Carl von Ossietzky Universität Oldenburg
Projekt Gruppe 2005 / 2006

31. Mai 2006

Knut Angenendt, Kilian Asangana,
Cigdem Cebe, Matthias Davidek,
Ralf Eckert, Daniela Hans,
Florian Hinz, Inga Laßwitz,
Lars Lüttmann, Malte Mathiszig,
Christoph Meyer, Nils Peters,
Martin Schmaeck, Jing Shui,
Christian Thevissen, Christian Wellinghorst

Inhaltsverzeichnis

1 Einleitung

Im Folgenden werden die Erfahrungen geschildert, welche im Rahmen der Projektgruppe „3D-Entwicklungsumgebung“, betreut von Prof. Dr. W. Kowalk gesammelt wurden.

Da die Projektgruppe neben dem Softwareprojekt das zweite „Programming in Large“ während des Informatikstudiums der CvO Universität Oldenburg darstellt, wurden vielerlei Eindrücke, sowohl fachlicher als auch gruppendynamischer Natur, vermittelt.

Auf der Softwareseite werden vor allem anderen die verwendeten Softwarekomponenten, namentlich OpenGL, Java und JOGL[1] diskutiert, sowie die Programmierwerkzeuge Eclipse und SVN. Ausserdem angewendet wurden die Textverarbeitungssysteme OpenOffice und LaTeX.

[1] Java Bindings for OpenGL

2 Software- Erfahrungsbericht

Da es für eine effiziente Entwicklung unumgänglich ist, auf vorhandene Software (wenigstens zur Unterstützung bei der Entwicklung) zurückzugreifen, wird im folgenden der hinterlassene Eindruck der einzelnen Werkzeug geschildert:

2.1 Eclipse

Bei Eclipse handelt es sich um eine integrierte Entwicklungsumgebung auf Open-Source-Basis, welche unter anderem auch die Entwicklung von Java-Programmen ermöglicht. Eclipse verfügt über diverse Features und kann durchaus mit jedem anderen, kommerziellen Entwicklungspaket verglichen werden.

2.1.1 Debug-Funktion

Der integrierte Debugger von Eclipse ist ein nützliches und wertvolles Hilfsmittel zum Auffinden von Programmfehlern. Die Debug-Funktionalität ermöglicht es, die Ausführung von Java-Programmen an jeder beliebigen, selbst durch sogenannte Breakpoints definierbaren Stelle zu unterbrechen. Daraufhin ist es dann möglich, das Programm schrittweise (d.h. Befehl für Befehl) auszuführen. Der Debugger folgt dabei auch Methodenaufrufen, wenn dies gewünscht wird. Besonders interessant ist die Möglichkeit, zur Laufzeit erzeugte Objekte nach jedem ausgeführten Schritt zu beobachten. Alle zum jeweiligen Zeitpunkt bekannten Objekte können angezeigt und in einer Baumstruktur betrachtet werden. So ist es möglich, genau nachzuvollziehen, wie ein Objekt zur Laufzeit des Programms von diesem verändert wird. Selbst Objekte, welche nicht im aktuellen Sichtbereich (also z.B. in der aktuellen Methode) liegen, können betrachtet werden, da es auch möglich ist, dem Debugger komplexe Ausdrücke zu übergeben, welche dieser dann auswertet. So befanden sich in unserem Fall beispielsweise viele Objekte in diversen Singleton-Klassen und wurden vom Debugger bei der Ausführung von Code an anderer Stelle nicht angezeigt, da sie nicht im lokalen Sichtbereich lagen. Um sie dennoch beobachtbar zu machen, genügte es, dem Debugger den entsprechenden Java-Code zu übergeben, um eine entsprechende Referenz auf das gewünschte Objekt zu erhalten (z.B. `SystemCache.getInstance().getScene()`). In sehr vielen Fällen ist der Debugger ein nahezu unerlässliches Hilfsmittel beim Auffinden von Programmfehlern, insbesondere von solchen, welche nur schwer intuitiv nachzuvollziehen sind. Die Fehlersuche wird durch konsequenten Einsatz der Debug-Funktionen von Eclipse wesentlich erleichtert und beschleunigt. Somit ist diese Funktionalität mehr als nur empfehlenswert und definitiv ein Highlight unter den Features von Eclipse.

