Vortrag

Verteilte Versionsverwaltung in Microsoft-Access-Projekten

Access, OASIS und Mercurial im Team

Christoph Jüngling

14. AEK in Nürnberg und Hannover, Oktober 2011

Kontakt zum Autor:

Christoph Jüngling
Schwarzenbergstraße 66
D-34130 Kassel

christoph@juengling-edv.de
http://www.juengling-edv.de
Mobil: +49 177 4643428

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Allgemeines

Abbildung 1: Titel

1.1 Ziele

Dieser Vortrag hat zum Ziel, das Szenario *Access-Programmierer mit verteilter Quellcodeverwaltung* zu beschreiben.

Es geht also weniger darum, wie man Quellcodeverwaltung generell macht oder was für Methoden und Vorgehensweisen es dafür gibt. Das wäre ein eigener Vortrag, den ich gern mal irgendwann halten kann. Wir wollen sehen, wie ein mit MS Access™ arbeitender Entwickler auch dann erfolgreich arbeiten kann, wenn er dies zeitweise unabhängig von einem zentralen Server tun muss. Und um zu zeigen, dass und wie seine Änderungen und die Anmerkungen dazu ohne Problem das Team wieder erreichen.

Mercurial und andere Quellcodeverwaltungssysteme sind naturgemäß nicht auf Windows beschränkt und auch nicht in der und ausschließlich für die Windows-Welt erfunden worden. Somit können in allen Projekten, die z.B. mit Mercurial arbeiten wollen, auch andere Betriebssysteme verwendet werden. Die erzeugten Repositories sind durch die Fähigkeiten von Mercurial pro-

blemlos auch zwischen verschiedenen Betriebssystemen austauschbar. Leider gilt dies für Access nicht, sollte aber als Hinweis für z.B. VB-Dotnet-Projekte im Gedächtnis behalten werden (siehe „Mono“).

Wir betrachten für diesen Vortrag nur den ganz konkreten Fall, dass wir unter Windows™ mit MS Access™, OASIS, TortoiseHg und Mercurial arbeiten. Andere Mütter haben gewiss auch schöne Töchter, aber wir verhalten uns hier einfach mal geradezu vorbildlich monogam.

Nebenbei geht es auch darum, mit ein paar verbreiteten Vorurteilen aufzuräumen.

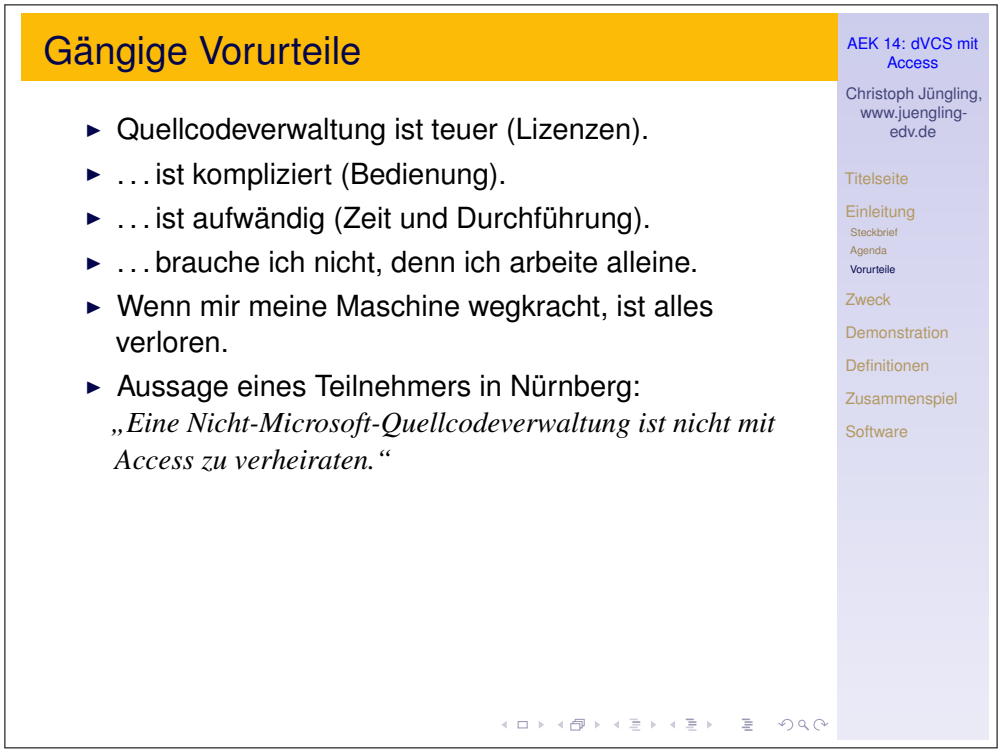

Abbildung 2: Verbreitete Vorurteile

1.2 Motivation

Neben dem bereits aus dem Namen offensichtlich zu erkennenden Zweck von Quellcodeverwaltung, nämlich eben den Quellcode zu verwalten, verfolgt oder besser erreicht man damit noch weitere Ziele. Vordringlich sind hier die bessere (oder überhaupt erst ermöglichte) Zusammenarbeit in einem Team und die Reduzierung des persönlich empfundenen Stresses zu nennen.

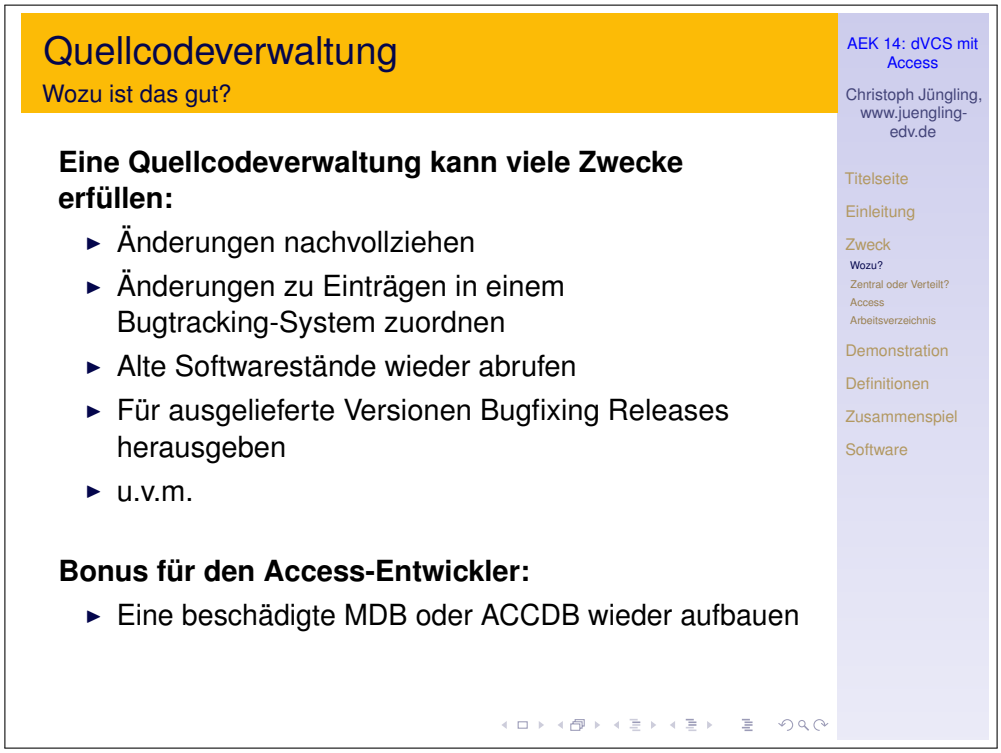

Abbildung 3: Gründe für Quellcodeverwaltung

Die Zusammenarbeit in einem Team kann bei *verteilter* Quellcodeverwaltung sogar noch um eben dieses Attribut „verteilt“ erweitert werden: Auch ein *verteiltes* Team kann hiermit wesentlich leichter zusammenarbeiten als beispielsweise mit einer zentralisierten Quellcodeverwaltung, da nicht immer eine permanente Internetverbindung vorausgesetzt werden kann.

