

Migration von MKS zu Subversion

Motivation, Durchführung, Erfahrungen

Dr. Markus Liebelt, T-Systems ES/SI/QBS/SEQ

Version 1.0
Stand 22.09.2008



Zusammenfassung

T-Systems hat in den letzten zwei Jahren einen Standard für die Softwareentwicklungsumgebung gesetzt, Subversion ist eine der wesentlichen Komponenten darin. Durch die Nutzung dieses zentral bereitgestellten und betriebenen Toolsets wurden Kosten deutlich reduziert. Da die T-Systems aber wie viele anderen Systemhäuser vor allem von lang laufenden Projekten lebt ist die aktuelle Herausforderung, laufende Projekte, die ein anderes Toolset nutzen, auf die SE Tools und damit auch auf Subversion zu migrieren.

Eines der häufig eingesetzten Werkzeuge ist das Konfigurationsmanagementwerkzeug Source Integrity von MKS. Source Integrity hat einige Eigenschaften, die von Subversion nicht oder nur unzureichend unterstützt werden. Zudem geht MKS von einem prinzipiell anderen Ansatz aus.

Der Vortrag schildert die Vorgehensweise, die kritischen Punkte in der Durchführung der Migration und die Erfahrungen, die während der Migration von MKS nach Subversion gesammelt werden konnten. Er erfolgt aus der Sicht von Anwendern und Projektadministratoren und schildert vor allem die Schwierigkeiten, die die Menschen in den laufenden Projekten mit der Migration hatten und wie diese bewältigt werden konnten.

Inhaltsverzeichnis

| | | |
|-----|--|----|
| 1 | Kontext..... | 4 |
| 2 | Einsatz von Source Integrity..... | 5 |
| 3 | Unterschiede im Konzept Subversion und Source Integrity..... | 6 |
| 3.1 | Pessimistisch nach Optimistisch..... | 6 |
| 3.2 | „Cherry Picking“ über Change Packages | 7 |
| 3.3 | Nomenklatur..... | 7 |
| 4 | Lösungskonzept | 8 |
| 5 | Migrationskonzept | 9 |
| 5.1 | Abbildung auf den neuen Prozess..... | 9 |
| 5.2 | Definition der Umstellung..... | 10 |
| 5.3 | Zeitpunkt der Umstellung..... | 10 |
| 5.4 | Reihenfolge | 10 |
| 5.5 | Übernahme der Historie..... | 10 |
| 6 | Umsetzung..... | 11 |
| 7 | Erfahrungen..... | 11 |
| | Quellenverzeichnis | 12 |

1 Kontext

T-Systems hat in den letzten zwei Jahren einen Standard für die Software-Entwicklungsumgebung gesetzt. Subversion ist eine der wesentlichen Komponenten darin. Die Gründe für die Wahl von Subversion waren damals:

- § Subversion war damals der absehbare Nachfolger von CVS.
- § Subversion hatte die richtigen Erweiterungen gegenüber CVS.
- § Subversion war für den Einsatz in verteilten Teams geeignet.
- § Subversion war das richtige Werkzeug für kleine und mittlere Projekte, und hatte das Potenzial, auch in großen Projekten und komplexen Prozessen (mit den richtigen Werkzeugen an der Seite) eingesetzt zu werden.

Die T-Systems hat sich damals bewusst für eine Open Source Lösung entschieden, auch wenn von verschiedener Seite die Vermutung geäußert wurde, dass dies (auch) dem (fehlenden) Preis geschuldet war (zu unrecht). Natürlich mussten wir uns damals mit den Verfechtern der Nutzung von kommerziellen Werkzeugen auseinandersetzen. Es gab damals eine heterogene Landschaft mit einzelnen Inseln der Seeligen. So gab es Bereiche und Gesellschaften, in denen Change Synergy eingesetzt wurde, an anderen Stellen wurde massiv Source Integrity von MKS eingesetzt.

Die Herausforderungen der Zeit (starke Globalisierung, Verlagerung von Geschäft zu unserem Offshore-Partner, ...) steigt die Notwendigkeit zur Standardisierung. Diese umfasst:

- § Homogene Nutzung der Werkzeuge in allen Projekten:
 - à Mitarbeiter benötigen beim Wechsel in Projekten keine Werkzeugschulung
 - à Investition in Umsetzung der definierten Prozesse kann im wesentlichen zentral, nur in Einzelfällen zusätzlich im Projekt erfolgen
- § Zentraler Betrieb spart Ressourcen:
 - à Aktuell können ~200 Projekte von 6 Administratoren in Deutschland sowie einigen Mitarbeitern in Weißrussland und Indien im first und second level support unterstützt werden.
 - à Es werden deutlich weniger Maschinen usw. benötigt, um die gleiche Anzahl Projekte zu treiben.

