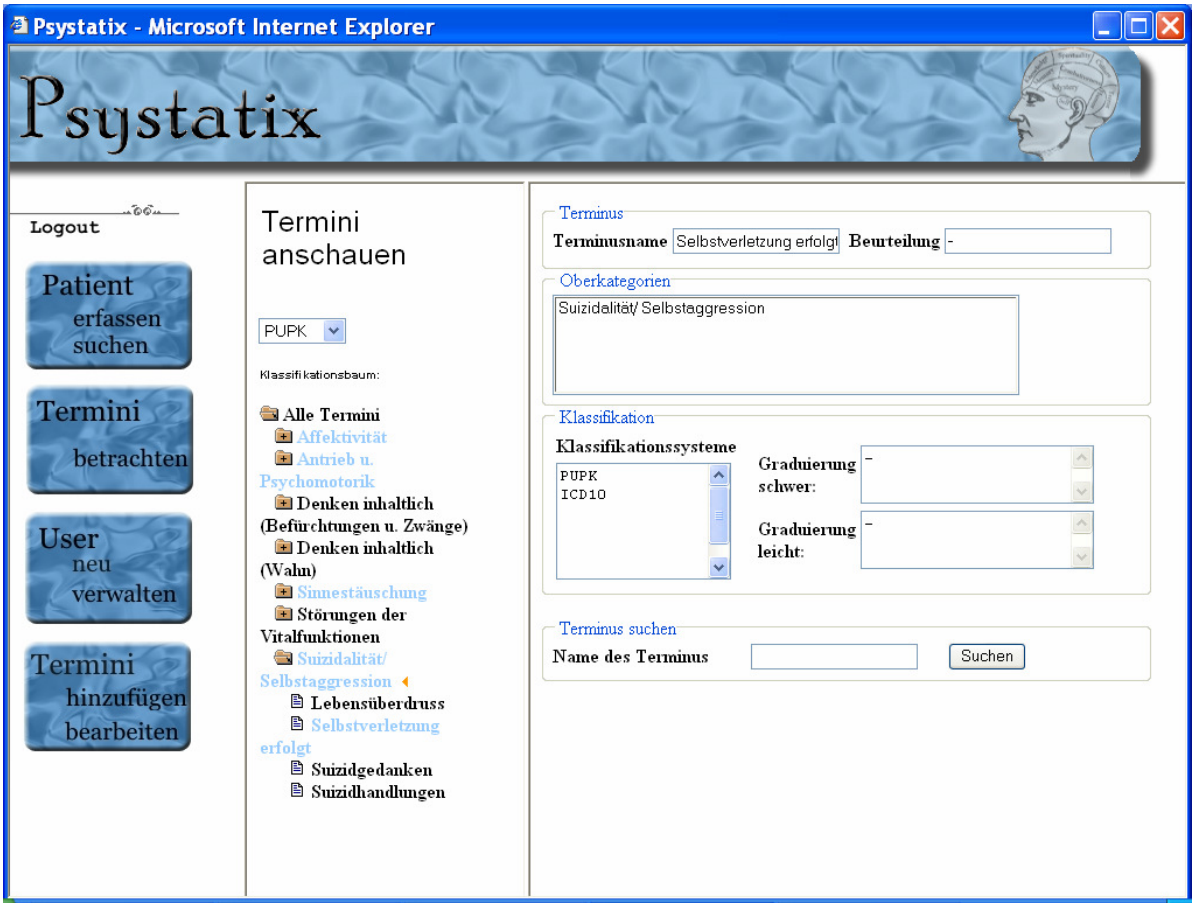


Abbildung 8: Design des "alten" Psystatix (Vergleich mit Abbildung 6)

5.3 Qualitätsverbessernde Massnahmen

Nebst der Implementierung von neuen Features war uns wichtig, die Qualität von Psystatix zu verbessern. In den folgenden Abschnitten werden Verbesserungsmassnahmen aufgelistet.