Dies ist beim Programmieren unterwegs (Zug, Flugzeug, Kreuzfahrtschiff, Rettungsboot, Südseeinsel) besonders augenfällig. Während die Stromversorgung mittels Solarzellen in aller Regel kaum noch ein Problem darstellt, ist ein Internetzugang vermutlich noch nicht auf jeder einsamen Insel gewährleistet. Kommt „Robinson“ dann nach längerer Abwesenheit wieder in Kontakt mit der Zivilisation, so kann er seine programmiertechnischen Ergüsse wieder mit dem Team teilen. Hier bieten Programme wie das vorgestellte Mercurial deutliche Vorteile gegenüber einem System, das den Quellcode nur dann verwalten kann, wenn man online ist.

Der empfundene Stress wird wiederum dadurch reduziert, dass jede Änderung im version control system, dt. Quellcodeverwaltungssystem oder Versionskontrollsystem (VCS) untilgbar[1] ihre Spuren hinterlässt und somit jede kleine Aktion nachvollziehbar ist — und auch wieder rückgängig gemacht werden kann. Der Entwickler erhält quasi nebenher ein inkrementelles

[1] Zwar gibt es z.B. mit Git auch ein System, das es erlaubt, die Geschichte umzuschreiben, aber das wird selbst in dessen Doku nur für Ausnahmefälle empfohlen.

Backup seines Projektes, das kommentiert („log“) und an den nötigen Stellen etikettiert („tags“) ist.

1.3 Unterschiede

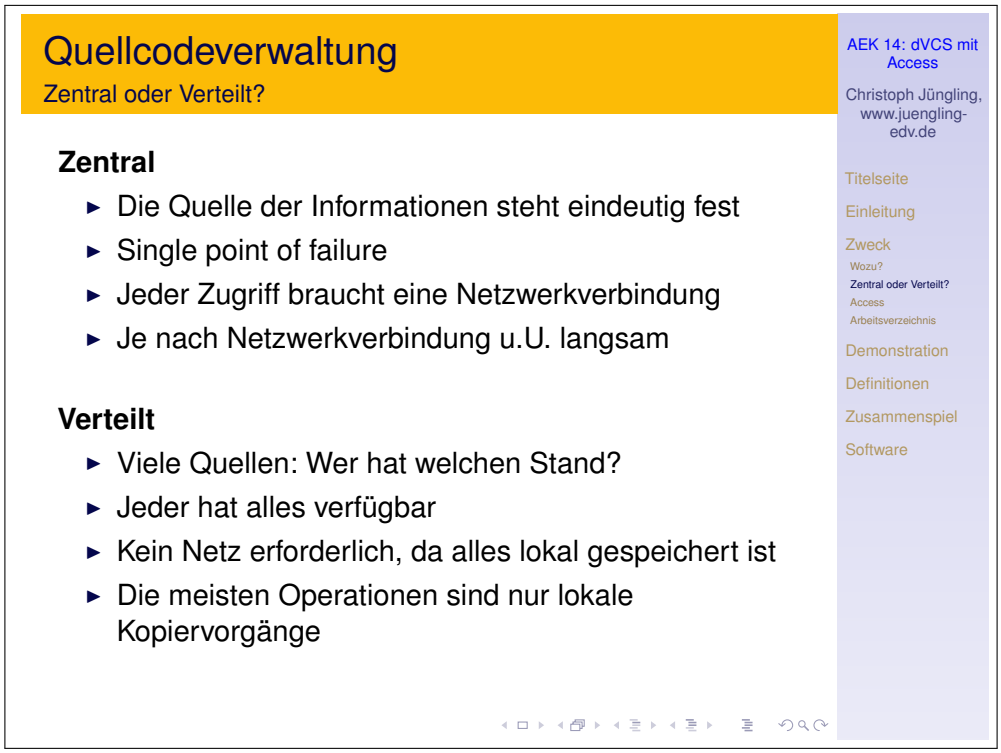

Abbildung 4: Zentrale oder verteilte Quelcodeverwaltung?

Auch in einem centralized version control system, dt. zentralisiertes Quellcodeverwaltungssystem (cVCS) können viele Leute an dem gleichen Projekt arbeiten. Es muss jedoch für jede Aktion eine Verbindung zum Server existieren (Abb. 5), was wie oben dargestellt nicht unter allen Umständen gewährleistet werden kann. Aber auch für den Fall, dass das normalerweise vorhandene Internet oder Firmennetzwerk gestört ist oder der VNC-Einwahlserver Probleme hat, kann weder neuer Code „eingecheckt“ (im Mercurial-Jargon ist „checkin“ = „commit“) noch lokale Änderungen rückgängig („revert“) gemacht werden. Nicht einmal eine Recherche im Log ist dann möglich!

Bei einem distributed version control system, dt. verteiltes Quellcodeverwaltungssystem (dVCS) geht es wesentlich entspannter zu (Abb. 6). Es kann auch ohne Serververbindung gearbeitet werden, da jeder Entwickler das komplette Repository in Kopie mit sich herumträgt. Eine Serververbindung ist also nicht notwendig, aber dennoch möglich.