Inzwischen werden die definierten Werkzeuge in vielen Projekten eingesetzt. Aber das genügt nicht:

- § Bestehende Projekte können nur dann übergeben werden, wenn die Erfahrungen und Fähigkeiten vorhanden sind. Diese gehen aber durch normale Fluktuation, aber auch Umstrukturierungen immer wieder verloren und müssen mühsam wieder aufgebaut werden.
- § Bestehende Lizenzverträge sind oft so gestaltet, dass Lizenzen nicht teilweise zurückgegeben werden können. D.h. für eine Wartungsverlängerung fallen die immer gleich hohen Wartungskosten an, auch wenn die Nutzung in Summe abnimmt.

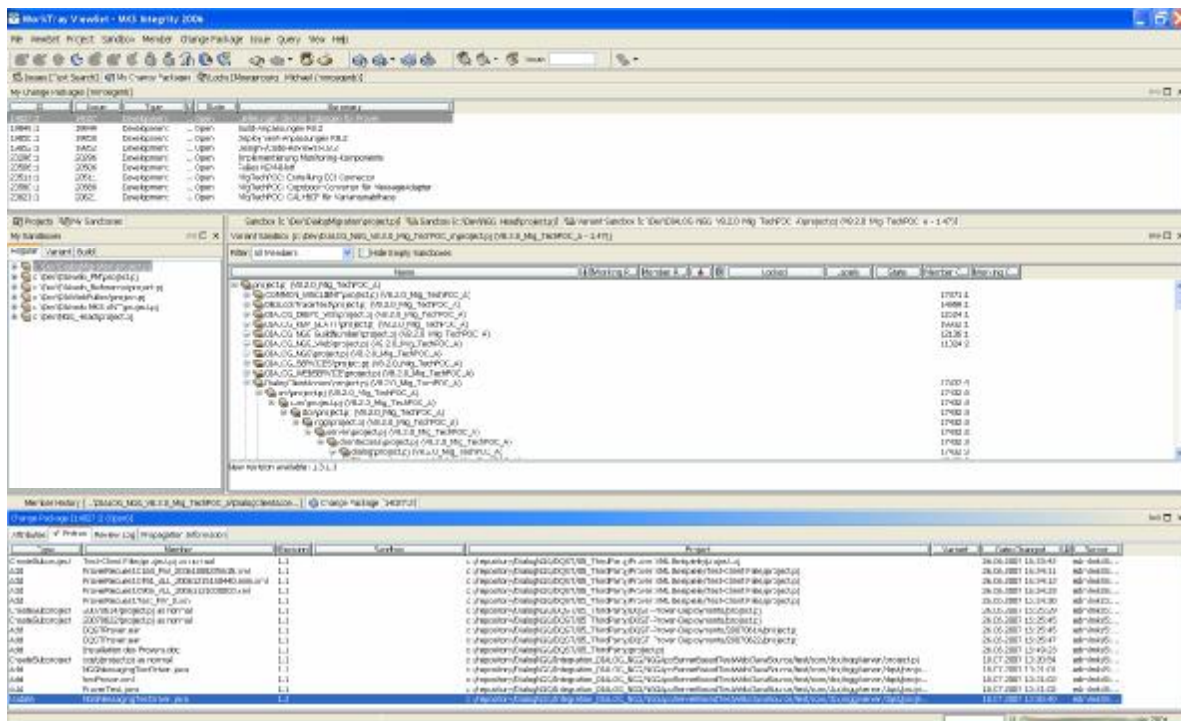
Aber wie können laufende Projekte davon überzeugt werden, dass eine Migration sich für sie mittelfristig auszahlt? Und wie muss diese Migration dann erfolgen, damit die Migration

- § ein kalkuliertes und beherrschbares Risiko darstellt
- § zur rechten Zeit erfolgt
- § mit jedem neu migrierten Projekt billiger und einfacher wird?

Dieser Vortrag stellt einige der Erfahrungen mit der Migration von Source Integrity von MKS hin zu Subversion dar. Dabei wird im Vortrag vor allem auf die Aspekte der Migration zu Subversion eingegangen. Es soll hier nicht verschwiegen werden, dass in großen Projekten neben Subversion Polarion eingesetzt wird, um den kompletten Prozess der Werkzeugkette, die mit den MKS Werkzeugen teilweise abgedeckt wurde, umzusetzen. Dies steht jedoch bei diesem Vortrag nicht im Zentrum.

