Abschlußbewertung : Planetenmanager (Team 3)

Teammitglieder:

Mandy Koplin, Mirko Schädel, Björn Schulz, Urs Möller

Vorlesungsbewertung

Vorlesung: (Björn)

Die Abarbeitung der Meilensteine empfand ich nicht als besonders hilfreich, denn wertvolle Zeit wurde dafür aufgewendet und unser eigener Produktionsfluss dadurch bestimmt. Zwei Zwischenpräsentationen und ein regelmäßiger Report hätte uns sicherlich mehr gebracht, zumal des öfteren die Besprechungszeit nicht für eigene Fragen ausreichte. Insgesamt ist der Umfang der Aufgaben zu groß. Desweiteren wären einige Vorlesungen in SWP I besser angesiedelt als in SWP II, z.B. "guter Programmierstil", "JUnit" und "Anforderungen an den Code". Der konsequente Ansatz "Test First" ließ sich dadurch nicht wie gewünscht realisieren. Weiterhin hilfreich wäre ein gesteigerter Erfahrungsaustausch in der Vorlesung. Oft tauchen gleiche oder ähnliche Probleme auf, mit denen auch andere Gruppe kämpften.
Es fehlte einfach an einer ausgereifteren Betreuung wärend der Übungszeit durch den Dozenten, die durch Rücksprachetermine ersetzt wurde.

Arbeitsbedingungen: (Björn)

Nur durch den gezielten Einsatz eigener Mittel ist ein sinvolles Arbeiten überhaupt möglich geworden. Ein Mitglied unserer Gruppe stellte uns dafür einen eigenen Server zur Verfügung und uns wurde eine permanente Interverbindung kostenlos bereit gestellt. Dadurch waren wir in der Lage ein eigenes Forum, CVS, Datenbank und einen eigenen Tomcat zu betreiben. Das effektive Arbeiten ist in den uns zugewiesen Labor nicht möglich gewesen. Der stark beschränkte Platz und das erzwungene Arbeiten auf dem Terminal-Server sind der Effektivität sehr abträglich. Der Zugang zu anderen (besser ausgestatteten und mit einem besseren Konzept versehenen) Laboren, blieb uns nur über private Kontakte erhalten und ermöglichte uns einen wesentlich effizienteren Arbeitsstil. Die Wahl der geeigneten Umgebung für so ein großes Projekt, sollte durch den Dozenten sinnvoll bestimmt werden.

Technologiebewertungen

Java 5 (Björn)

Obwohl es oft sehr sinnvoll gewesen wäre die neuen Möglichkeiten von Java 5.0 zu nutzen, z.B. die Generics, finde ich den Verzicht darauf sinnvoll. Denn die verwendeten Tools sind bei weitem noch nicht kompatibel genug und es wären mehr Probleme aufgetreten hinsichtlich der Verwendung der neuesten Java-Version.

Struts als Framework: (Urs)

Struts ist ein sehr gutes Werkzeug bei der Entwicklung der View-Controller-Komponenten einer Architektur. Durch die Möglichkeit, dynamisch konfiguriert die Ziel-Jsps der Actions anzuzeigen gewinnt man viel an Flexibiltät. Auch die relativ einfache Möglichkeit, ActionForm-Klassen deklarativ zu erzeugen, ist von großem Wert und hält die Erfassung von Eingabedaten flexibel. Meiner Meinung nach gibt es aber wie in den meisten Frameworks noch ein paar Schwachpunkte. So denke ich nicht, dass die Trennung von Design und Logik einer JSP-Seite wirklich gut trennbar

ist denn die Anzeige von FormBeans und die Integration komplexerer Formulare erfordert trotzdem Kenntnisse von Struts. Auch geht Struts davon aus, dass ein Benutzer stets linear die Navigationselemente auf einer Seite nutzt, sobald aber auch der Aspekt der Vor-Zurück-Knöpfe im Browser ins Spiel kommt hört die Framework-Unterstützung. Hier hätte ich mir mehr erhofft.