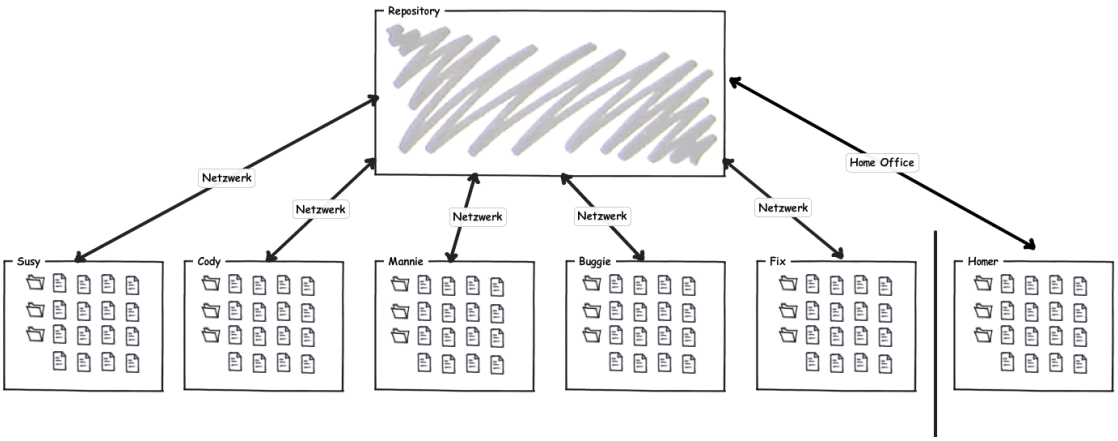

Abbildung 5: Zentralisiertes Quellcodeverwaltungssystem

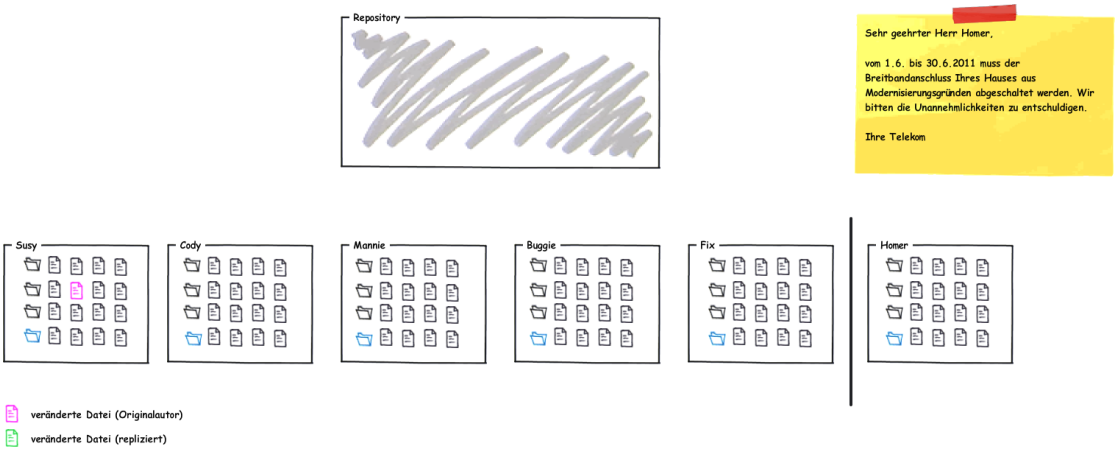

Abbildung 6: Verteiltes Quellcodeverwaltungssystem

1.4 Besonderheiten bei MS Access

Normalerweise sind Programmiersprachen so aufgebaut, dass jede Klasse in einer einzelnen Datei gespeichert wird. Das hat den großen Vorteil der Übersichtlichkeit und bietet darüber hinaus auch eine gewisse Unabhängigkeit von einer bestimmten Entwicklungsumgebung, engl. integrated development environment (IDE). Denn der Programmcode liegt in den vielen kleinen Dateien zumeist im ASCII/ANSI-Format vor, das selbst der simpelste Editor versteht.

Bei MS Access™ ist das leider anders: Die MDB oder ACCDB ist ein Containerformat, das die einzelnen Formulare, Module, Klassen usw. enthält. Das Dateiformat ist von außen betrachtet alles andere als lesbar, und von einer Veränderung mittels Notepad oder gar MS Word™ sollte man unbedingt Abstand nehmen. Wir brauchen also einen Zwischenschritt (Abb. 7), der die

einzelnen Access-Objekte in viele Dateien exportiert, die dann dem VCS angeboten werden. Umgekehrt muss dieser Zwischenschritt ebenfalls funktionieren, also nach einer Änderung durch das VCS aus den Dateien wieder Access-Objekte machen. Für diesen Zwischenschritt verwenden wir in diesem Vortrag OASIS.

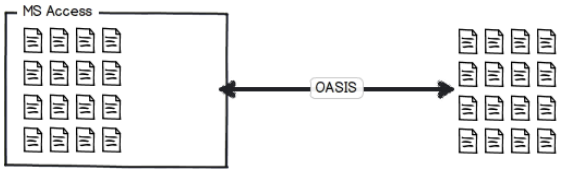

Abbildung 7: Access und die kleinen Dateien

2 Verwendete Software

2.1 Downloadressourcen

Zum Zeitpunkt dieses Vortrages (also im Oktober 2011) befanden sich die Quellen für die verwendete Software an den folgenden Stellen:

OASIS: `http://www.dev2dev.de/oasis.html`

TortoiseHg: `http://tortoisehg.bitbucket.org/`

Der Download von TortoiseHg schließt auch eine passende Mercurial-Version mit ein, so dass diese nicht separat heruntergeladen und installiert werden muss.

Die Installation der beiden genannten Programme gestaltet sich sehr einfach: Führe (mit Adminrechten) einfach das Installationsprogramm aus.

2.2 Zusammenspiel der Komponenten

Wie bereits in Kap. 1.4 Besonderheiten bei MS Access (S. 9) erwähnt, müssen die Access-Objekte in einzelne Dateien exportiert werden, die dann dem VCS zur Verwaltung überstellt werden. Das dVCS wiederum arbeitet mit einem *lokalen* Repository und beliebig vielen *entfernten*. Abb. 8 soll dies am Beispiel eines einzelnen Server-Repositories grafisch veranschaulichen.

Im Unterschied zu der von Microsoft favorisierten Vorgehensweise, ein Microsoft Source Code Control Integration (MSSCCI) genanntes Addin zu verwenden, geht OASIS einen etwas anderen

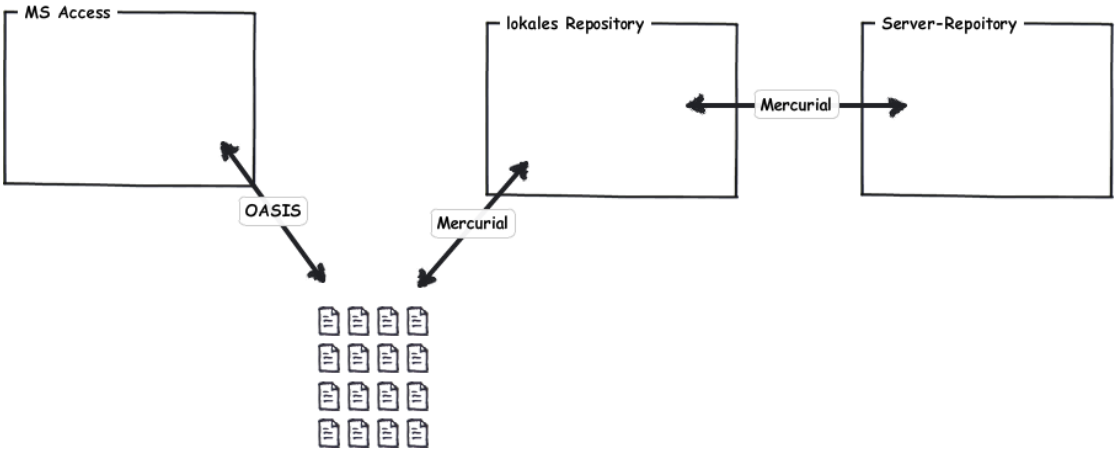

Abbildung 8: Zusammenarbeit von Access, OASIS und Mercurial

Weg. Da ein MSSCCI-Provider nur einen sehr begrenzten Funktionsumfang unterstützt, diesen aber vollständig erwartet, wären wir mit einem von außerhalb der Microsoft-Welt stammenden VCS schlecht beraten. Da Microsoft selbst derzeit kein *verteiltes* Quellcodeverwaltungssystem anbietet, gehen wir in diesem Vortrag einen gänzlich anderen Weg: Wir verwenden OASIS anstelle der MSSCCI und Mercurial mit der Oberfläche TortoiseHg anstelle von z.B. Team Foundation Server (TFS) oder dem älteren Visual SourceSafe (VSS).

Interessant ist in diesem Zusammenhang, dass Microsoft gerade im Oktober 2011 bekanntgegeben hat, dass ihre eigene Open-Source-Codehosting-Site `http://www.codeplex.com` nun ebenfalls Mercurial unterstützt.

2.3 Position des lokalen Repositories

Die Position des lokalen Repositories ist noch erwähnenswert, denn in dieser Hinsicht unterscheidet sich ein dVCS von einem cVCS. Wie Abb. 9 und Abb. 19 zeigen, befindet sich das lokale Repository einfach im Arbeitsverzeichnis, genauer gesagt in dem dortigen Unterverzeichnis `.hg`.