2 Einsatz von Source Integrity

Source Integrity als Teil der Toolsuite von MKS wird von vielen Projekten rein als Versionsmanagement eingesetzt. Versionsmanagement (im Unterschied zu Konfigurationsmanagement) beschäftigt sich mit der Historie einzelner Software Komponenten, hat dagegen keine Möglichkeiten, aus diesen Softwarekomponenten Software Konfigurationsitems zu bilden. In diesem Fall hat Source Integrity gegenüber Subversion keine Vorteile. Source Integrity unterscheidet sich im Einsatz oft dadurch, dass so genannte Change Packages gebildet werden, und Entwickler Änderungen auf Change Packages commiten. Ein Change Package sammelt damit alle Änderungen eines Entwicklers auf, diese werden vom KM-System von den Änderungen anderer Entwickler getrennt gehalten.



Meist wird Source Integrity aber im Kontext eines Workflows eingesetzt, in dem über Tags auf Softwarekomponenten verteilt werden, und diese Tags dann den Konfigurationsmanagement-Prozess (kurz: KM-Prozess) steuern. So ist es z.B. möglich, alle Change Packages, deren Status den Zustand „erledigt“ haben, zusammenzuziehen zu einer neuen Konfiguration. Diese Nutzung ist in Subversion alleine nicht abbildbar, ist aber im Zusammenspiel mit Erweiterungen wie CollabNet oder Polarion (aber meist anders abgebildet) möglich. Diese und einige weitere kleine Unterschiede erschweren den Umstieg von Projekten, die schon jahrelang mit einem proprietären KM-System gearbeitet haben.

Dagegen ist Source Integrity ein sehr mächtiges und schweres Werkzeug, das zuerst einmal gelernt werden muss. Auch nach Jahren der Nutzung kennen Anwender nicht alle Aspekte ihrer Umgebung, einzelne Aspekte davon sind relativ unbefriedigend. Oft genannt wurden:

- § Ständig Serververbindung des MKS Client schadet häufig, ist selten benötigt. Dies wegzukonfigurieren wird in den seltensten Fällen durchgeführt.
- § Der Umgang mit Sandboxen im Gegensatz zu einem Checkout in Subversion ist (vor allem in verteilten Umgebungen mit wenig performantem Netz) aufwändig.
- § Manche Feature sind gefährlich und oft nicht klar.

3 Unterschiede im Konzept Subversion und Source Integrity

Was sind nun die wesentlichen Unterschiede von Subversion und Source Integrity?

3.1 Pessimistisch nach Optimistisch

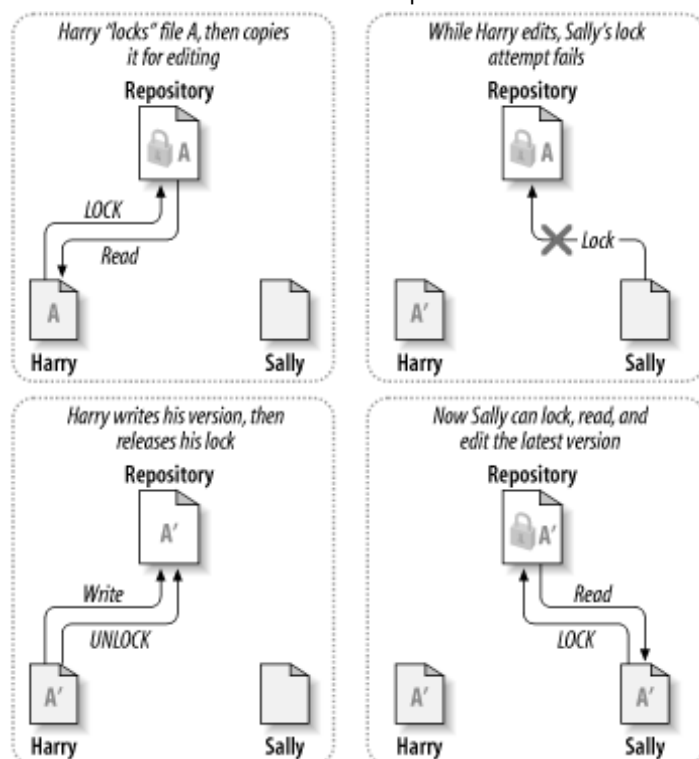


Abbildung 1: Lock-Modify-Unlock Strategie (nach [Red Book])

Source Integrity verlangt in der Verwendung das so genannte pessimistische Sperren. Dies bedeutet kurz, dass vor einer durchzuführenden Änderung ein Entwickler die zu ändernden Files sperrt, diese dann ändert, und anschließend die zuvor gesperrten Dateien wieder freigibt.