2.1.2 Automatisierung der Code-Generierung

Eclipse bietet verschiedenen Funktionen an, um das Erstellen von Quellcode zu beschleunigen, indem z.B. Code-Fragmente, die stets gleich strukturiert sind, automtisch erzeugt werden oder eine Art „automatische Vervollständigung“ angeboten wird.

get- und set-Methodengenerierung
Eclipse bietet die Möglichkeit, automatisch get- und set-Methoden für die Variablen einer Klasse zu erzeugen. Dieses Feature kann sich als hilfreich erweisen, in der Regel allerdings nur um ein wenig Zeit zu sparen. Problematisch hierbei ist, dass in vielen Fällen die generierten Methoden ohnehin erweitert werden müssen und im Interesse einer sinnvollen Dokumentation das, ebenfalls automatisch generierte Javadoc-Fragment, in der Regel komplett ersetzt werden muss. Dennoch kann diese Funktion hilfreich sein - immerhin spart sie ein paar Zeilen zu schreibenden Code und sie kann (bis zu einem gewissen Grad) sicherstellen, daß keine get- und set-Methoden vergessen werden.

Generierung von try-catch-Blöcken
Es besteht die Möglichkeit, Eclipse automatisch try-catch-Blöcke erzeugen zu lassen, wo diese nötig sind. Da Eclipse die Notwendigkeit eines solchen Blocks automatisch erkennt, können so sehr schnell solche Code-Fragmente erstellt werden. Dies ist für den Programmierer eine sehr angenehme Funktion, da ihm mitgeteilt wird wo eventuell auftretende Exceptions abgefangen werden müssen und ihm auch die Arbeit des Abfangens abgenommen wird. Allerdings hat auch diese Funktion ihre Tücken, da die Automatisierung zwar viel Komfort bietet, aber auch dazu verleitet die abgefangenen Exceptions nicht weiter zu behandeln, sondern „versanden zu lassen“.

„automatische Vervollständigung“
Eclipse bietet eine Art „automatische Vervollständigung“ für geschriebenen Code an. Hierbei wird für jedes Objekt (bzw. für jede Variable, die ein Objekt repräsentiert) eine Auswahlliste der Objektvariablen und -methoden eingeblendet, sobald hinter dem Objekt- bzw. Variablennamen ein Punkt geschrieben wurde. Diese Liste wird darüberhinaus beim Weitertippen automatisch weiter eingegrenzt. Diese Funktion bietet verschiedene Vorteile: Zum einen kann sie das Erstellen des Quellcodes teilweise erheblich beschleunigen, da man nicht jeden Funktionsaufruf oder Variablenzugriff voll ausschreiben muss. Zum anderen können auf diese Art und Weise Tippfehler vermieden werden, da natürlich nur die tatsächlich vorhandenen Variablen- und Methodennamen eines Objekts in der Auswahlliste angeboten werden.

javadoc-Tool
Eclipse bietet eine komfortable Schnittstelle zu der „ javadoc.exe“, dem Programm des JDK, das aus den Javadoc-Kommentaren im Quellcode die HTML-Seiten des Javadocs generiert.
Hierzu selektiert man das Paket, zu dem man gerne die HTML-Seiten generieren möchte und wählt unter „Project → Generate Javadoc“.

In dem sich öffenenden Dialog muss nun einmal der Pfad zur „`javadoc.exe`“ eingegeben werden und schon können die HTML-Seiten erstellt werden. In dem Dialog können viele Einstellungen dazu gemacht werden, wie die generierten Seiten aussehen und welche Einträge sich in diesen Seiten befinden sollen. So kann von „private“ bis „public“ alles ausgewählt werden, damit die generierten Seiten die gewünschten Informationen beinhalten. Für unsere Dokumentation reichen die Standardeinstellungen, da wir kein eigenes Template für die HTML-Seiten erstellt haben sowie keine externen Seiten verlinken. Ebenso werden, falls Fehler auftreten, diese in der Konsole inklusive Links zu den fehlerhaften Kommentaren ausgegeben. Dadurch ist es sehr leicht, die fehlerhaften Kommentare zu korrigieren.
Es empfiehlt sich, diese Schnittstelle zu benutzen, da man sich so nicht mit den genauen Kommandozeilen-Befehlen auseinandersetzten muss.