2.4 Konfiguration OASIS

Startet man neu mit der Quellcodeverwaltung in einem Projekt, so ist es notwendig, die Einstellungen von OASIS durchzugehen und an das jeweilige Projekt anzupassen. Anschließend wird die `.oasis`-Datei dem VCS unterstellt.

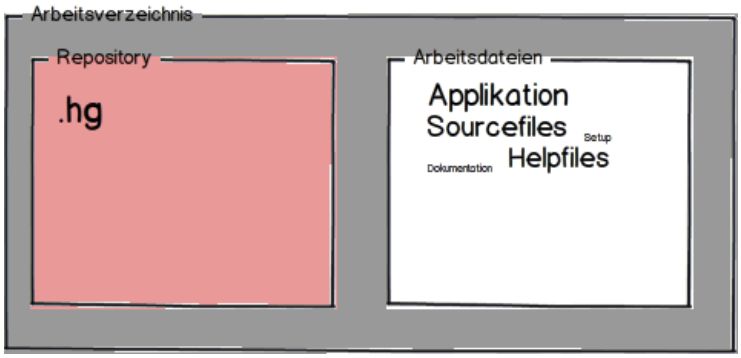

Abbildung 9: Position des lokalen Repositories

Wenn wir in ein bestehendes Projekt einsteigen oder einfach nur einen Clone davon erzeugen (s. Kap. 3.2), ist die `.oasis`-Datei bereits im Repository enthalten. Ihre Einstellungen werden automatisch wirksam, wenn die neue Datenbankdatei den gleichen Basisnamen wie die `.oasis`-Datei bekommt.

2.4.1 Lizenz

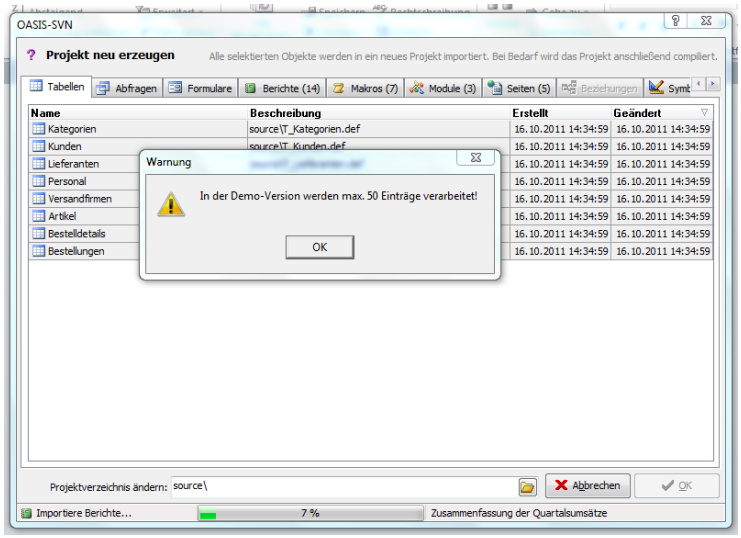

Abbildung 10: OASIS-Demoversion

OASIS ist Shareware, was im konkreten Fall bedeutet, dass das Programm dem Anwender in der unregistrierten Version nur einen begrenzten Leistungsumfang zugesteht. Solange also noch keine Lizenz erworben wurde, muss mit Meldungen wie in Abb. 10 gerechnet werden. Für die ersten Tests mag die Demoversion genügen, jedoch wird bereits die Nordwind-Datenbank diese Grenze überschreiten.

Weitere Informationen zu der Lizenzierung (insbesondere auch zu Volumenlizenzen) gibt es auf http://www.dev2dev.de/showstory-3.html. Eines kann ich vorab schon sagen: Es ist nicht teuer!

2.4.2 Objekttypen

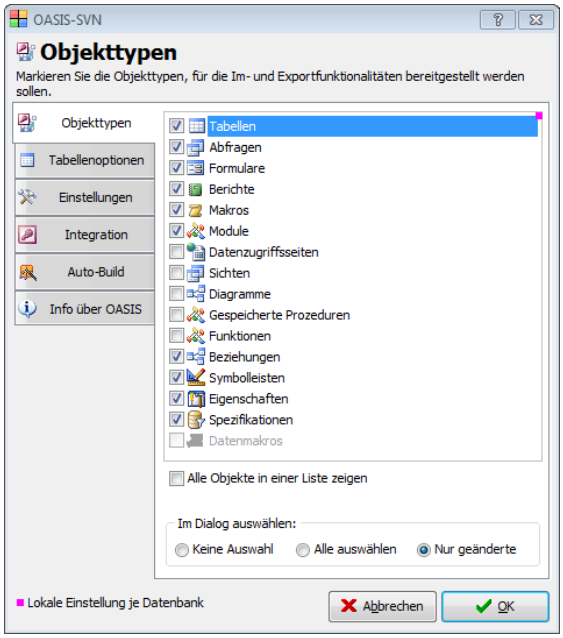

Abbildung 11: OASIS-Einstellungen: Objekttypen

Der Screenshot in Abb. 11 zeigt wohl die gängigsten Objekte, die man aktivieren möchte. Selbstverständlich ist das in jedem Projekt ein wenig anders. Es spricht auch nichts dagegen, einfach alles anzukreuzen, da OASIS in den Fällen, wo z.B. kein Bericht vorhanden ist, auch nichts exportiert.

Die Option „nur geänderte“ bei der Frage nach den auszuwählenden Objekten ist nur eine Empfehlung, siehe dazu Kap. 2.4.4 Einstellungen (S. 14).

2.4.3 Tabellenoptionen

Ein kritischer Punkt auf der Seite „Tabellenoptionen“ ist sicher die Frage, ob Tabellendaten exportiert werden sollten. Ich empfehle hier, nach der Art der Tabellen zu unterscheiden: Sind es reine Datentabellen, die vom Benutzer bearbeitet werden, oder sind es eher statische Tabellen, die

Informationen für Dropdownlisten enthalten oder vielleicht sogar Konfigurationsinformationen für das Programm selbst?

Empfehlung: Für Tabellen, die vom Benutzer gepflegt werden, sollten keine Daten exportiert werden. Tabellen, die für den Betrieb des Programmes notwendig sind, sollten Daten enthalten.

Um im Einzelfall festzulegen, welche Tabellen mit und welche ohne Daten exportiert werden sollen, klicke auf „Erweitert“ (Abb. 12). Zur Orientierung wird hier auch angezeigt, wieviele Datensätze gerade in der jeweiligen Tabelle enthalten sind.

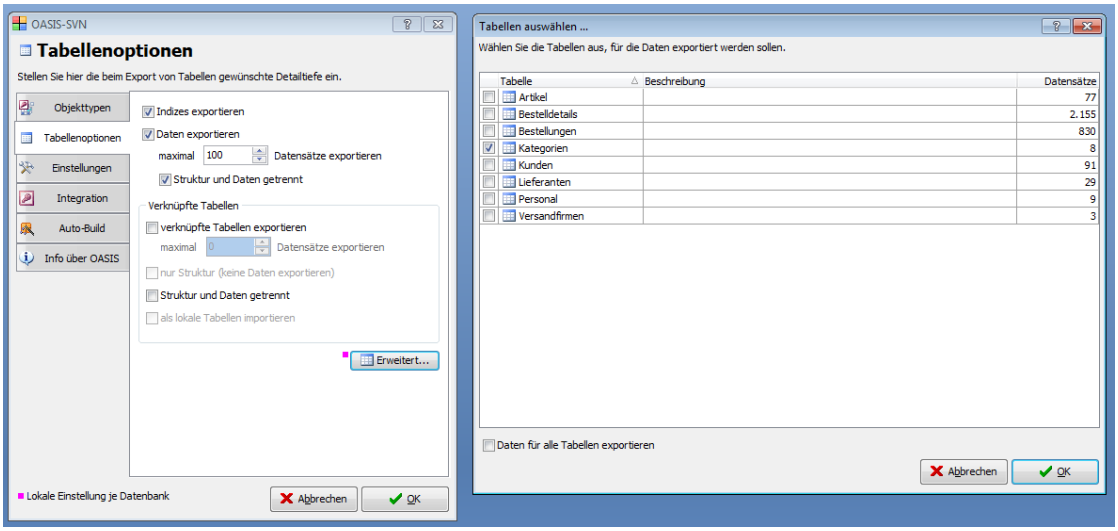

Abbildung 12: OASIS-Einstellungen: Tabellenoptionen

2.4.4 Einstellungen

Abb. 13 zeigt den gesamten Einstellungsdialog von OASIS. Die dortigen Einstellungen können getrost als Empfehlung betrachtet werden.