Dies ist seit vielen Jahren die bevorzugte Vorgehensweise der kommerziellen KM-Systeme. Diese Strategie ist dann auch unverzichtbar, wenn Konflikte grundsätzlich nicht erlaubt sein sollen, oder diese große Auswirkungen haben.

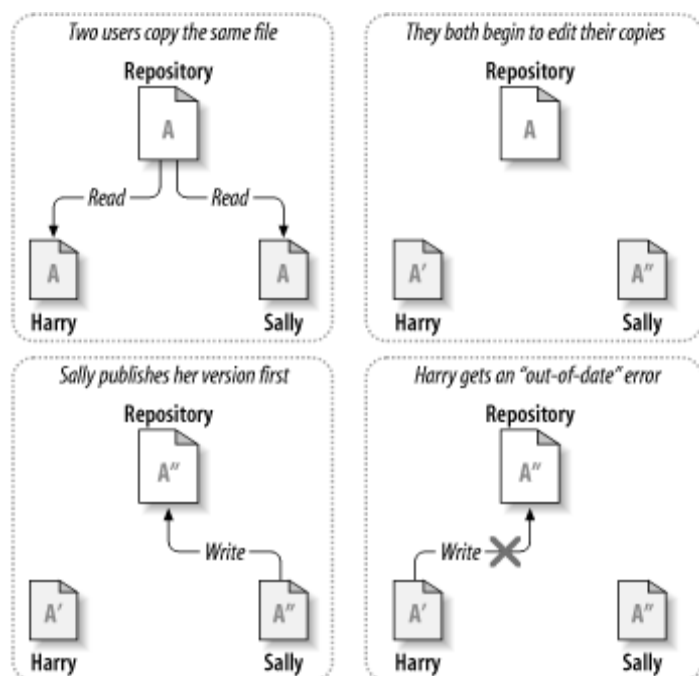


Abbildung 2: Copy-Modify-Merge Strategie (nach [Red Book])

Subversion dagegen arbeitet im Regelfall mit einem optimistischen Sperrkonzept. Hier hat jeder Entwickler eine Kopie des Sourcebaums und arbeitet unabhängig von anderen Entwicklern lokal auf einer Kopie. Er macht seine Änderungen, und gibt diese dann frei. Tauchen hier Konflikte auf, ist der Entwickler anschließend gezwungen, die Änderungen, die er selbst durchführen möchte, mit den Änderungen von anderen Entwicklern zu mergen, es sei denn (was der Normalfall ist), es treten keine Konflikte auf.

Bei genauer Betrachtung stellt man meist fest, dass optimistische Sperrverfahren im wesentlichen Vorteile haben:

1. Es können mehr Entwickler unabhängig voneinander auch komplexere Aktionen durchführen, ohne sich in die Quere zu kommen.

2. Es gibt keine Deadlocks bei Änderungen. Deadlocks sind hier Situationen, in denen zwei Leute zwei voneinander unabhängige Änderungen durchführen wollen, die sich so ungeschickt überkreuzen, dass während der Änderung jeder der Bearbeiter eine blockierte Ressource des anderen benötigt.
3. Locks werden teils zu früh geholt („Wer weiß, ob ich die Änderung nachher noch durchführen kann?“) oder zu spät zurückgegeben („Vielleicht komme ich heute Mittag noch zu ...“). Dadurch werden Änderungen von anderen verhindert.
4. Änderungen, die Folgeänderungen nach sich ziehen (z.B. größere Refactorings) können in Einzelfällen „stecken bleiben“.

Es sollte aber nicht verschwiegen werden, dass es Umstände geben kann, in denen ein pessimistisches Sperrverfahren eingesetzt werden sollte. Dies können z.B. sein:

1. Gleichzeitige Änderungen verschiedener Bearbeiter können nicht werkzeuggestützt abgeglichen und „gemerged“ werden. So ist ein Merge von den meisten Binärformaten schwierig, bei manchen sogar unmöglich.
2. Konflikte führen zu nicht hinnehmbaren Konsequenzen. Hier ist eine Sperre vor Änderung einfach vernünftig.