2.1.3 Orientierungshilfen

Bei einem so komplexen System wie unserem, ist es sehr schwer die nötige Übersicht zu behalten. Deswegen bietet Eclipse einige vorzügliche Möglichkeiten an, schnell durch die Klassen zu navigieren:

Callhierarchie
Wenn man eine Methode ändern (oder löschen) will, kann es nützlich sein nachzusehen, welche andere Methode diese aufruft. Bei Eclipse klickt man dazu den Methodennamen an und kann dann mit „Strg + Shift + h“ die sogenannte *Callhirarchie* anzeigen lassen. Dies kann ggf. schon mal ein bißchen dauern.

Deklaration
Häufig kann es sehr hilfreich sein, wenn man schnell nachgucken kann, was eine bestimmte Methode macht, oder wie eine Variable initialisiert wurde. Am einfachsten geht das in Eclipse, wenn man „Strg“ drückt und dann die Methode oder die Variable anklickt. (Es wird dann auch ein „mouseover“-Effekt angezeigt.)

Tasks
Gerade beim Arbeiten mit mehreren Personen ist es sehr hilfreich unfertige Methoden oder gefundene Probleme im Code zu kennzeichnen. Dazu bietet sich zum einen „//TODO “ an. Dieses setzt sowohl einen Marker auf den Seitenrand und zum anderen kann man über „Window → Show View → Other → Basic → Tasks“ alle unfertigen Aufgaben eines Projektes anzeigen lassen. Alternativ dazu kann man auch „/FIXME “ nutzen.

Search
Die wohl beste Suchfunktionion in Eclipse ist unter „Search → File“ zu finden. Diese durchsucht alle .java Dateien nach einem String und listet das Ergebnis übersichtlich auf. Vorteil: Sie findet, was man sucht. Nachteil: sie findet es auch in auskommentierten Code.

2.2 Plattformunabhängigkeit: Linux und MacOS

Eine der größten Stärken der Programmiersprache *Java* ist die Plattformunabhängigkeit. Dadurch können in *Java* geschriebene Programme auf den verschiedensten Systemen ausgeführt werden. Bei unserem Projekt lag der Schwerpunkt auf *Windows-, Macintosh-* und *Linuxsystemen*. Es musste auf den jeweiligen Systemen die *JOGL-* und *JOAL-API* installiert werden, damit unser Projekt korrekt funktioniert. Jedes System hat andere, spezielle Binaries für *JOGL* und *JOAL*. Diese müssen je nach System in bestimmten Directories vorliegen.
Die Entwicklung unseres Projektes wurde zum Großteil unter *Windows* durchgeführt. Zwischenzeitliche Tests unter *Linux* und *Macintosh* sollten klären, ob unser Projekt auch auf anderen Systemen, nicht nur unter *Windows*, funktioniert. Die Tests unter *Mac OS* liefen fehlerfrei. Die meisten Probleme ergaben sich unter *Linux*.
Das erste Problem war die Ausgabe von Umlauten auf dem Bildschirm. Während die Umlaute unter *Windows* und *Mac OS* korrekt dargestellt wurden, konnte man unter *Linux* anstelle der Umlaute nur Sonderzeichen sehen. Damit die Umlaute unter allen Systemen korrekt dargestellt werden, mussten für diese die entsprechenden Unicodes verwendet werden.
Weitere Probleme traten bei der GUI auf. Manche spezielle Funktionen von *Java Swing* für die GUI funktionierten unter *Windows* und *Mac OS* einwandfrei, aber funktionierten nicht erwartungsgemäß unter *Linux*. Durch lange Fehlersuche, lesen der *Java Swing* Dokumentation und einfaches Ausprobieren mehrerer Möglichkeiten konnten auch diese Probleme behoben werden.
Zusammengefasst kann man sagen, dass man mit der Programmiersprache *Java* einen gewissen Grad an Plattformunabhängigkeit erreichen kann. Aber es gibt Kleinigkeiten, die bei verschiedenen Systemen zu Problemen führen können. Man kann also nicht mit *Java* unter einem einzigen System ein Programm entwickeln und dieses als plattformunabhängig bezeichnen, ohne es getestet zu haben.