5.3.1 Refactoring

Wie im Abschnitt 5.1 erwähnt worden ist, war der Wiedereinstieg in Psystatix keine einfache Angelegenheit. Insbesondere auf der Darstellungsseite brauchte es Zeit um den Durchblick zu gewinnen. Dies lag einerseits an der schlechten Lesbarkeit von jsp-dateien aufgrund des Codegemisches von HTML und Java, andererseits am unterschiedlichen Programmierstil der Teammitglieder. Wir hatten es während PSE versäumt „Coding Conventions“ für jsp-dateien aufzustellen, die dem Code eine einheitliche Struktur verliehen hätten und dadurch die Lesbarkeit erhöht hätten. Oft fehlte uns während PSE die notwendige Zeit um einfachen, sauberen Code zu schreiben. Sobald der Code compilierte und funktionierte ging man dazu über, das nächste Requirement zu implementieren. Kurz gesagt, es war höchste Zeit für ein

Refactoring. Wir haben dies nun bei der Weiterentwicklung von Psystatix nachgeholt. Im Model gab es dabei nur kleine Änderungen, da der Code hier bereits relativ gut war. In der Darstellungsschicht hingegen wurde jedoch praktisch jede Datei neu überarbeitet und möglichst vereinfacht.

5.3.2 Einführung von Standards (HTML / CSS)

Zur Verbesserung der Qualität von Psystatix haben wir sämtliche HTML-Dateien dem "XHTML 1.0 Transitional"- Standard angepasst. Dies führt einerseits zu einer einheitlichen Strukturierung der HTML-Datein, andererseits zu einer korrekten HTML-Syntax. Ausserdem haben wir für die Darstellung der Webseiten konsequenter auf CSS gesetzt.

5.3.3 Benützung eines Logging-Frameworkes

Es ist allgemein bekannt, dass sich Logausgaben beim Debuggen einer Applikation als sehr hilfreich erweisen. Deshalb entschieden auch wir uns für die Verwendungen eines Logging-Frameworks. Wir wählten dabei das etablierte Log4J-Framework aus. Wir haben Log4J jedoch nicht nur für Debug-Zwecke benützt, sondern auch um eine Art Logbuch getätigter Aktionen zu führen. So speichert Psystatix mit Log4J beispielsweise jedes Login in eine Datei ab.

```
INFO 19 Apr 2005 09:10:13,249 TP-Processor11 login - Erscheinen auf Loginseite, Quelle: 80.219.5.78 Browser: Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1)
INFO 19 Apr 2005 09:10:21,912 TP-Processor11 login - Admin Frank Wettstein hat sich eingeloggt.
```

Abbildung 9: Auszug aus der Datei login.log

5.4 Dokumentation

5.4.1 Strukturierung

Um eine einheitliche Struktur in der Dokumentation zu erhalten und die Erstellung von Dokumenten zu formalisieren, haben wir eine Vorlage (s. Anhang) erstellt. Die so erstellten Dokumenten sollen nicht nur dokumentieren, wie Teile von Psystatix funktionieren, sondern auch helfen den Verlauf der Implementation festzuhalten, da nach jeder grösseren Änderung eine neue Version des entsprechenden Berichts geschrieben wird.
Das Ziel dieser Strukturierung der Dokumentation war es einerseits, sie einheitlicher und dadurch für den Leser verständlicher zu gestalten, andererseits das Erstellen der Dokumente für uns zu erleichtern, so dass mehr Dokumentationen geschrieben würden. Dies hat sich bewährt, denn durch das Vorhanden sein einer Vorlage, denkt man eher daran, dass noch ein Dokument erstellt werden soll, und es ist auch klar welche Informationen es enthalten muss, so dass weniger vergessen wird.

