ABSCHLUSSBEWERTUNG

FÜR DAS BUG TRACKING SYSTEM PESTCONTROL

AN DER BEUTH HOCHSCHULE FÜR TECHNIK BERLIN IM RAHMEN DES MODULS SOFTWARE PROJEKT II

Datum 26. Januar 2011
Version 1.0

Team FIX IT

Philipp Liepert - 757952
Philipp Haase - 758378
Patrick Böhm - 758036
Max Schwaab - 757891
Fabian Engels - 753950

Inhaltsverzeichnis

TECHNIK

Java

Da uns Java aus dem Studienalltag sehr bekannt ist, viel uns der Einstieg in das Projekt mit dieser Programmiersprache äußerst leicht. Bei allen Teammitgliedern stellte sich ein nahezu routinierter Einsatz von Java ein, der auch dank der guten Entwicklungsumgebung Eclipse viel Spaß bereitete. Die Fehleranfälligkeit ist dank starker Typisierung und dem Java Garbage-Collector gering und auch die Lesbarkeit des Codes konnte ohne große Mühen gewährleistet werden.

Viele Frameworks und die sehr gute Dokumentation zu Java haben uns im gesamten Verlauf des Projektes unterstützt.

Wicket

Wicket war für alle Teammitglieder ein kompletter Neueinstieg, was vor allem zu Beginn des Projekts zu erheblichen Problemen führte. Mit der Zeit konnten wir uns jedoch mehr und mehr mit dem Framework anfreunden und lernten die Stärken und Schwächen von Wicket deutlich kennen.

Das moderne Webframework stellt für viele alltägliche Aufgaben bei der Entwicklung von Webapplikationen einfache Komponenten zur Verfügung, welche die Arbeitszeit deutlich verkürzen. Hier fällt jedoch auch die Schattenseite von Wicket auf. Sobald man versucht, die gegebenen Komponenten zu Verändern bzw. zu Erweitern, kommt man mit der Teils sehr Verschachtelten und komplexen Struktur der Komponenten nur schlecht zurecht.

Auch das Exception-Handling unter Wicket brachte unsere zuständigen Teammitglieder an den Rand der Verzweiflung, da verschiedene Module einfach noch nicht enthalten sind. Hier möchten wir jedoch erwähnen, dass sich Wicket zu diesem Zeitpunkt erst in der Version 1.5 befand und weitere Verbesserung geplant waren.

Ein weiteres Problem von Wicket ist die mangelhafte Dokumentation und die geringe Qualität von Tutorials. Schon Probleme, die sich im nach hinein als sehr einfach erwiesen, kosteten uns so viel Zeit.

EJB

EJB wurde in unserem Studium bisher nicht behandelt und schon nach kurzer Zeit wurde uns klar, welch große Wissenslücke wir durch das Verwenden von EJB in unserem Projekt geschlossen haben. EJB ist ein sehr mächtiges Framework, dessen Funktionsumfang wir schon nach relativ kurzer Einarbeitungszeit sehr gut benutzen konnten. Möglichkeiten wie das Prinzip der Remote-Interfaces oder der Dependency Injection haben vieles Vereinfacht und uns noch einmal verdeutlicht, welchen Wert EJB für uns hat.

Ein weiterer Pluspunkt für EJB ist die starke Unterstützung von Annotationen, die das Verwenden des Frameworks stark vereinfacht haben. Lediglich einige Konstrukte, wie das Verwenden von sehr vielen Interfaces, könnten etwas eleganter gelöst werden, was jedoch unseren durchaus positiven Eindruck von EJB keineswegs trübt.

JPA

Die Java Persistence API ist unserer Meinung nach eine sehr ausgereifte Schnittstelle für objektrelationales Mapping und bietet erstaunlich viele Möglichkeiten des Datenzugriffs. Leider ließen sich die Many-to-Many-Beziehung oder die bidirektionale Assoziation nicht richtig verwenden, wodurch wir auf Funktionen wie die Löschweitergabe verzichten mussten

Hibernate

Hibernate bietet eine gut zu nutzende Implementierung für JPA.

JUnit

Für das automatisierte Testen der Logik ist JUnit ein äußerst wertvolles Framework. Da wir uns jedoch in die verschiedenen Technologien bzw. Frameworks einarbeiten mussten, war ein Testgetriebenes Entwickeln mit JUnit nicht möglich. Tests wurden erst nachträglich den Funktionen der Klassen angepasst und unterstützten uns so kaum beim Projektverlauf.

Zusätzlich ist JUnit nicht geeignet für Userinterface-Tests und hat einen weiteren Nachteil in Verbindung mit J2EE, da die Beans nur in ihrem Container laufen und nicht ohne Weiteres getestet werden können. Aus früheren Erfahrungen mit JUnit wissen wir jedoch, welche Leistungen dieses Framework bietet, auch wenn es in unserem Projekt nur eine untergeordnete Rolle spielt.

MySQL

Die kostenlose Datenbank MySQL lief während des ganzen Projekts fehlerfrei und zeigte damit beeindruckend die Zuverlässigkeit von Open-Source Software. Dank XAMPP konnten wir MySQL auch zur lokalen Verwendung einwandfrei benutzen, was aufgrund einer Port-Sperre im Hochschul-Netz auch die wöchentlichen Abgaben im Programmierlabor gewährleistete.