2.3 SVN

Softwareentwicklung im Team ist heutzutage kaum noch ohne ein Versionierungssystem wie RCS, CVS oder Subversion (SVN) denkbar. Lange Zeit galt CVS als Standard, doch nach und nach scheint dieses System durch den inoffiziellen Nachfolger SVN abgelöst zu werden. Die Handhabung von SVN erinnert sehr stark an CVS, wobei einige CVS-Schwächen bei der Entwicklung von SVN vermieden wurden. So wird auf Versionsnummern einzelner Dateien verzichtet und stattdessen mit Revisionsnummern gearbeitet, sodass z.B. beim Auschecken einer älteren Versionen deren Konsistenz gewährleistet ist. Eine weitere Verbesserung ist die interne Verwendung von Versions-Deltas. Führt ein Entwickler ein Update auf seinen lokalen Daten durch, werden vom Server nicht die kompletten aktuellen Dateien übertragen, sondern lediglich die Änderungen, die seit dem letzen Update erfolgt sind. Dies vermindert die Netzlast.
Aus den oben genannten Gründen entschieden wir uns dazu, für die zentrale Verwaltung unseres Projekts ein SVN-Repository zu verwenden. Für jedes Team-Mitglied wurde ein Account erstellt, sodass zum einen gewährleistet war, dass alle jederzeit Zugriff auf die neuste Version des Projekts haben und andererseits nachvollziehbar blieb, wer wann welche Änderungen durchgeführt hat. Neben Java-Quellcode verwalteten wir im Repository auch die komplette Dokumentation und später die verwendeten Programmbibliotheken wie JOGL und JOAL in den passenden Versionen. Schließlich legten wir sogar die .classpath-Datei, die fester Bestandteil jedes Eclipse-Projekts ist, im Repository ab. Auf diese Weise wurde gewährleistet, dass nach dem Auschecken des Projekts die passenden jar-Dateien in Eclipse automatisch eingebunden wurden.
Ändern zwei oder mehrere Entwickler die selbe Datei im Repository, so ergibt sich unweigerlich beim Einchecken ein Konflikt, der behoben werden muss. In vielen Fällen geschieht dies jedoch automatisch, beispielsweise wenn an verschiedenen Bereichen einer Klasse gearbeitet wurde. In anderen Fällen, beispielsweise wenn die selbe Zeile von beiden Entwicklern verändert wurde, muss derjenige, die als letztes eincheckt, manuell für die Auflösung des Konflikts sorgen. Gerade wenn mehrere Tage oder sogar Wochen zwischen Aus- und Einchecken einer Datei liegen, kann es natürlich vermehrt zu derartigen Konflikten kommen, was im Laufe der Entwicklung unseres Projekts hin und wieder der Fall war. Sinnvoller wäre es sicherlich gewesen, offene Änderungen so früh wie möglich einzucheken um potentielle Konflikte frühzeitig aus dem Weg zu räumen.
Abgesehen von einigen anfänglichen Schwierigkeiten, die einige Mitglieder des Teams mit dem System hatten, den bereits angesprochenen Konflikten und einigen selten auftretenden, undurchsichtigen Feh-

lern beim Auschecken, erfüllte das SVN seinen Zweck. Eclipse bietet zwar keine SVN-Funktionalitäten an, so wie sie von Haus aus für CVS vorhanden sind, diese lassen sich jedoch über das PlugIn Subclipse relativ leicht einbinden. Darüber hinaus verwendeten einige von uns den Standalone SVN-Client TortoiseSVN, der ebenfalls nichts zu wünschen übrig lies.

2.4 LateX

Um einheitliche Dokumente innerhalb des Projektes zu erstellen einigte man sich auf LateX und damit verbunden dem TeXnicCenter.
Es handelt sich bei LateX eher um einen Interpreter, der etwa 300 fest eingebaute Befehle kennt und ausserdem einen Mechanismus um eigene Makros zu definieren zur Verfügung stellt.
Die Zeilenumbrüche (sowie Silbentrennung) werden durch ein Optimierungsverfahren im Absatz gleichzeitig bestimmt, sodass ein einfacherer Algorithmus danach den optimalen Seitenumbruch bestimmt. Diese Optimierungsverfahren arbeiten mit Bewertungspunkten, sodass mögliche Umbruchspunkte bewertet werden können.
Zeilen- und Seitenumbruch werden von vielen Parametern (Registern und elastischen Längen) gesteuert, die auch jederzeit im Dokument geändert werden können.
TeX wird durch eine Vielzahl von Makropaketen ergänzt, die eine effiziente und komfortable Nutzung von TeX erst ermöglichen.