Als erstes sei hier ein Hinweis gegeben, der direkt mit der Quellcodeverwaltung zu tun hat: *Niemand weiß, wo bei einem anderen Entwickler das Arbeitsverzeichnis liegen wird!* Wird OASIS einigermaßen geschickt konfiguriert, dann ist dies auch völlig egal. Dies betrifft die beiden ersten Einstellungen wie in Abb. 14 dargestellt. Hier ist es wichtig, dass zunächst ohne das Häkchen bei „immer <CurrentProject.Path> verwenden“ in das Feld „Im- und Exportverzeichnis für diese Datenbank“ ein Unterverzeichnisname eingegeben wird. Klicke *danach* auf „immer <CurrentProject.Path> verwenden“ und dieses Unterverzeichnis wird dem Projektpfad hinzugefügt (Abb. 15).

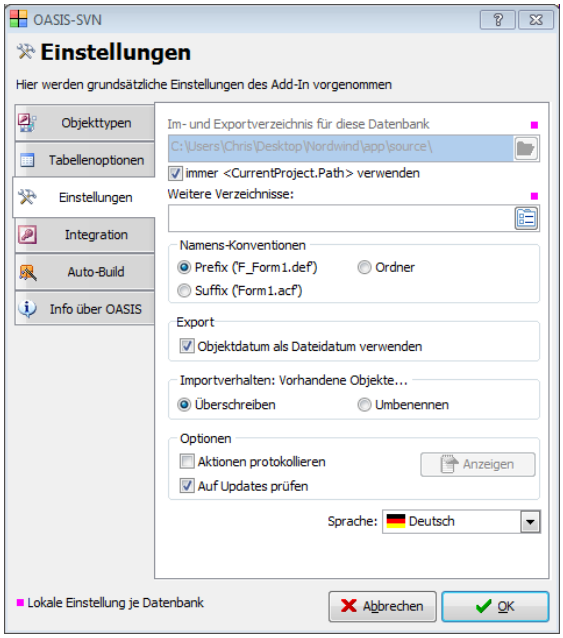

Abbildung 13: OASIS-Einstellungen: Einstellungen

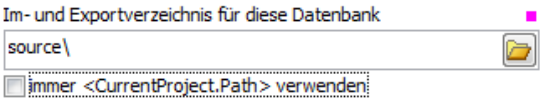

Abbildung 14: Source-Ordner vor Eingabe

Die **Namenskonventionen** sind ebenfalls ein wichtiger Punkt, der in einem Projekt festgelegt werden muss. Sie bestimmen direkt, wie die Dateinamen aus den Access-Objekt-Namen aufgebaut werden sollen.

Die Option „Objektdatum als Dateidatum“ unterstützen die in Kap. 2.4.2 Objekttypen (S. 13) erwähnte Option „nur geänderte“.

2.4.5 Integration

Zu der Einstellungsseite „Integration“ ist wenig zu sagen. OASIS hat eine gute Integration in das Produkt TortoiseSVN, welche natürlich nur aktiv werden kann, wenn diese Software installiert ist. In Abb. 16 ist das nicht der Fall, weshalb hier nur eine entsprechende Meldung erscheint. Da wir mit Mercurial und nicht mit Subversion arbeiten, ist das für uns keine Einschränkung. Vom Programmautor weiß ich, dass er an einer Mercurial-Integration arbeitet, so dass sich früher oder später hier noch etwas tun wird.

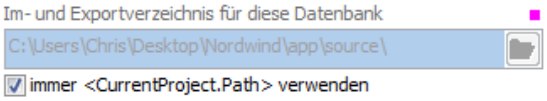

Abbildung 15: Source-Ordner nach Eingabe

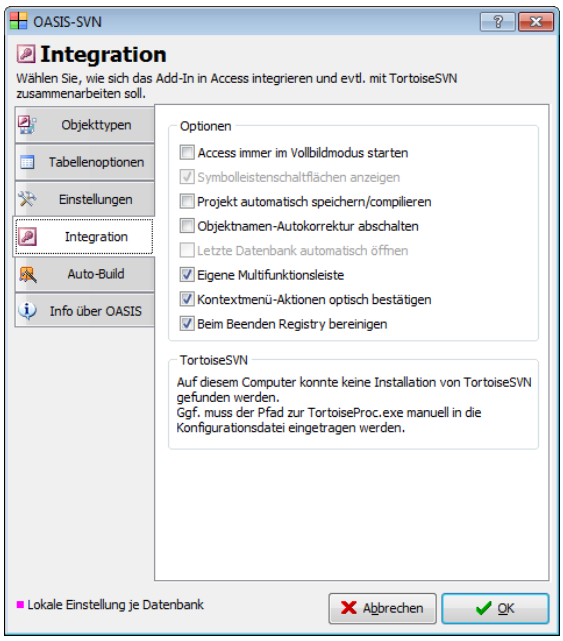

Abbildung 16: OASIS-Einstellungen: Integration

2.5 Konfiguration TortoiseHg und Mercurial

2.5.1 Lizenz

Mercurial und TortoiseHg stehen unter „GPL Version 2 oder später“, siehe dazu `http://mercurial.selenic.com/wiki/License` und den About-Screen von TortoiseHg.

2.5.2 Allgemeines

Die Konfiguration dieser beiden Programme erfolgt sehr komfortabel über TortoiseHg, so dass wir uns in aller Regel nicht direkt mit irgendwelchen INI-Dateien beschäftigen müssen.

Als erstes müssen wir die verschiedenen Ebenen unterscheiden, die bei TortoiseHg bzw. Mercurial

konfiguriert werden können:

1. Installation
2. User
3. Repository

Dabei gelten die Einstellungen in dieser Aufzählung von oben nach unten in der Form, dass die obere jeweils die Grundlagen für die untere(n) darstellt. Das bedeutet, dass eine Repository-Einstellung für dieses Repository Vorrang vor den User- und Installationseinstellungen hat.

Auf **Installationsebene** wird oft nur wenig gemacht, da diese meist mit den Standardeinstellungen gut auskommt.

Auf **Userebene** werden wir in der Regel unseren Namen mit eMail-Adresse eintragen wollen wie in Abb. 17 gezeigt, da dieser natürlich nicht für alle User dieses Rechners, aber für alle *meine* Repositories gilt.

Empfehlung: Die deutsche Übersetzung von TortoiseHg ist in einigen Teilen recht originell, so dass man etwas raten muss, bevor man hinter die Bedeutung kommt. Hier ist es besser, die englische Version beizubehalten. Die Umstellung erfolgt aus der Workbench über das Menü `Datei→Einstellungen→TortoiseHg→Sprache`. Danach ist ein Neustart der Workbench fällig.