5.4.2 ER/UML

Zur Dokumentation der zweiten Entwicklungsphase haben wir ein ER-Diagramm der Datenbank und ein UML-Klassendiagramm des Models erstellt (s. Anhang). In der Datenbank sind einige Änderungen klar ersichtlich. So wurden neue Tabellen erstellt, um die Verbindung zwischen Termini und ihren Dokumentationsquellen zu speichern. Im Model hingegen hat sich nicht viel geändert. Hier gibt es einige neue Attribute und in einigen Klassen neue Methoden. Allerdings mussten einige schon bestehende Methoden durch neue Anforderungen abgeändert werden.

5.4.3 Hilfe-Dokument

Da jeder Benutzer eines Programms froh ist, wenn er eine Hilfe hat, in der steht, wo welche Funktionen zu finden sind, haben wir eine kleine Hilfsdatei in unser Programm integriert. Dies dient auch als Dokumentation, da hier ersichtlich wird, welche Funktionalitäten das Programm hat, und wie das User-Interface aufgebaut ist. Ausserdem hilft ein solches Dokument dem User bei der Orientierung.

5.4.4 Installationsanleitung

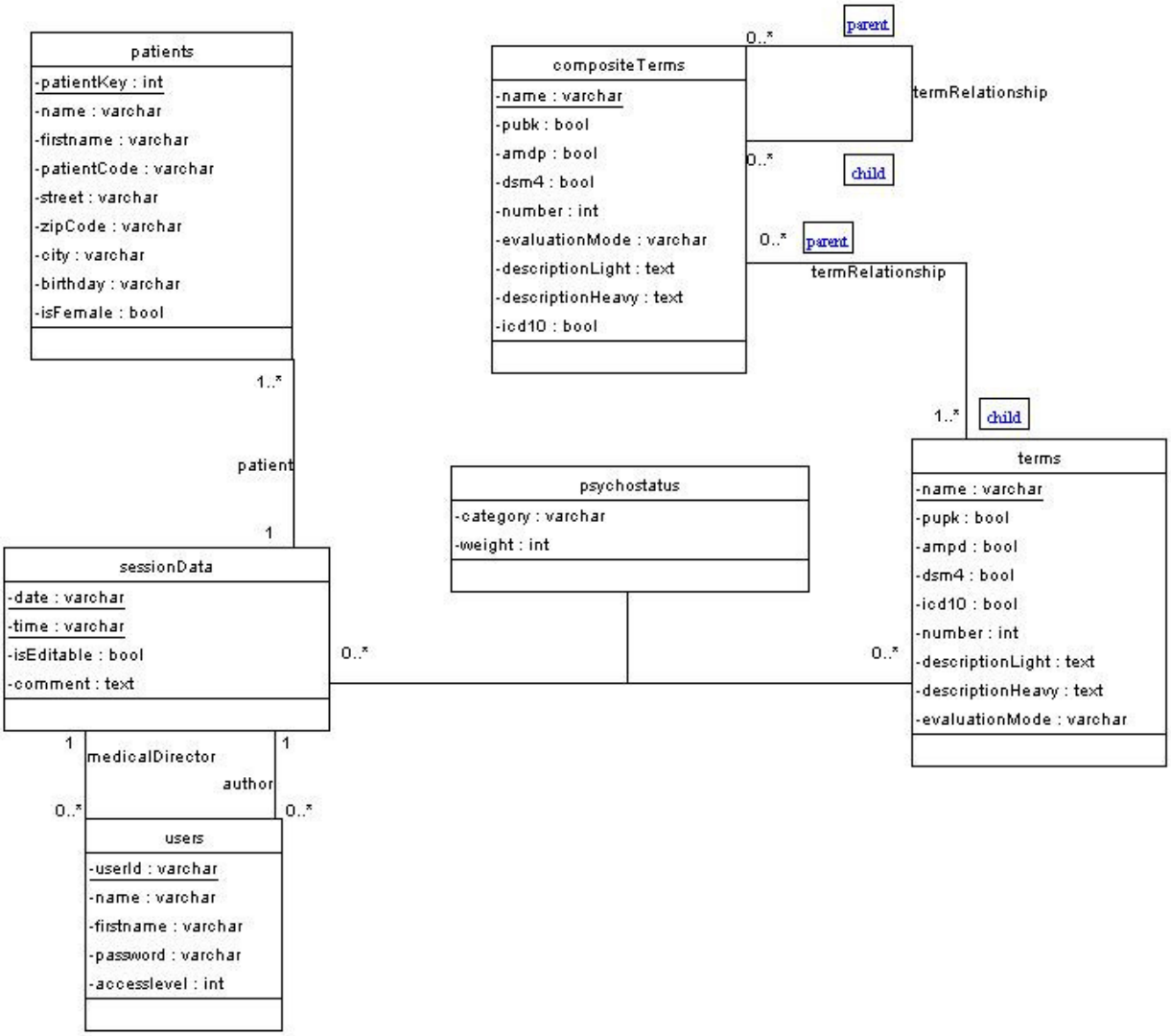
Da unser Programm viele verschiedene Hilfsprogramme (Tomcat, MySQL, Java VM) braucht, ist eine Installation nicht so einfach. Insbesondere müssen MySQL und Tomcat richtig konfiguriert werden, damit die Verbindung mit Psystatix funktioniert. Um die Installation zu erleichtern, haben wir eine Installationsanleitung verfasst, welche genau erklärt, wie man die einzelnen Programme einrichten muss und wie Psystatix installiert wird. Hierbei haben wir uns aus Zeitgründen darauf beschränkt die Installation für die von uns mitgelieferten Versionen von Tomcat, MySQL und Java auf Windows XP zu beschreiben. Diese Wahl trafen wir, weil Psystatix, obwohl es plattformunabhängig ist, vorläufig auf Windows XP betrieben werden soll.
Ein weiterer Vorteil einer Installationsanleitung besteht darin, dass hier die Schnittstellen zwischen den Programmen dokumentiert sind. Durch die Dokumentation der Installation hoffen wir auch die Probleme, welche wir beim erneuten Aufsetzen von Psystatix hatten, beheben zu können, so dass bei einer allfälligen Weiterentwicklung die Installation schneller abgewickelt werden kann.