WERKZEUGE

Eclipse

Unser komplettes Team ist sich einig: Eclipse ist die beste IDE für Java-Projekte! Durch die unglaubliche Vielfalt an Erweiterungen und Plugins ließ sich Eclipse an die Bedürfnisse von jedem einzelnen Teammitglied anpassen. Einige Plugins, wie Subclipse, m2eclipse, JAutodoc oder Checkstyle gehörten jedoch bei jedem zum Standard-Repertoire.

Mit den typischen Funktionen, die eine IDE bieten sollte, bietet Eclipse mit vielen verschiedenen Versionen eine Entwicklungsumgebung für eine Vielzahl an Sprachen. Hierdurch kann man beliebig zwischen seinen Eclipse-Versionen wechseln kann, ohne sich an eine andere IDE gewöhnen zu müssen.

Bei der Wahl der Plugins ist jedoch Vorsicht geboten, da manche Kombinationen von Erweiterungen Eclipse instabil werden lassen.

Enterprise Architect

Der Enterprise Architect ist ein solides Werkzeug bei Erstellen von Diagrammen verschiedenster Art und unterstützte uns während unseres Projekts vor allem in der Analyse- und Entwurfsphase. Leider ist die Bedienung teilweise umständlich und nicht sehr intuitiv. Insgesamt wirkt der Enterprise Architect veraltet, obwohl wir mit Version 7 ein relativ neues Exemplar verwendet haben. Die Autogenerierung von Code oder Operationsspezifikation waren willkommene Bestandteile, konnten aber nicht auf voller Strecke überzeugen, da nachträglich viele Anpassungen nötig waren.

SVN

Größere Softwareprojekte sind ohne eine Versionsverwaltung wie Subversion kaum denkbar. Zum Ende des Projekts haben wir über 1000 verschiedene Versionen erzeugt, viele Konflikte unter den gleichen Dateien aufgelöst und etliche Dateien von unterschiedlichen Teammitgliedern verschmolzen. Die von SVN abgenommene Arbeit ist enorm und möchte von keinem unserer Teammitglieder mehr vermisst werden. Vereinzelt kam es zu Kompatibilitätsproblemen, da wir mit Tortoise SVN, Subclipse und Subversive verschiedene SVN-Clients verwendet haben. Jedoch ließen sich die aufkommenden Fehler meist schnell beheben.

Maven

Mit Apaches Build-Management-Tool Maven hat uns dank Dependency- Source-Code- und JavaDoc Management vor allem bei der Analyse von verschiedenen fremden Bibliotheken geholfen. Das automatische Downloaden von Abhängigkeiten oder Source-Code erleichtert einem oft die Arbeit. Jedoch kam es mit Maven häufig zu Fehlern. Die Konfiguration erwies sich vor allem zu Beginn unseres Projektes als sehr Aufwändig. Das nutzen der Konsole für Maven ist dank des Eclipse-Plugins m2eclipse nicht mehr notwendig und sollte unserer Meinung nach heutzutage eher „eine Option für die Eingabe“ als „die Standard-Eingabe“ sein. Auch das nachträgliche aktivieren von Maven auf ein Eclipse-Projekt war äußerst

schwierig und nur durch unnötiges hin- und her-kopieren von Dateien möglich. Obwohl uns die Grundidee von Maven durchaus zusagt, gibt es hier noch deutlichen Optimierungsbedarf.

FAZIT

Nach beinahe einem Jahr, in dem wir uns mit dem Projekt Pestcontrol beschäftigt haben, blicken wir ein bisschen stolz auf unsere Leistung. Kaum jemand von uns hat sich zu Beginn des Projekts den Umfang vorstellen können, den unser Bugtracking System noch annehmen sollte.Trotz aller Schwierigkeiten, die uns vor allem bei der Umsetzung ereilten, konnten wir ein Lauffähiges Bugtracking-System mit verschiedenen Features erstellen, die wir von Anfang an geplant hatten. Manch eine Funktion, die wir uns noch gewünscht hätten, blieb jedoch aufgrund der zahlreichen anderen Hochschul-Projekte auf der Strecke.

Unser Team funktionierte einwandfrei und wurde mit dem Verlauf des Projekts immer stärker. Schnell wurde klar, wo welches Mitglied seine Stärken hatte und konnte so gezielt in diesen Bereichen seine Fähigkeiten unter Beweis stellen. Der enorme Wissenszuwachs den wir uns im Laufe des Softwareprojektes erarbeitet haben, hat uns alle beeindruckt.

Nachdem man am Vortag bis tief in die Nacht programmiert hat, um die Meilensteine noch einzuhalten, war um 8 Uhr Früh im Labor zu stehen ein ziemlich harter Brocken. Die wöchentlichen Abgaben und Besprechungen haben uns jedoch sehr geholfen das Projekt kontinuierlich voran zu treiben. Und zu guter Letzt sind die berüchtigten Dienstag Abende tatsächlich zu langen Nächten geworden, die niemand von uns vergessen wird.