2.5.3 Entferntes Repository bedienen

In den Fällen, wo ein entferntes Repository automatisch mit allen meinen Änderungen versorgt werden soll, wird das dVCS quasi wie ein cVCS benutzt. Dennoch wird das lokale Repository weiterhin (und vorrangig) gepflegt! Die dafür hilfreiche Einstellung „Push after commit“ (in der deutschen Übersetzung „nach Übernahme verteilen“) kann auf Repository-Ebene vorgenommen werden, wenn das aktuelle Repository ein Clone eines anderen ist. Das ist zum Beispiel sinnvoll, wenn das entfernte Repository auf einem oft erreichbaren Server liegt. Dabei wird der Kurzname des entfernten Repositories aus der Dropdownliste ausgewählt wie in Abb. 18 gezeigt.

Dieses Vorgehen empfiehlt sich auch, wenn die Programmierung in einer virtuellen Maschine erfolgt und sichergestellt werden soll, dass auch bei einem Absturz der VM nichts verloren geht.

Falls das entfernte Repository mal nicht erreichbar ist, wird der lokale Commit trotzdem zuerst gemacht und bleibt auch erhalten, selbst wenn der „Push“ fehlschlägt.

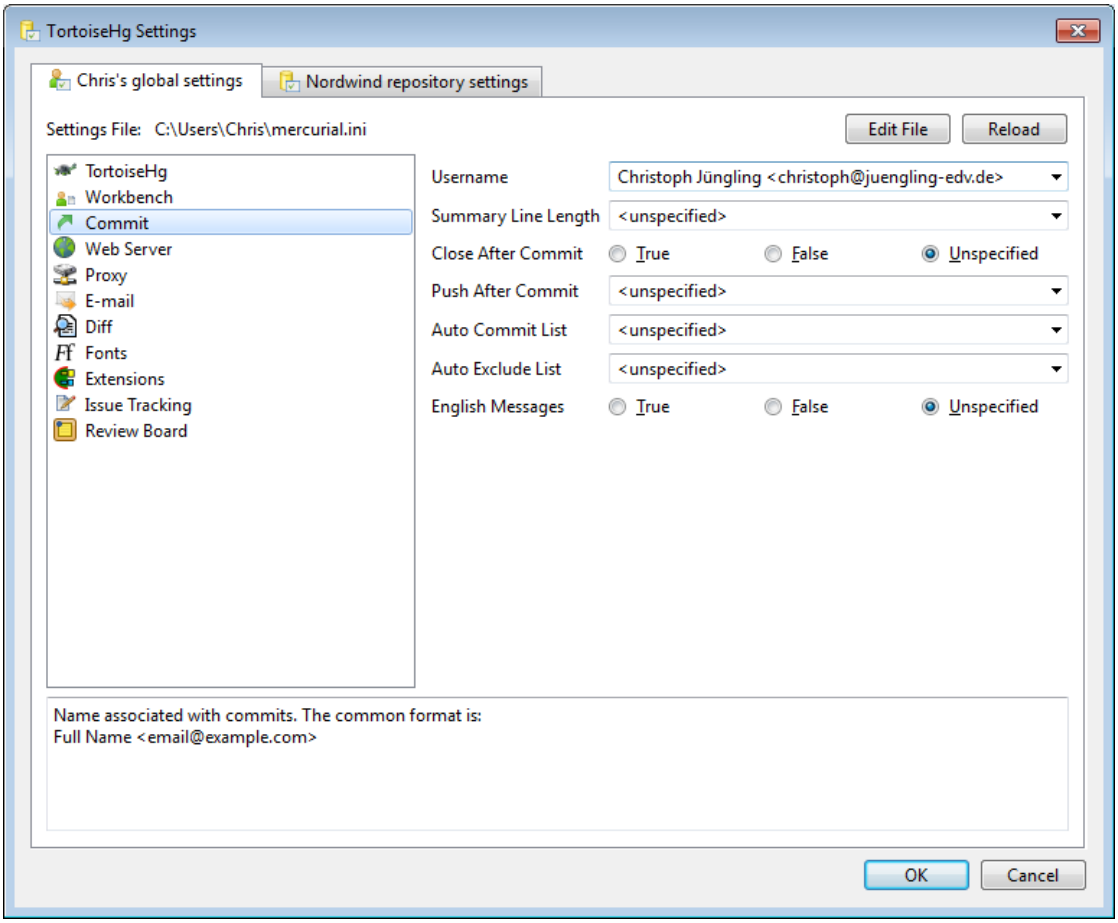

Abbildung 17: TortoiseHg-Einstellungen: User

2.6 Empfohlene Verzeichnisstruktur

Generell kann ein mit Mercurial verwaltetes Projekt beliebig viele Dateien und Ordner enthalten. Mercurial wird immer die gesamte Arbeitsverzeichnishierarchie betrachten, jedenfalls soweit die Dateien seiner Verwaltung unterstellt wurden. Die in Abb. 19 gezeigte Verzeichnisstruktur ist daher keineswegs die einzig mögliche, sondern nur eine Empfehlung. Jeder mag seine eigenen Ideen verwirklichen.

3 Szenarien

Im Folgenden sollen einige wichtige Szenarien und die dabei erforderlichen Vorgehensweisen beschrieben werden. Selbstverständlich gibt es nahezu beliebig viele solcher Szenarien, so dass

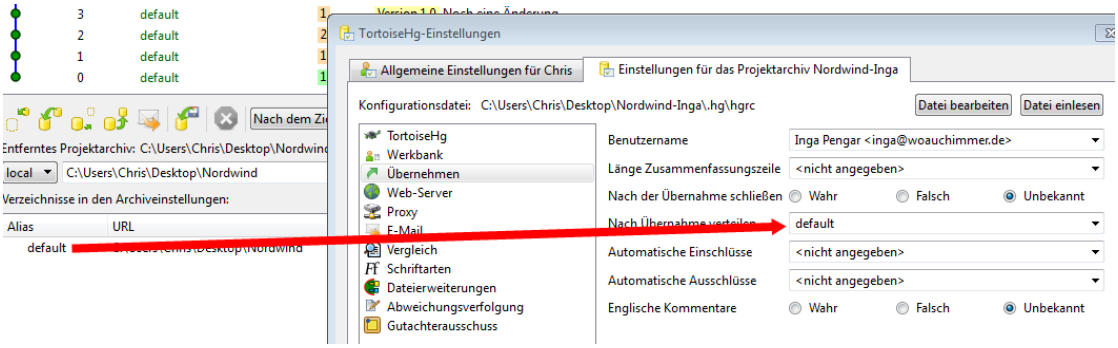

Abbildung 18: Push after commit

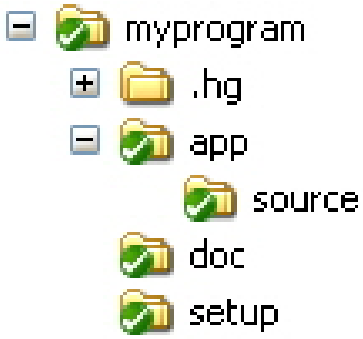

Abbildung 19: Verzeichnisse

eine solche Aufzählung niemals vollständig sein könnte.

3.1 Projekt unter Quellcodeverwaltung stellen

Um in einem bestehenden Access-Projekt mit der Quellcodeverwaltung zu beginnen, gehe wie folgt vor:

1. Richte um die Access-Datenbank herum die empfohlene Verzeichnisstruktur gemäß Kap. 2.6 Empfohlene Verzeichnisstruktur (S. 18) ein.