5.4.5 Ausblick

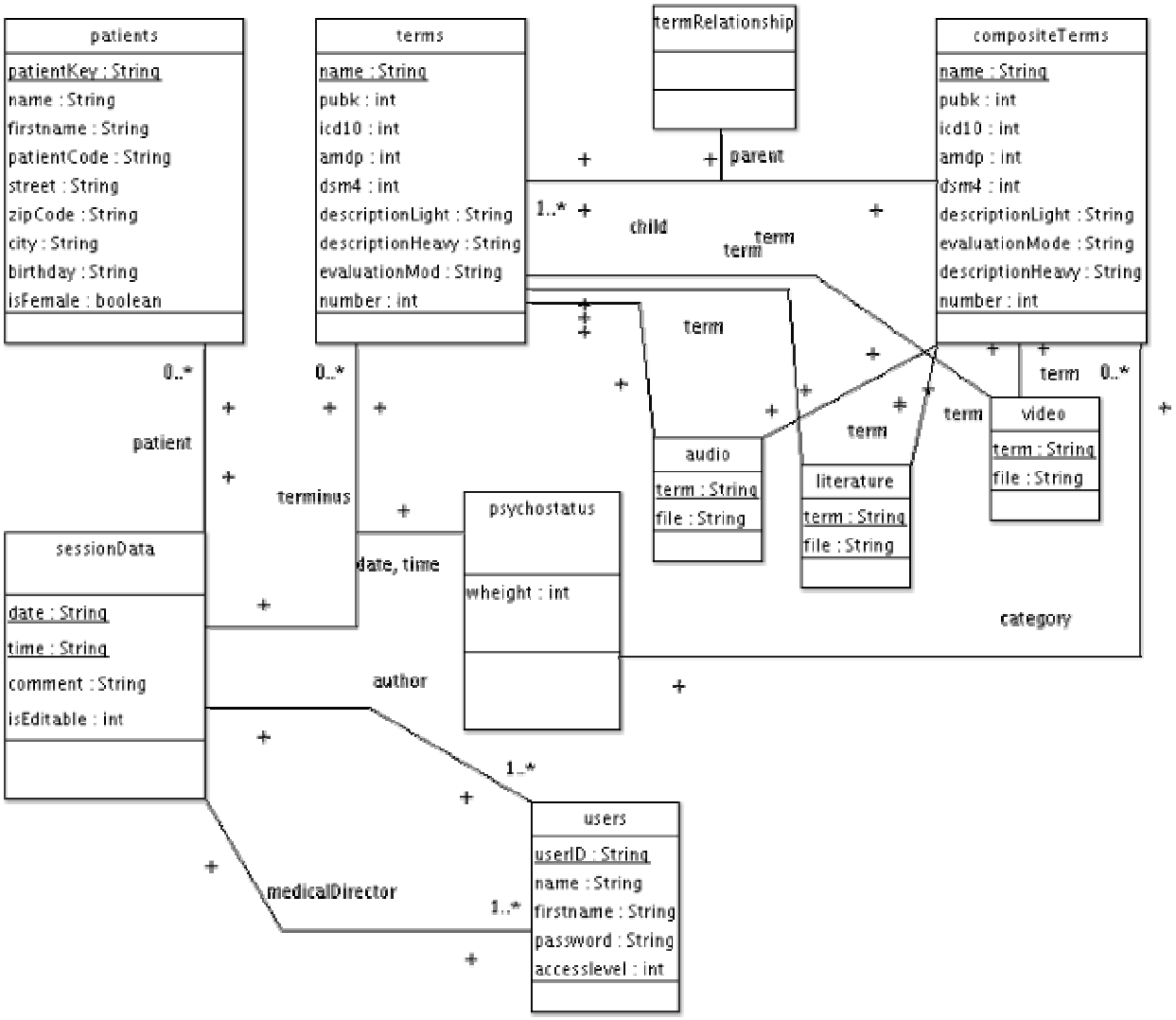
Wir haben nun eine weitere Phase in der Entwicklung von Psystatix abgeschlossen. Unser Programm erfüllt die wichtigsten Anforderungen aus Sicht des Kunden und realisiert alle gewünschten Primärfunktionen. Allerdings ist ein solches Programm nie wirklich fertig. Es gibt noch einige Dinge, die unser Programm und seine Anwendbarkeit verbessern würden. Es war uns nicht möglich im Rahmen unseres Projektes ein ausführliches Sicherheitskonzept für Psystatix zu erarbeiten. Da in dem Programm Daten von Patienten gespeichert werden, welche nicht frei zugänglich sein sollten, wäre aber ein gutes Sicherheitskonzept wichtig. Um unbefugten Zugriff auf Daten zu verhindern, muss Psystatix in der aktuellen Version auf einem gesicherten Netzwerk, welches den Zugang von aussen limitiert, betrieben werden. Ein weiterer Punkt welcher für den Kunden sehr interessant ist, wäre die automatische Generierung von Diagnosen. Allerdings ist dies nur mit grossem Aufwand zu realisieren, und war deshalb für uns im bisherigen Rahmen nicht machbar.
Dies und sicherlich noch vieles mehr könnte in einer Weiterentwicklung getan werden. Wir hoffen, dass wir in Zukunft weitere Gelegenheiten erhalten unser Programm zu verbessern und zu erweitern, so dass die Psychiater im Inselspital ein möglichst vollständiges Programm, erhalten, welches ihnen die Arbeit erleichtert.