Hibernate als Framework: (Urs)

Bis auf einige speziellen Gemeinheiten halte ich Hibernate für ein gut nutzbares und sehr reifes Projekt. Besonders die automatische kaskierende Speicherung von Relationen vereinfacht vieles. Durch Lazy-Loading kann auch viel Performance gewonnen werden, eine eigene Reflection/JDBC-basierte Persistenz-Schicht wäre sicher mühevoller gewesen.

Hibernate (Mirko)

Für das O/R Mapping verwendeten wir Hibernate, da es einerseits großen populären Anklang in der Java Community gefunden hat sowie mit der EJB3 Spezifikation verschmilzt und damit eine zukunftssichere Technologie darstellt. Außerdem sammelte ich bereits im Praktikum Erfahrungen mit Hibernate, die nun vertieft werden konnten. Hibernate bietet recht umfangreiche Möglichkeiten zum Abfragen von Objekten durch seine ausgeprägte Query-Language (HQL) und hat in dieser Hinsicht etwas mehr Funktionalität als JDO.
Das Hibernate-Mapping haben wir anfänglich versucht mittels Xdoclet generieren zu lassen, jedoch traten bei der Anwendung von Xdoclet einige Fehler auf, sodass wir die Mapping Dateien von Hand erstellt haben. Diese Mapping-Dateien haben wir auch während der Entwicklungsphase noch geringfügig angepasst um Feintuning durchzuführen.
Die Dokumentation von Hibernate durch die Referenz reicht für einen Einstieg aus, jedoch nicht für tiefergehende Fragen, da benötigt man weiterführende Literatur, bspw. „Hibernate in Action“ von Gavin King. Oftmals sind in der Hibernate-API allerdings einige Methoden nicht ausreichend dokumentiert.
Die Mapping Dateien sind sehr fein granulierbar und bieten für fast jeden Anwendungsfall Einstellmöglichkeiten. Allerdings sollte man bei Verwendung von Standardmappings auch aufpassen welche Datentypen dabei für die Datenbank verwendet werden bzw. welche Länge sie bei Zeichenketten haben. So wird beispielsweise ein java.lang.String auf einen varchar(255) bei Hibernate/MySQL gemappt. Dies kann zu Schwierigkeiten führen, wenn man längere Zeichenketten verarbeiten möchte. Dieses Beispiel sollte nur einen Aspekt liefern, dass dies früh bedacht werden sollte.
Da wir Java 1.4 einsetzten und Hibernate 3 für unsere Anforderungen keine wesentlichen Vorteile brachte, verwendeten wir die Version 2.1.
Die Verwendung von Lazy Loading bei Hibernate für einige Assoziationen fiel uns recht schwer, weil dies natürlich voraussetzt, das entsprechend lange der Zugriff auf die Datenbank, gekapselt durch die Hibernate Session, notwendig ist. Die Folge davon ist, das die Session entsprechend lange offen gehalten werden muss, dies führt zu einer schwierigen Design-Entscheidung, wo das Öffnen und Schließen einer solchen Session erfolgt und man in der entsprechenden Literatur keine zufriedenstellenden Lösungen findet.

Projekterfahrungen

Arbeitsumfang (Urs)

Neben den großen Aufgabenpaketen Architekturentwurf und Technologiewahl bleibt die Anforderung, auch die entsprechenden Klassen zu erzeugen, bestehen. Und die Architektur vergrößert die Anzahl der Klassen. Das führt zu einem größeren Aufwand als man ihn zunächst einschätzt. Insgesamt war der Aufwand für eine Lehrveranstaltung mit Übung eindeutig zu hoch. In dem Projekt ein Aufwand von durchschnittlich zwei Arbeitstagen pro Woche von vier Personen.