3.2 „Cherry Picking“ über Change Packages

Da Source Integrity Nutzer im Regelfall auf Change Packages arbeiten, besteht hier die Möglichkeit, erst zum Zeitpunkt der Integration zu entscheiden, welche Änderungen von welchen Entwicklern in einem Release ausgenommen werden sollen. Das hört sich auf den ersten Blick attraktiv an, kann aber Probleme mit sich bringen:

- § Viele Änderungen sind zwar syntaktisch unabhängig (d.h. werden in unterschiedlichen Dateien durchgeführt), haben aber semantische Abhängigkeiten, die nicht offensichtlich sind. Diese müssen dann entsprechend markiert und erkannt werden.
- § Eine späte Integration kann dazu führen, dass erst sehr spät Probleme erkannt werden, die aufgrund unabhängig entwickelter, aber nachher gemeinsam eingesetzter Codeteile entstehen. Durch die späte Integration bleibt nur wenig Zeit für die Problembehebung, so dass evtl. nur schlecht getesteter, unreifer Code ausgeliefert wird.

Dies wird von den meisten Projekten aber nur in Ausnahmefällen eingesetzt. Der normale Entwickler arbeitet auf dem „Head“ des aktuellen Branches, und bekommt damit alle Änderungen von allen Entwicklern automatisch bei einem „resync“ zur Verfügung gestellt.

In Subversion ist Cherry Picking technisch nicht so einfach umsetzbar. In der Open Source Community ist man es auch gewohnt, sehr oft und frühzeitig zu integrieren. Hier hat man also eine technische Möglichkeit, die im Werkzeug nicht umsetzbar ist, durch eine entsprechende Kultur (erfolgreich) angegangen. Falls notwendig, können isolierte Entwicklungen einfach erreicht werden, indem so genannte Entwickler-Banches eröffnet werden.

3.3 Nomenklatur

Die zwar ähnliche gebrauchte, in der Bedeutung aber leicht (oder stark) unterschiedliche Bedeutung der Begriffe bringt einige Probleme mit sich. Hier nur einige wenige Beispiele.

| Begriffe | Source Integrity | Subversion |
|----------------|--|---|
| Check Out | Reserviert ein Member für die Bearbeitung durch den Anforderer | Legt eine lokale Kopie beim Bearbeiter an, ohne diese zu reservieren. |
| Check In | Gibt Veränderungen an den Server zurück. Dies führt zu einer neuen Revision. | Heißt in Subversion „commit“. |
| Resynchronize | Führt einen Abgleich mit dem Server durch. | Heißt in Subversion „update“ |
| Revert Members | Setzt Veränderungen zurück auf den Stand, den ein Member auf dem Server hat. | Identisch. |
| Branch | Abspaltung eines Verzeichnisbaums auf Basis eines Checkpoints. | Abspaltung eines Verzeichnisbaums auf Basis einer Revision. |
| Tag | Heißt hier Label, markiert alle Member einer | Technisch identisch zum Branch, meist |

| | | |
|----------------------|--|---|
| | bestimmten Revision mit einer eindeutigen Markierung. | aber durch Konvention oder technische Einstellungen der Berechtigungen unveränderlich gehandhabt. |
| Change Set / Package | Das Change Package sammelt alle Änderungen eines Entwicklers. Es entspricht damit dem Software Configuration Item. Der Konfigurationsmanager entscheidet zum Integrationszeitpunkt, welche Change Packages integriert werden sollen. | Ein Change Set ist die Sammlung aller Änderungen, die ein Entwickler zu einem Zeitpunkt in Subversion speichert. Diese Änderungen werden entweder alle zusammen oder keine davon gespeichert (transaktional). |