Anhang

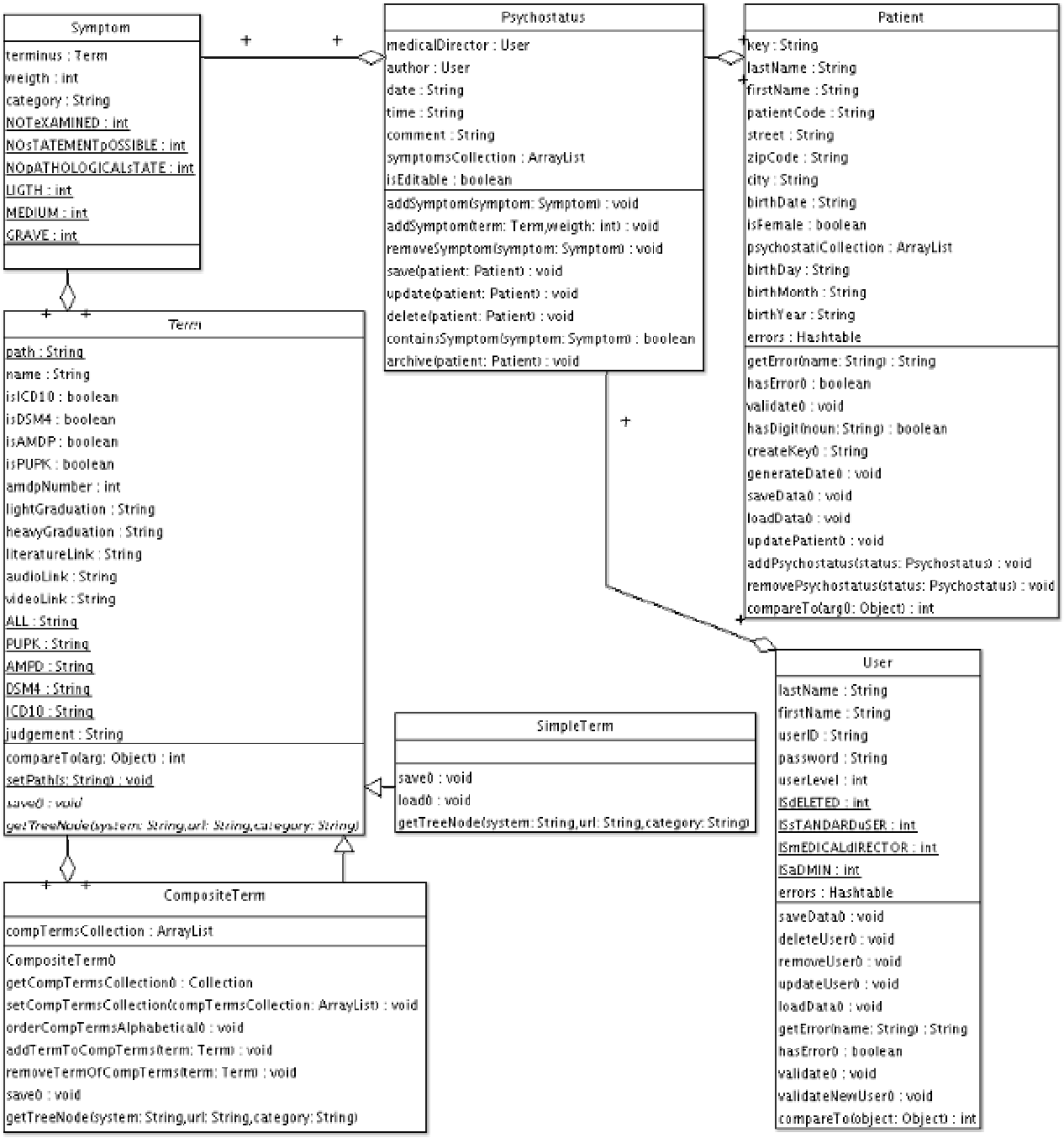
1. ER-Diagramm der Datenbank nach PSE



2. ER-Diagramm der Datenbank nach Phase 2



3. UML-Klassendiagramm



Da es keine Änderungen in der Struktur des Models gab wird hier auf die Darstellung des Models nach PSE verzichtet.