Nutzen von Codegeneratoren (Urs)

In einer vollständig ausgearbeiteten Architektur gibt es eine große Anzahl von Klassen, die sich durch automatische Generierung erzeugen lassen würden. Ich will das kurz anhand eines anderen Projekts skizzieren: In meinem J2EE-Projekt habe ich die Möglichkeit, aus den persistenten Entity-Bean-Klassen (fast) alle weiteren benötigen Klassen zu erzeugen. Über Xdoclet erhalte ich ValueObjects, DTOs, Lookup-Manager, Home-und Remote-Interfaces. Diese Möglichkeiten zur Generierung finde ich großartig, da meine persistenten Klassen auch gleich Methoden zur Erzeugung oder Speicherung von den Value-Objects besitzen. Zwar ist die Steuerung von Xdoclet nicht trivial aber da ich auch sehr viel an Code gewonnen wäre ein Einsatz auch hier sinnvoll gewesen. Eine weitere Möglichkeit, die ich im Nachhinein gerne genutzt hätte ist die Generierung der Struts-ActionForm-Klassen.

Fazit: Es gibt eine große Anzahl an Bibliotheken (besonders Apache-Commons-Projekt), die die Arbeit erleichtern und noch Möglichkeiten zur Optimierung der Arbeitsprozesse (Codegenerierung, bessere Modellierungswerkzeuge). Eine Ansatz, der mir in der LV gefehlt hat war Model-Driven-Architecture, die Aussicht aus einem optimierten Modell Code zu generieren finde ich sehr verlockend.

Allgemein / Teamarbeit (Mirko)

Die Teamarbeit in unserem Team lief insgesamt gut, die Implementierungsphase kommt meines Erachtens zeitlich viel zu kurz, die Phase sollte eventuell schon in SWP I begonnen werden, damit die Gruppen zum Schluss nicht in zeitliche Bedrängung kommen.
Für einige Meilensteine war sehr viel Aufwand nötig, da die Einarbeitung in die entsprechenden Techniken länger als erwartet dauerte sowie die Praxis oft von der Theorie abweicht und die Fehlersuche sehr viel Zeit in Anspruch nimmt.
Für ein erneutes Projekt würde ich die Aufteilung der Zuständigkeiten anders bzw. klarer wählen, da bei uns doch verschiedene Projektmitglieder in mehreren Teilen mitgewirkt haben und es somit zu erhöhtem Abstimmungsaufwand kam.
Die Verwendung des CVS brachte viele Vorteile und funktionierte insgesamt gut. CVS hat dabei den großen Nachteil, dass es bei Umbenennen/Verschieben/Löschen von Dateien oder Ordnern einige Probleme hat und dies zu inkonsistenten Zuständen im Projekt führte.

Resume (Mandy)

Das Softwareprojekt als Ganzes fand ich eine sehr interessante Erfahrung, sowohl in der Teamarbeit als auch im Erstellen eines vollständigen Projektes. In beiden Bereichen haben ich dieses Semester viel gelernt und für zukünftiges Arbeiten mitgenommen.

Die Aufteilung des Projektes in zwei Semester ist auf jeden Fall notwendig, aber dennoch zu wenig. Im ersten Semester hielt sich der Arbeitsaufwand in Grenzen. Um so größer wurde dieser Aufwand im 2. Semester in der Implementierungsphase. Die Einarbeitung in verwendete Technologien und Architekturen nimmt einen großen Teil des Semesters ein. Die Semesterferien als Einarbeitung reichen bei weitem nicht aus, zumal die verwendeten Technologien und Architekturen zu zahlreich und zu umfangreich waren. Zudem ist es ein großer Unterschied, an einem Testprojekt oder am eignen "echten" Projekt zu arbeiten.

Was die Teamarbeit anging, so fand ich, verbraucht das Koordinieren des Teams, der zu vergebenen Arbeit und das Zusammenfügen dieser viel Zeit. Mit unserem 4-Mann-Team waren wir noch "unterbesetzt" und haben trotzdem viel Zeit mit Besprechungen verwendet. Anhand anderer Projekte mit kleineren Gruppen ist schon merkbar, wie groß der Teamaufwand mit "nur" 4 Leuten wurde.