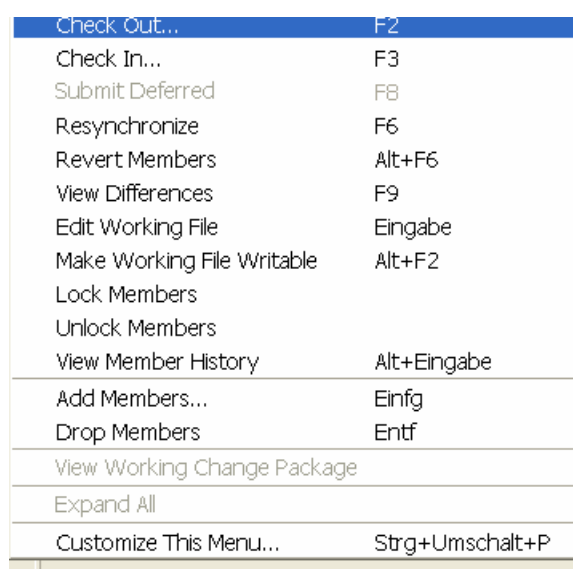


Abbildung 3: Source Integrity Member Menu

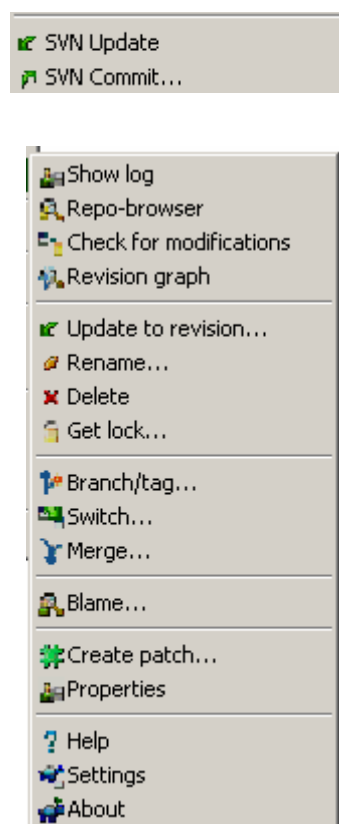


Abbildung 4: Subversion Menu

4 Lösungskonzept

Die bisherigen Auslassungen haben verdeutlicht, dass der Umstieg von MKS auf Subversion im Projekt nicht ohne Schwierigkeiten von sich geht. Wie kann dieser Umstieg erleichtert werden, welche Reihenfolge ist hier zu empfehlen? Dabei möchte im hier Lösungskonzept, Migrationskonzept und Umsetzung unterscheiden.

Das Lösungskonzept gibt folgendes vor:

1. Jedes Projekt definiert zuerst den bisherigen KM-Prozess auf einer abstrakten Ebene.
2. Typische mögliche KM-Prozesse in Subversion werden vorgestellt.
3. Die Unterschiede der bisherigen zu den neu möglichen Prozessen werden diskutiert.
4. Es wird entschieden, wie die grundsätzliche Strategie der neuen KM-Prozesse aussieht.
5. Welche Zöpfe können beim Umstieg abgeschnitten werden?
6. Die KM-Prozesse werden von innen (Kern) nach außen (irgendwann notwendig, aber zu Beginn verzichtbar) entwickelt.
7. (Optional) Es wird entschieden, ob die bisherigen Prozesse den Einsatz von Polarion verlangen, um die KM-Prozesse, die notwendig sind, abbilden zu können.

Der Unterschied hier zum Migrationskonzept besteht meines Erachtens darin, dass man zuerst einmal zielorientiert an die Aufgabe herangeht: was muss umgesetzt werden, welche Möglichkeiten der Prozessdefinition habe ich, was ist verzichtbar. Es geht zuerst einmal nicht darum, ob in welcher Reihenfolge welche Änderungen durchgeführt werden können. Der Mehrwert eines Beraters, der schon mehrfach dies mit

Projekten durchgeführt hat, besteht darin, dass er Fehlentwicklungen bremsen und aus falsche Annahmen hinweisen kann.

Wir hatten bei den folgenden Bausteinen des Lösungskonzepts die meisten Schwierigkeiten:

- Branchingstrategie:** Welche Branches benötigen wir im Projekt zu welchem Zeitpunkt?
Die unten stehende Abbildung ist ein Beispiel aus einem Projekt (anonymisiert), wie sich dort die Branches entwickelt haben.
- Kein Cherry Picking:** Der Branch definiert (im Wesentlichen), welche Änderungen in welchem Release enthalten sein werden.
- Prozess vs. Technik:** „Wir haben dann immer ... Aber warum haben Sie ... Das ist doch jetzt egal, wir haben immer ...“
Es ist für viele Mitarbeiter schwierig, die Arbeitsebene zu verlassen und wieder über die Gründe nachzudenken, warum die Prozesse so definiert wurden.
- Locking:** „Wie kann ein Entwickler sich sicher sein, dass ihm niemand in die Quere kommt?“
Oft geäußert, auf Nachfrage war es meist schwierig zu erklären, ob dieses Problem in der Vergangenheit schon bestand.