2.4.1 MiKTeX

MiKTeX ist eine TeX-Distribution das Betreibssystem Windows. Mit Hilfe eines Installationsprogramms werden die benötigten TeX-Pakete aus dem Internet geladen und auf dem Rechner installiert. Eine Aktualisierung der Paktete ist zu jeder Zeit möglich und auch werden zusätzlich benötigte Pakete bei Bedarf nachgeladen und installiert.
Die meisten Programme der Distribution liegen als Kommandozeilenversionen vor, daher ist zum Schreiben von LaTeX-Quelltexte ein Texteditor notig. Es existieren mehrere Editoren, die für die Verwendung mit MikTeX angepasst sind. Dazu zählt das für MiKTeX entworfene und häufig parallel genutzte TeXnicCenter, das auch innerhalb unsere Projekts genutzt worden ist.

2.4.2 TeXnicCenter

TeXnicCenter ist ein freier Texteditor um unter Windwos LaTeX-Dokumente erstellen zu können.
TeXnicCenter hat integrierte Funktionen, die das Arbeiten, Strukturieren, Formatieren und vieles mehr erleichtern. Durch die Syntaxhervorhebung kann der Quelltext einfacher erstellt und bearbeitet werden. Der Editor enthält die LaTeX-Bausteine als Icons oder Tastenkombinationen, sodass oftmals das Eintippen langer Bausteine (für z.B. mathematische Befehle und Konstrukte) nicht mehr notwendig ist. Es wird eine umfangreiche Verwaltung von Projekten angeboten und Programme wie BibTeX können sehr einfach in die Umgebung eingebunden werden. Ebenfalls ist das Übersetzten der LaTeX-Quelltexte in die Formate DVI, PS und/oder PDF einfach per Tastenkombination oder Icon ausführbar.
Fehler innhalb des Quellcodes werden unten in der Konsole angezeigt und per Mausklick kann zu dem Fehler im Quellcode gesprungen werden, sodass ein langes Suchen überflüssig wird.

2.4.3 TeXnicCenter - innerhalb des Teams

Zu Beginn der Projektgruppe kannte sich nur ein kleiner Teil mit LaTeX und dem TeXnicCenter aus, sodass sich innerhalb der Gruppe erst geeinigt werden musste, ob die Dokumente in diesem Format erstellt werden sollen. LaTeX erscheint im Gegensatz zu OpenOffice erst als wesentlich komplizierter bei der Erstellung von Texten. Doch nachdem man sich einen kleinen Befehlssatz, mit dem LaTeX arbeitet, angeeignet hat, lassen sich einheitliche Dokumente erstellen, was auch ein Vorteil von TeX ist.
Nachteilig ist wohlmöglich, dass es eine Weile dauert, bis man sich an die Umgebung von TeX gewöhnt hat und auch das Einbinden von Grafiken ist zuerst sehr gewöhnungsbedürftig.

Jedoch hat das Team dem Sprung gewagt und hat sich gemeinsam an TeX herangetastet. Die Mitgleider die sich auskannten haben den anderen stets ausgeholfen, sodass es später allen nicht mehr so schwer fiel mit dem TeXnicCenter zu arbeiten.

2.5 Star Office

Neben der Verwendung von Latex wurde auch auf andere Anwendungen zur Erstellung von Textdokumenten zurückgegriffen. Hierbei handelt es sich Star Office bzw. um Open Office. Da diese beiden Produkte sich sehr ähnlich sind, da Star Office auf Open Office basiert, gehen wir an dieser Stelle nur auf die Sun Variante Star Office ein. Diese Office Anwendungen wurden vor allem für die Erstellung der Seminarausarbeitungen und der Seminar- sowie Projektpräsentation verwendet. Die aktuelle Version von Star Office ist bei der Textverarbeitung sowie bei der Erstellung von Präsentationen durchaus vergleichbar mit MS Office. Dies betrifft sowohl die Bedienung als auch den Funktionsumfang. Die Vorteile von Star Office im Vergleich zu MS Office liegen zum einen in den Kosten, da das MS Office Packet um ein vielfaches teuerer ist. Auch was die Kompatibilität betrifft ist Star Office die bessere Wahl. Es können alle gängigen Officeformate gelesen und geschrieben werden, also sowohl Microsoft Dokumente als auch die Star(Open) Office Varianten. Das betrifft auch die entsprechenden Dokumente älterer Versionen (bis MS Office 6.0 bzw. Star Office 3.0). Ähnlich wie bei der Microsoft Variante können auch andere Formate geschrieben werden, wie XML oder HTML. Ein weitere Vorteil ist die Exportfunktion für PDF. Es ist also keine zusätzliche Software nötig, um ein Dokument als PDF zu speichern. Das Star Office lässt sicht nicht nur unter Windows sondern auch unter Linux problemslos Installieren.