2. Erzeuge mit TortoiseHg im Hauptverzeichnis dieser Struktur ein Repository (`TortoiseHg→ Create Repository here`).

3. Nimm die Repository-Einstellungen gemäß Kap. 2.5 Konfiguration TortoiseHg und Mercurial (S. 16) vor.

4. Öffne die Datenbank und konfiguriere OASIS gemäß Kap. 2.4 Konfiguration OASIS (S. 11).

5. Exportieren alle benötigten Access-Objekte mit OASIS.

6. Öffne die TortoiseHg-Workbench, füge alle gewünschten Dateien hinzu (Abb. 20), lasse nicht gewünschte ignorieren, schreibe eine aussagekräftige Commit-Message und führe den Commit aus.

7. Ggf. folgt noch ein Push zu einem entfernten Repository.

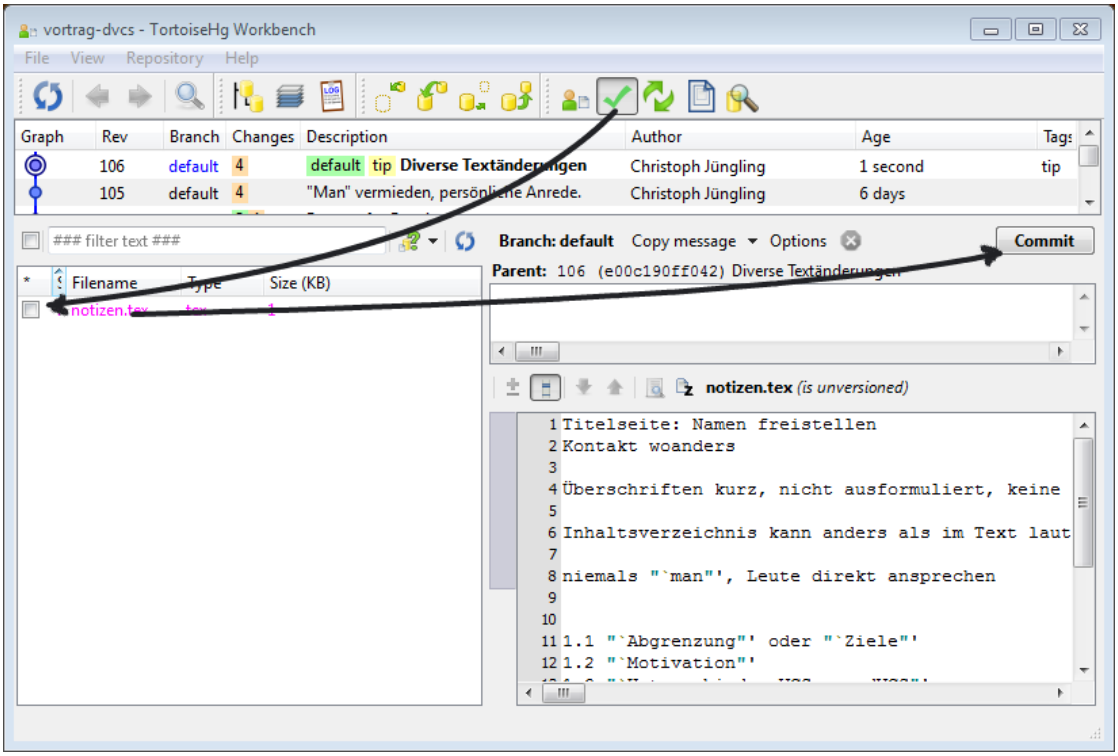

Abbildung 20: Dateien hinzufügen

3.2 In ein Projekt einsteigen

Wer in ein bereits unter Quellcodeverwaltung stehendes Access-Projekt zusätzlich einsteigen will, braucht zunächst ein Clone des bestehenden Projekt-Repositories, welches am besten von dem zentralen Server (sofern vorhanden) geholt wird (Abb. 21). Es gibt natürlich noch weitere mögliche Quellen:

- `x:\myapp`
- `\\servername\freigabe\myapp`
- `https://www.irgendwo.de/myapp`
- Lokale Festplatte oder USB-Stick: `myapp.hg`

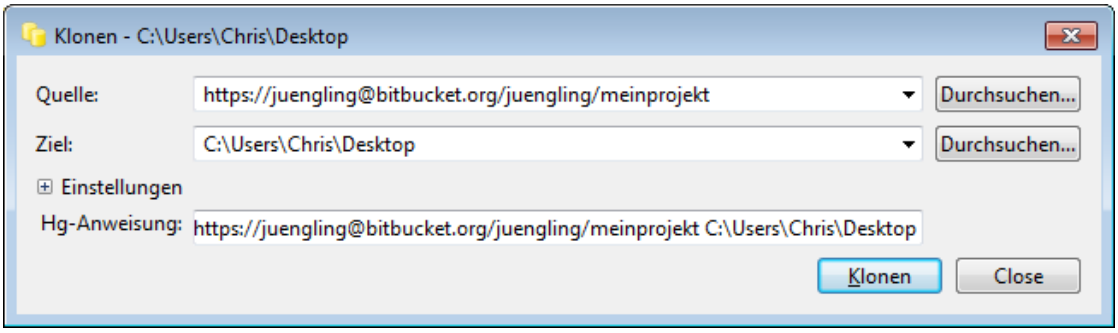

Abbildung 21: Clonen eines Repositories

Danach wird Access gestartet und mittels OASIS eine Datenbank neu erstellt. Dabei muss als Dateiname der Basisnamen der `.oasis`-Datei und das selbe Verzeichnis zum Speichern gewählt werden. Dann liest OASIS deren Einstellungen und ist gleich richtig konfiguriert.

Ebenso muss darauf geachtet werden, dass der selbe Access-Datenbanktyp (accdb oder mdb) und die Datenbankversion verwendet wird! Wie im Vortrag in Hannover unfreiwillig gezeigt wurde, gibt es sonst einige unerwünschte Seiteneffekte. Wichtig!

3.3 Korrupte Access-Datenbank neu erstellen

Dies ist im Grunde der gleiche Vorgang wie Kap. 3.2 In ein Projekt einsteigen (S. 20) beschrieben, nur dass hier das Repository bereits vorhanden ist. Eventuell muss vorher noch die korrupte Datenbank umbenannt werden, bevor die neue erstellt wird.

Das Ergebnis ist dann der Stand, wie er zuletzt im Repository vorhanden war. Hast du hinreichend oft committet, ist also nicht viel verloren.

3.4 Einfache Änderungen

Dies ist der ganz normale Fall beim Programmieren, wenn ausnahmsweise mal nichts dazwischen kommt. Unsere Vorgehensweise wird in etwa der folgenden entsprechen:

1. MS Access™: Entwickeln und Testen
2. OASIS: Exportieren (gerne auch öfter, sicher ist sicher)
3. TortoiseHg: Ggf. neue Dateien hinzufügen
4. TortoiseHg: Commit
5. Wiederholen der vorigen Schritte so oft wie nötig: MS Access™ → OASIS → TortoiseHg

3.5 Änderungen einspielen

Um Änderungen von anderen Programmierern einzuspielen, wird das Kommando Pull gefolgt von einem Update (evtl. Merge) verwendet. Das Repository muss dafür natürlich über wenigstens eine Quelle verfügen. Dadurch werden Änderungen von einem entfernten Repository geholt und auf die Dateien im Arbeitsverzeichnis angewendet. Dieser Vorgang wird in der Regel von einem Commit abgeschlossen, wodurch die Änderungen (und deren historische Herkunft) dem Repository wieder hinzugefügt werden.