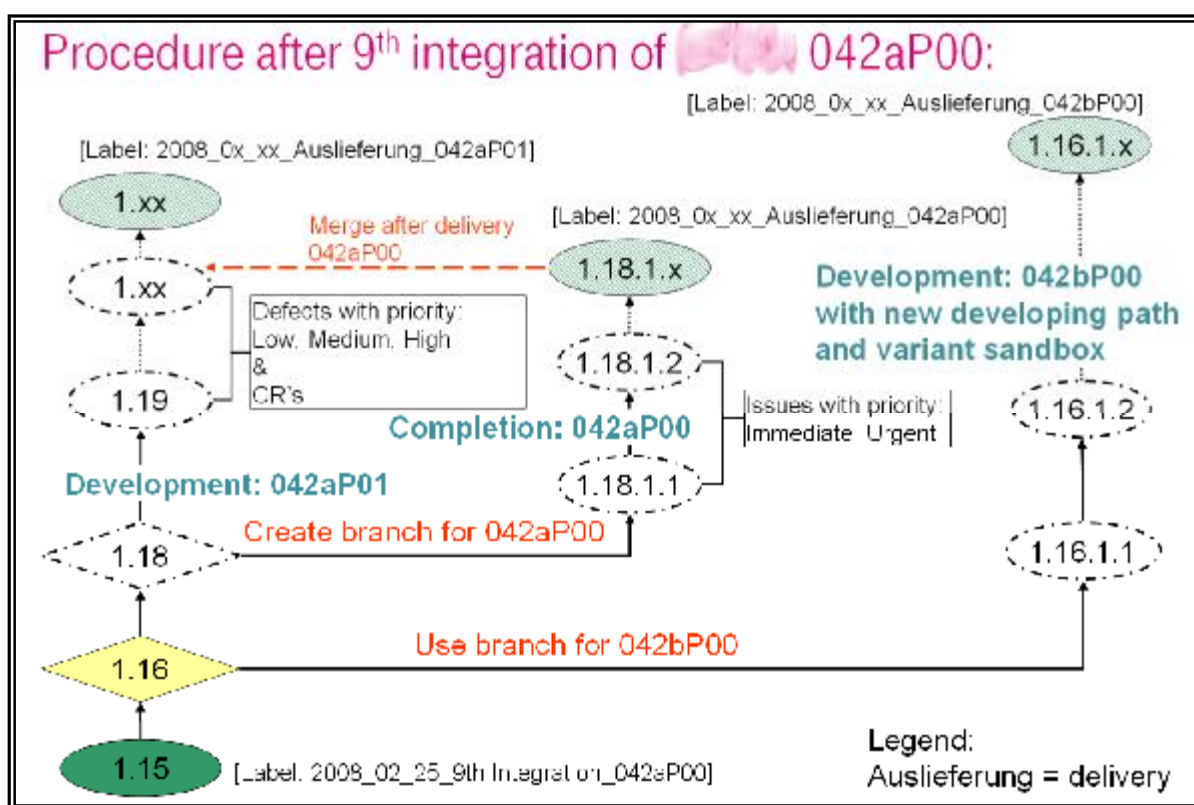


Abbildung 5: Beispiel für eine Branchingstrategie

5 Migrationskonzept

Nachdem das Lösungskonzept von den Verantwortlichen (mit unserer Unterstützung) gemeinsam erarbeitet wurde, kann das Migrationskonzept erstellt werden. Hier geht es vor allem um die folgenden Fragen:

- § Wie wird der bisherige Prozess auf den neuen Prozess abgebildet?
- § Wie erfolgt technisch die Umstellung?
- § Wann ist der richtige Zeitpunkt dafür?
- § Welche Teile werden in welcher Reihenfolge umgestellt?
- § Wie viel Historie muss übernommen werden?

In der Migrationsphase sollten die grundsätzlichen Fragen geklärt sein, hier geht es eher um die praktische (und pragmatische) Umsetzung. Hier können auch taktische Fragen geklärt werden.

5.1 Abbildung auf den neuen Prozess

Auch wenn man sich im Projekt über den alten und neuen Prozess verständigt hat, so ist die Abbildung nicht immer ohne Probleme. Dabei müssen oft auch taktische Fragen geklärt werden:

- § Stellen wir die Begriffe im Prozess so um, dass sie den neuen Werkzeugen eher entsprechen oder bleiben wir bei den alten Begrifflichkeiten, um möglichst viele unserer Entwickler abzuholen?
- § Wie geht man damit um, dass Dinge im bisherigen Prozess entfallen, die im neuen Prozess einfach nicht abgebildet sind? Hier sollte noch einmal geprüft werden, dass dies das Verständnis von allen ist.
- § Wo werden die Teile des Prozesses abgebildet, die bisher in einem Werkzeug umgesetzt wurden, im neuen Werkzeug aber nicht oder nur unzureichend unterstützt werden?

5.2 Definition der Umstellung

Es ist im Projekt zu klären, was genau umzustellen ist. Meist kann ein großes Projekt in mehrere kleine, von einander unabhängige Teile unterteilt werden. Hier empfiehlt es sich, folgendes zu tun:

- § Prototypische Umstellung eines kleinen unabhängigen Teils.
- § Umstellung der Entwicklung für diesen Teil, Dokumentation der Erfahrungen und Verfeinerung der Prozesse und Werkzeuge.
- § Bei erfolgreicher Arbeit Umstellung des nächsten unabhängigen Teils.

Ist diese Art der Vorgehensweise erfolgreich, so kann inner halb von wenigen Monaten ein Projekt komplett umgestellt werden.