1. Beispiel einer Dokumentation nach Vorlage

Ant

Version: 2.0
Änderungen gegenüber der ersten Version:

- Erzeugen einer binären Distribution der Applikation
- Targets zur Bedienung von Tomcat enthalten
- TODO: Einbindung von junit Tests

Beschreibung:
Damit die folgenden targets funktionieren, muss die Datei /server/lib/catalina.ant.jar von Tomcat 5 in das Verzeichnis „lib“ von ant kopiert werden.

Folgende Ant-Targets stehen zur Verfügung:

ant **all**	führt **clean** und **compile** aus, zwingt also zu einer vollständigen Kompilierung
ant **clean**	löscht die Ordner „dist“ und „build“
ant **compile**	führt zuerst **prepare** aus und kompiliert anschliessend sämtliche Dateien im Ordner „src“
ant **dist**	erzeugt eine binäre Distributation der Applikation. Vorgängig werden **compile** Und **javadoc** ausgeführt.
ant **install**	installiert die Applikation dynamisch auf dem Tomcat.
ant **javadoc**	erzeugt eine Javadoc API für sämtliche Java-Klassen
ant **list**	listet sämtliche Applikationen auf, die auf dem Tomcat installiert sind
ant **prepare**	erzeugt die verzeichnisstruktur von „build“, kopiert ausserdem statische Dateien vom Ordner „web“ in „build“
ant **reload**	zwingt Tomcat zu einem Neustart. Dies ist nützlich, falls Klassen geupdatet wurden oder neue jar eingebunden werden.

5. User Stories

Nr	Story	Priori-tät	Abh. von	Zeit View	Zeit Model	Zeit Total
1	Einloggen als normaler User oder als Administrator	1	-	2	3	**5**
2	Eingabe aller erforderlichen Patienten- und Sitzungsdaten	1	-	6	5	**11**
3	Suche nach einem schon vorhandenen Patienten (mittels Kennnummer oder zum Beispiel Name) und Möglichkeit, an diesem Änderungen vorzunehmen.	2	2	6	6	**12**
4	Möglichkeit an einem Patienten Änderungen vorzunehmen(Adressänderung)	2	2	3	10	**13**
5	Wahl eines Klassifikationssystems	3		3	8	**11**
6	Wahl der Kategorie	1	5	5	8	**11**
7	Auswahl der Termini	1	5,6	20	8	**28**
8	Gezielte Suche nach Synonymen von Termini beim Erstellen des Psychostatus	4	5,6	15	6	**21**
9	Übersicht über die gewählten Termini, also der eigentliche Psychostatus	3	5,6,7	3	6	**9**
10	*Vorschläge von verschiedenen Diagnosen und Möglichkeit eine zu wählen*	*5*				
11	*Für jede Diagnose kann man erfahren, welche Termini zu ihr gehören und welche man tatsächlich ausgewählt hat.*	*5*				
12	Eingabe eines zusätzlichen Kommentars möglich	4	9	1	18	**19**
13	Möglichkeit das Klassifikationssystem zu wechseln -> Vorschläge für Termini	3	5,6,7	15	12	**27**
14	Abspeichern als Word Dokument	2	5,6,7	12	0	**12**
15	Bei bereits vorhandenen Patienten kann man die alten Psychostati (oder das ganze Dokument?) anklicken und anschauen.	4	14	6	12	**18**
16	Ein alter Psychostatus eines Patienten kann nachträglich geändert werden.	5	15	3	15	**18**
17	Home-Seite	1	-	10	0	**10**
18	DB aufsetzen	1	-	0	12	**12**
19	Daten aus DB in Model (Collections) laden	1	18	0	35	**35**
20	Speichern des Psychostatus in der DB	2	5,6,7	1	10	**11**
21	Jeder User kann "Erklärungen" anwählen und darauf von Hand ein Wort eingeben und danach suchen. Es erscheint eine Erklärung samt Synonymen, Zugehörigkeit zu welchen Klassifikationssystemen und Kategorien.	4	19	12	10	**22**
22	Jeder User kann zum Suchen das Klassifikationssystem wählen, dann die Kategorie uns schliesslich den Begriff und dann auf Search klicken. Darauf erscheint eine Erklärung samt Synonymen, Zugehörigkeit zu welchen Klassifikationssystemen und Kategorien.	4	19	12	10	**22**
23	Ein Administrator kann neue Termini hinzufügen. Dazu muss er verschiedene Angaben machen: Begriff nennen, Klassifikationssystem(e) wählen, Kategorie wählen, Erklärungen zum Begriff eingeben. Auf der nächsten Seite wählt er Synonyme aus, die schon im System abgespeichert sind. Dazu kann er jeweils zuerst die Kategorie des gesuchten Synonyms anwählen um schneller zum Ziel zu kommen.	3	19	12	20	**32**
24	Ein Administrator kann die bestehende Termini löschen. Zuerst kann er das oder die Klassifikationssystem(e) anwählen, aus dem/denen der Terminus gelöscht werden soll. Darauf wird der Terminus markiert und dann auf den Lösch-Button geklickt.	3	19	12	10	**32**
25	Ein Administrator kann die bestehende Termini ändern. Es erscheinen die Angaben, die zu diesem Terminus bereits gemacht wurden Klassifikationssystem(e), Kategorie, Erklärungen und Synonyme. Diese Daten können nun abgeändert werden und es geht ganz analog zu oben weiter.	3	19	12	30	**42**