Alternativ zum „Pull“ vom entfernten Repository kann auch eine Bundle-Datei importiert werden.

In der Workbench ist dann grafisch zu erkennen, welchen Weg die Programmänderungen genommen haben (siehe Abb. 22 von Revision 2 auf 4).

Grafische	Rev	Zweig	Änderun	Beschreibung	Autor
	9+	default		★ Arbeitsverzeichnis ★	Christoph Jüngling
	9	default	1	default tip **Applikationsname geändert**	Christoph Jüngling
	8	default	1	Applikationsname geänderrt	Christoph Jüngling
	7	Access 2007	1	Access 2007 Applikationsname geänderrt	Christoph Jüngling
	6	Access 2007	1	Ignore-Filter erweitert	Christoph Jüngling
	5	Access 2007	2	Projekt in Access 2007 per OASIS importiert und erneu	Christoph Jüngling
	4	default	1	Zusammengeführt	Christoph Juengling
	3	default	1	Titel des About-Formulars geändert	Christoph Juengling
	2	default	1	Titel des About-Formulars in "Über" geändert	Inga Pengar
	1	default	2 1	Hgignore erweitert, Formular "About" und Screenshot	Christoph Juengling
	0	default	4	OASIS-Konfiguration, Globale Deklarationen	Christoph Juengling

Abbildung 22: Beispiel mit Branch und Merge

4 Fazit

Mercurial (Hg) ist keineswegs das einzige Quellcodeverwaltungssystem und es ist auch nicht das einzige, das den Zusatz „verteilt“ („distributed“) trägt. Ich habe dieses System für meine Arbeit und diesen Vortrag ausgesucht, weil ich daran sehr einfach und übersichtlich zeigen konnte, wie man mit einem dVCS auch in Access arbeiten kann. Und das, obwohl Microsoft selbst einen ganz anderen Weg geht, MS Access™ an die eigenen Systeme Team Foundation Server und das ältere Visual SourceSafe anzubinden.

Dennoch bin ich der Ansicht, dass die Arbeit mit Mercurial einige Vorteile gegenüber den hauseigenen Systemen von Microsoft hat. Mein persönliches Lieblingsfeature ist der konsequente Verzicht auf Dateisperren, die mir in einem früheren Projekt mit urlaubswütigen Kollegen schon so manches Problem beschert haben. Zwar ist ein „Merge“ auch keineswegs immer völlig stressfrei, aber insgesamt doch leichter zu handhaben als der Versuch, den Admin zu erreichen, ihn davon zu überzeugen, dass er nun *schon wieder* etwas tun muss, was er selbst für völlig überflüssig hält, ihn anschließend zum Kaffee einzuladen und sich von seinen Problemen mit Usern erzählen zu lassen.

Auch die letzte Aktion, als sowohl der Team Foundation Server neu aufgebaut und die Repositories migriert wurden als auch ich fast zeitgleich eine neue Maschine bekam, ist mir noch in guter Erinnerung. Das lokale Arbeitsverzeichnis (einschließlich der Access-Datenbank) war schlichtweg unbrauchbar geworden und bis ein störungsfreies Arbeiten wieder möglich war, ging locker ein halber Tag mit administrativer Unterstützung drauf. Bei Mercurial wäre das Umkopieren des Arbeitsverzeichnisses einschließlich des lokalen Repositories im Windows-Explorer ein leichtes gewesen, und der geänderte Pfad zum Server (dem „entfernten Repository“) in Sekundenschnelle über TortoiseHg geändert gewesen. Genau so sieht stressfreies Arbeiten aus.

Das war es erstmal mit diesem Script. Wenn darüber hinaus Fragen offengeblieben (oder gerade dadurch erst entstanden) sind, werde ich mich gerne bemühen, diese in Karls Forum auf `http://www.donkarl.com/Forum` zu beantworten.

Ach ja, und der Admin freut sich auch über Mercurial: Das mit dem Kaffee machen wir heute nämlich immer noch, aber seit wir Mercurial verwenden, hat sich unser Verhältnis deutlich entspannt :-)

Glossary

changeset Eine Menge von Veränderungen an Dateien. 24

clone Kopie eines Repositories, wobei Hg automatisch einen Link zu seiner Basisversion einfügt. 12, 17, 20

commit Ein Commit stellt gemachte Änderungen der Quellcodeverwaltung zur Verfügung, die daraus ein Changeset macht. 22

cVCS centralized version control system, dt. zentralisiertes Quellcodeverwaltungssystem. 8, 11, 17

dVCS distributed version control system, dt. verteiltes Quellcodeverwaltungssystem. 8, 10, 11, 17, 23

Git verteiltes Versionskontrollsystem, Open Source, `http://git-scm.com`. 7

Hg Kurzname für „Mercurial“. 24

hgsubversion eine Kurzname für "‘Mercurial"’ (Hg)-Extension zur Integration mit einem Subversion (Svn)-Server, Open-Source, `http://mercurial.selenic.com/wiki/HgSubversion` und `https://bitbucket.org/durin42/hgsubversion`. *siehe* Mercurial

IDE integrated development environment. 9

Kaffee ist vielleicht das einzige Getränk, das guten Gewissens als Treibstoff für die Softwareentwicklung angesehen werden kann, `http://de.wikipedia.org/Wiki/Kaffee`. 23

Mercurial (Hg) ist ein verteiltes Versionskontrollsystem, Open Source, `http://mercurial.selenic.com`. 5–7, 10, 11, 15, 16, 18, 23, 25

merge Ein Merge ist die Zusammenführung des Codes aus verschiedenen Zweigen. 22, 23

MS Access™ Microsoft Access, Datenbanksystem, `http://office.microsoft.com/de-de/access/`. 5, 6, 9, 22, 23

MS Word™ Microsoft Word, Textverarbeitungssystem, http://office.microsoft.com/de-de/word/. 9

MSSCCI Microsoft Source Code Control Integration. 10, 11

OASIS Office Access Sourcecode Integration System, Autor: Bernd Gilles, http://www.dev2dev.de/oasis.html. 6, 10–15, 20–22

pull Mit dem Kommando „Pull“ holt man sich Changesets aus einem entfernten Repository. 22

push Mit dem Kommando „Push“ schickt man Changesets an ein entferntes Repository. 20

Subversion (SVN) zentrales Versionskontrollsystem, Open Source, http://subversion.apache.org. 15, 25

Svn Subversion. 24

Team Foundation Server (TFS) zentrales Versionskontrollsystem von Microsoft. 23

TFS Team Foundation Server. 11

TortoiseHg ist eine Windows Shell-Erweiterung für das verteilte Versionskontrollsystem Mercurial, Open Source, http://tortoisehg.bitbucket.org/de/. 6, 10, 11, 16, 17, 19, 20, 22, 23

TortoiseSVN ist eine Windows Shell-Erweiterung für das Versionskontrollsystem Subversion, Open Source, http://tortoisesvn.net. 15

update Ein Update nach (oder unabhängig von) einem Kommando „Pull“ spielt die Änderungen des ausgewählten Changesets in das Arbeitsverzeichnis ein. 22

VCS version control system, dt. Quellcodeverwaltungssystem oder Versionskontrollsystem. 7, 10, 11

Visual SourceSafe (VSS) Visual SourceSafe, ein altes filesystembasiertes Versionskontrollsystem von Microsoft. 23

VSS Visual SourceSafe. 11

Windows™ Graphische Benutzeroberfläche von Microsoft. 6

Index