3 Teamwork- Erfahrung

Die Projektgruppe „Yage 3D“ wurde ursprünglich aus 2 Projektgruppen zusammengeschlossen, diese sind die Projektgruppe „Welteditor“ und die Projektgruppe „Animated Mesh“. Insgesamt bestand das Team aus 16 Teilnehmern.

Durch die Zusammenarbeit in der Gruppe wurde natürlich in erster Linie die Teamfähigkeit geschult. Das heißt, man musste flexibel sein, was Arbeitsweise und zeitliche Vereinbarungen betraf. Ab und zu musste man seine eigenen Ziele zurückstecken können, um sich mit anderen zu arrangieren. Auch dadurch, dass unsere PG- Treffen über einen so langen Zeitraum stattgefunden haben, musste man sich mit den Leuten abstimmen und konnte bei Problemen nicht einfach das Handtuch werfen. Gerade später für den Beruf sind Eigenschaften wie Teamfähigkeit von hoher Bedeutung. Was an der Arbeit in der Gruppe positiv aufgefallen ist, war, dass eine Gemeinschaftsgefühl entstand, jeder seinen eigenen Teil zur Arbeit beisteuerte und die Verantwortung annähernd gut verteilt war, so dass man sich selber nicht zu sehr unter Druck gesetzt fühlte. Negativ war die Tatsache, dass es keinen Leiter des Teams gab und so zuweilen bei der Arbeit das eigentliche Ziel leicht aus den Augen verloren werden konnte. Durch mangelnde Absprachen kam es zum Beispiel vor, dass Arbeit doppelt ausgeführt wurde, dies hätte durch eine bessere Kommunikation verhindert werden können. Daraus haben wir für die Zukunft gelernt, dass bei der Teamarbeit Organisation und Kommunikation untrennbar miteinander verbunden sind. Denn ohne Absprachen ist ein Team nicht arbeitsfähig. Andere Probleme, die auftraten, waren, dass unterschiedliche Lernstände vorhanden waren. So musste die Arbeit so verteilt werden, dass die Kommilitonen mit Vorkenntnissen, schwierigere Aufgaben zu bearbeiten hatten, als diejenigen mit weniger Kenntnissen. Außerdem sollte angemerkt werden, dass zu Beginn der Projektarbeit die Motivation sehr groß war und doch mit der Zeit leider merklich nachließ. Dies kann zu einem mit der Dauer des Projektes zusammenhängen, zum Anderen aber auch damit, dass man bei manchen Aufgaben einfach stecken blieb und nicht mehr weiterwusste. So saß man beispielsweise 2 Stunden vor einem Problem, dass jemand anderes innerhalb von 2 Minuten lösen konnte, dies lässt sich mit den schon beschriebenen unterschiedlichen Lernständen begründen oder aber auch dadurch das man bei solch einer Arbeit oft die Übersicht verlieren konnte. Positiv war, dass eine Art Teamleiter bzw. Ansprechpartner in der jeweiligen Gruppe gewählt wurde. Dieser hatte zwar nicht die übliche leitende Funktion (dies war ja gleichermaßen auf alle Teammitglieder aufgeteilt), aber stand als Ansprechpartner bei Problemen und auch bei der Kommunikation mit dem leitenden Professor bereit.

Abschließend lässt sich sagen, dass die langfristige Arbeit in einem solchen Team die sozialen Kompetenzen verbessert, sich schwierige Arbeit besser zu mehreren erledigen lässt, aber dass die Koordination teilweise hätte verbessert werden müssen, um bestmögliche Ergebnisse zu leisten.