Schlussbewertung

Das Projekt MonsterMail war in unserem bisherigen Studium das umfangreichste und arbeitsintensivste Projekt, nicht nur bezüglich der vielen verschiedenen eingesetzten Technologien, sondern auch bezüglich der Anzahl der Projektteilnehmer. Da eines der Teammitglieder nur im ersten Semester des Softwareprojekts mitarbeiten konnte und daraufhin ein Ersatz eingesetzt wurde, war eine Einarbeitungsphase notwendig. Der Quereinstieg in ein bereits fortgeschrittenes Projekt in so kurzer Zeit ist nicht einfach, dennoch konnte die neue Teambesetzung sich arrangieren und alle geplanten Funktionalitäten umsetzen. Das Ausscheiden des Teammitglieds aus dem vergangenen Semester bedeutete Abfluss von Wissen über das Projekt.

Zur Organisation unseres 5-köpfigen Teams wurden ein Forum zur Diskussion von Fragen und ein Bug-/Issue-Tracking-System (Anfangs ProjectPier, später Eventum) eingesetzt und zusätzlich ein wöchentliches Meeting durchgeführt. Intensive Kommunikation wurde ein zentraler Punkt in der Zusammenarbeit, sodass wir oft auch während der Entwicklung einzelner Programmbausteine über Online-Chat miteinander kommunizierten.

Bei der Aufgabenverteilung konnten wir die einzusetzenden Technologien nicht immer konkret einer Person zuordnen. Insbesondere zu Beginn des Projektes waren nahezu alle Teammitglieder mit der Ausarbeitung der Grundarchitektur befasst. Dies bedeutete allerdings, dass erst zum Ende des Projektes wichtige Technologien bearbeitet und ins Projekt integriert wurden, was einen gewissen Zeitdruck verursachte. Vorteilhaft dabei war allerdings, dass sich mehrere Teammitglieder zu einer Technologie informiert haben und sich über die Verwendung dieser austauschen konnten. Oft wurden die Ergebnisse zu einer Technologie in den Teambesprechungen diskutiert, sodass alle Teammitglieder einen Einblick bekamen.

Ein besonderer Vorteil war, dass die Usability der Nutzeroberfläche bereits im vergangenen Semester im Modul Human Computer Interaction analysiert und optimiert wurde. Eine intuitive, benutzerfreundliche Oberfläche entstand. So konnten die entstandenen Entwürfe ohne zeitaufwendige Umgestaltung für das Design verwendet werden.

Datenbankanbindung

Gelernt haben wir während des Projektes viel. Wir haben gesehen, wie die verschiedenen Komponenten einer Architektur zusammengefügt werden können und wie besser nicht. Speziell unsere Datenbanktransaktionssteuerung hätte anders implementiert werden müssen. Zu spät haben wir erkannt, dass das Zentralisieren der Transaktionssteuerung auf Requests des Anwenders (und damit auf die vom Apache-Wicket-Framework bereitgestellten Submit-Methoden) den Handlungsspielraum stark einschränken, besonders bezüglich des Nachladens von Objekten (lazy-fetching) aus der Datenbank. Auch die JavaScript-Bestandteile wurden eingeschränkt, da bei diesen keine Submit-Methode aufgerufen wird.

Für die Datenbankanbindung wurde das Generic-DAO-Pattern genutzt. Dieses Pattern war anfangs schwer zu verstehen, aber letztendlich sehr sinnvoll und löste die Anbindung unterschiedlicher Logikklassen an die Datenbank sehr elegant. Zusätzlich erlaubt dieses Pattern die Applikation schnell um neue Objekte zu erweitern, denen durch die Generizität bereits die Grundmethoden des Datenbankzugriffs bereitstehen. Nur objektspezifische Zugriffsmethoden müssen in einer abgeleiteten Klasse implementiert werden, was den Aufbau der Datenbankklassen sehr gut strukturiert und übersichtlich gestaltet.

AspectJ

AspectJ bietet eine sehr komfortable Möglichkeit wiederholende Zustände/Aufgaben zu umweben. Wir nutzen dies für die Transaktionssteuerung und die zentrale Ausnahmebehandlung mit MulTEx.

Dabei hat AspectJ einen großen Vorteil bewirkt. Es mussten keinerlei Anpassungen am Quellcode der Oberflächen vorgenommen werden.

Mit seiner einfachen Steuerung wird man schnell dazu verleitet viele Funktionalitäten außerhalb des eigentlichen Codes zu steuern. Dadurch haben die Entwickler keinen direkten Überblick an welchen Stellen der Aspect greift und dabei die Funktionalität beeinflusst. Um den Aspect zu visualisieren benutzen wir das

AspectJ-Eclipse-Plugin. Dies zeigt, durch einen farbigen Pfeil am Rand, an welcher Stelle ein Aspect eingreift.

Da der Aspect bei der Kompilierung in den Code eingewoben wird, erscheinen zur Laufzeit auf der Konsole Fehlermeldungen mit Zeilennummern, die im Quellcode nicht zu finden sind. Teilweise kann erst durch aufwendiges Debuggen der eigentliche Fehler gefunden werden.