Nr	Story	Priori-tät	Abh. von	Zeit View	Zeit Model	Zeit Total
26	Ein Administrator kann eine bestehende Kategorie löschen. Er kann zuerst eines oder mehrere Klassifikationssysteme wählen und darauf die Kategorie markieren und so mit allen in ihr enthaltenen Begriffen löschen.	3	19	6	30	**36**
27	Ein Administrator kann eine Kategorie hinzufügen. Er gibt den Namen der Kategorie an. Er muss die Klassifikationssysteme wählen, zu welchen die neue Kategorie gehört.	3	19	5	15	**20**
28	Beim Hinzufügen einer neuen Kategorie können aus einer Liste von bereits abgespeicherten Termini, die der neuen Kategorie angehörigen gewählt werden.	3	19	8	15	**23**
29	Beim Hinzufügen einer neuen Kategorie besteht auch die Möglichkeit neue Termini in ein Textfeld einzugeben.	3	19	6	12	**18**
30	*Der Administrator kann einer Diagnose weitere Termini anfügen. Er kann die Diagnose wählen und dann über ein Textfeld einen neuen Begriff eingeben.*	5				
31	*Der Administrator kann Termini von einer Diagnose entfernen. Er kann die Diagnose wählen und dann einen der erscheinenden Termini löschen*	5				
32	Ein Administrator kann Benutzer hinzufügen. Es erscheint eine Seite mit einer Liste aller Benutzer. Dort kann auch ein neuer Benutzer hinzugefügt werden.	3	-	6	18	**24**
33	Ein Administrator kann Benutzer löschen. Es erscheint eine Seite mit einer Liste aller Benutzer. Von diesen kann einer ausgewählt und gelöscht werden.	3	-	6	15	**21**
34	Ein Administrator kann Benutzer bearbeiten(wenn zum Beispiel jemand sein Passwort vergessen hat). Es erscheint eine Seite mit einer Liste aller Benutzer. Von diesen kann einer ausgewählt und editiert werden (in den darunter liegenden dafür vorgesehenen Textfelder).	3	32	6	15	**21**
35	*Statistiken*	5				
36	Importfunktion für die Termini, Kategorien... (damit nicht alles von Hand eingetragen werden muss).	2	-	2	30	**32**

Erklärungen

- Prioritäten: 1 = sehr wichtig(muss so schnell als möglich umgesetzt werden)
  2 = wichtig
  3 = wichtig, aber pressiert noch nicht
  4 = eher unwichtig
  5 = nice to have
- Das Schräggedruckte ist für beide Seiten noch recht unklar und im Moment noch unwichtig(betrifft die Diagnosestellung und die Statistiken).