In unserem Projekt hatten wir damit selten Probleme, aber es ist eine nicht zu unterschätzende Auswirkung. Daher sollten Aspects eher sparsam benutzt werden.

MulTEx

MulTEx ist ein Framework zur zentralen Ausnahmebehandlung mit internationalisierten und parametrisierten Meldungstexten. Das Framework hilft den heutigen Anforderungen an eine Ausnahmebehandlung gerecht zu werden.

Internationalisierte Fehlermeldungen sind ein gefordertes Muss. Für die Übersetzung werden ResourceBundles verwendet, dabei ist es sehr hilfreich, dass dem Entwickler, die zu übersetzende Properties-Datei generiert wird. Dafür musste Ant in Maven konfiguriert werden. Somit können sehr leicht nachträglich weitere Sprachen in die Applikation eingebracht werden, ohne an der Ausnahmebehandlung Änderungen vornehmen zu müssen.

Durch die Zentralisierung kann der Meldungstext an ein „festes“ Ausgabemedium weitergegeben werden. Bei uns ist das die Wicket FeedbackBox.

MulTEx selbst funktioniert einwandfrei. Leider sind die Dokumentationen und die Kommentierungen teilweise veraltet, oder machen das Vorgehen nicht deutlich klar. Mit Unterstützung des Entwicklers Prof. Knabe war es jedoch möglich MulTEx für unser Projekt nutzbar zu machen und gewinnbringend einzusetzen.

JUnit

JUnit ist ein sehr hilfreiches Test-Framework. Wir haben zum größten Teil White-Box-Tests geschrieben, dabei werden die Ergebnisse getestet sowie die innerhalb einer Funktion verursachten Veränderungen.

Um die Tests zu schreiben und aktuell zu halten, musste sehr viel Zeit aufgewendet werden. Vor allem da für jede einzelne Methode und ihre möglichen Exceptions vorher ein bestimmter Zustand hergestellt und nachträglich wieder rückgängig gemacht werden muss.

Durch die Implementierung der Test-Methoden konnten teilweise nicht bedachte Zustände in der Logik aufgedeckt werden. Auch können nachträgliche Änderungen an der Logik schnell und einfach auf Funktionsfähigkeit und eventuelle Seiteneffekte getestet werden.

Wicket

Ein großer Teil der Projektzeit fiel der Entwicklung unter Wicket zu. Anfangs schien die Technologie noch etwas träge und aufwendig. Vieles wirkt immer noch redundant, aber eine so konsequente Trennung von Design und Logik ist selten. Das und die enge Anbindung an Java wird Wicket in Zukunft wahrscheinlich noch einiges an Aufwind bescheren.

Wicket wirkt fundierter und leistungsfähiger als bspw. JSP. Besonders der Anfang gestaltete sich allerdings sehr schwer, da man schon die simpelsten Dinge nachschlagen muss. Ein Link ist niemals einfach nur ein Link. Er kann nicht direkt mit einem Text versehen werden und trägt meist eine Klassendeklaration mit sich herum. Das erscheint nicht sofort als schlüssig. Diese Komponente sei nur stellvertretend für viele andere Komponenten angesprochen.

JavaMail

Für die Kernfunktionalität unseres Projekts, das Versenden und Empfangen von E-Mails, haben wir JavaMail verwendet. Mit JavaMail zu arbeiten ist sehr bequem, da keine Socket-Programmierung benötigt wird.

AES-Verschlüsselung und SHA-1-Hashverfahren

Die beide Technologien wurden für die Passwortverschlüsselung und das Validierungssystem eingesetzt. Da Java immer aktuell gehalten wird, was solche Algorithmen betrifft, ist es einfach von einem Hash-Verfahren zum anderen zu wechseln. Leider ist die Generierung der Schlüssel etwas umständlich.

Arbeitspensum

Letztendlich investiert man unwahrscheinlich viel Arbeit in das Softwareprojekt. Die Arbeitslast orientiert sich zwar immer an der Projektgröße und man könnte sich schon am Anfang gut betten, um am Ende entsprechend bequem zu liegen. Aber wenn man versucht, das Projekt mit einem gewissen Hang zur Perfektion abzuschließen, landet man sehr schnell im überdurchschnittlichen Stundenbereich. Oft hängt man sich auch schnell an Kleinigkeiten auf.

Im Gegensatz zu den meisten anderen Projekten, die wir während des Studiums durchgeführt haben, war die Betreuung im Softwareprojekt sehr gut. Durch regelmäßige Rücksprachen konnten Fragen schnell geklärt werden und man arbeitete nicht wochenlang in die falsche Richtung, bevor man eine Fehlentscheidung bemerkte. Die Meilensteine motivierten uns das Projekt stetig voranzutreiben und auch die wöchentlichen Teamsitzungen halfen in kleinen Schritten vorwärts zu gehen. Die Arbeit im Team war recht stimmig und wird an diesem Projekt mit Semesterschluss garantiert nicht zu Ende sein.

Auf das entstandene Projekt sind wir sehr stolz, denn nach so viel Arbeit ist doch mehr entstanden als wir vor einem Jahr zu den ersten Anfängen erwartet hätten.