5.3 Zeitpunkt der Umstellung

Es ist darauf zu achten, dass die Umstellung nicht zu einem Zeitpunkt erfolgt, in dem das Projekt kritisch unterwegs ist. Gute Zeitpunkte für die Umstellung sind:

- § Nach einer erfolgreichen Auslieferung
- § Stabile Teile, die aktuell nicht verändert werden

5.4 Reihenfolge

Folgende Reihenfolge sollte eingehalten werden:

- § Umstellung für den Piloten
- § Test im Piloten
- § ...

Dabei sollten möglichst keine Abhängigkeiten zur Entwicklungszeit der einzelnen Teile zueinander bestehen. Je nach verwendeter Programmiersprache gibt es hier Mechanismen, um die Entwicklung in MKS und Subversion von einander getrennt zu halten.

Es sollte ebenfalls darauf geachtet werden, dass die meisten Entwickler die meiste Zeit mit nur einem Werkzeug arbeiten müssen, und nur in Ausnahmefällen in einer Übergangszeit sowohl das alte wie auch das neue Werkzeug verwenden müssen.

5.5 Übernahme der Historie

Grundsätzlich kann die Übernahme der Historie gelingen, es ist aber teils abzuraten, dies zu tun. Einige Gründe dafür:

- § Die Übernahme der Historie von 10 Jahren belastet das neue Repository von Beginn an. Uns sind zwar keine prinzipiellen Grenzen von Subversion bekannt, aber teils bereits hier das Betriebssystem schon Probleme. So kann in Linux ein Verzeichnis mit mehreren Millionen Einträgen sich mit der Zeit zu einem Problem auswachsen.
- § Es gibt zwar Konverter (uns von der Firma Polarion dankenswerterweise bereitgestellt) von verschiedenen KM-Werkzeugen zu Subversion, aber nicht immer gelingt die Migration der Historie ohne Probleme. Dabei kann nachträglich nicht einfach getestet werden, ob die Historie wirklich und funktional vollständig zur Verfügung steht.
- § Die Migration der Historie selbst ist ein aufwändiger Prozess, der viele Ressourcen bindet. Es muss einen triftigen Grund für die Umstellung geben.

Wir haben gute Erfahrungen damit gemacht, dass das bisherige System bis auf weiteres read-only zur Verfügung steht. Solange nicht alle Teile migriert sind, muss das Altsystem sowieso weiter betrieben werden, und damit ist das Altsystem weiter am Leben zu halten.

6 Umsetzung

Die Umsetzung beginnt meist mit einer Pilotphase, in der der Kernprozess soweit erprobt wird, dass das Projekt genügend Vertrauen in den neuen Ansatz erwirbt. Hier bietet es sich an, mit einem kleinen Teilprojekt zu starten, das unabhängig vom Rest entwickelt werden kann, so dass die Arbeit der Pilotphase nicht komplett verloren ist.

In dieser Phase werden auch erste echte Erfahrungen mit den (für die zu migrierenden Mitarbeiter) neuen Werkzeugen gesammelt. Dabei stellt sich dann heraus, wo die Mechanismen der alten Werkzeuge gut übertragbar sind, und wo größere Probleme bestehen, die gut über Schulungen unterstützt werden müssen. Die Erfahrungen mit den Werkzeugen müssen zum einem im Entwicklerhandbuch dokumentiert werden, zum anderen in die Konfigurationen von Client und Server einfließen. In dieser Phase werden noch öfter Änderungen vorgenommen, die dann über die verschiedenen Arbeitsplätze hinweg transportiert werden müssen. Hierbei machen die vielfältigen Konfigurationsmöglichkeiten von Subversion durchaus Schwierigkeiten, da die Veränderungen zeitgleich zu erfolgen haben, um einen Effekt zu erzielen.

Es ist meines Erachtens wichtig, dass ein später evtl. automatisierter Workflow zu Beginn händisch erprobt und auf Herz und Nieren getestet wird. Klar ist, dass Subversion zwar die Technik bereitstellt, um ein Versionsmanagement aufzubauen, ein echtes Konfigurationsmanagement ist damit nicht möglich.

7 Erfahrungen

Hier nun in aller Kürze die gemachten Erfahrungen der letzten 6 Monate, ohne Anspruch auf Vollständigkeit:

1. Source Integrity hat als echtes KM-System einige Möglichkeiten, die mit Subversion Bordmitteln nur unter Schwierigkeiten abgebildet werden können. Hier empfiehlt es sich, tief zu prüfen, wie wichtig welches Feature ist. Oft stellt sich dabei heraus, dass organisatorisch damit umgegangen werden kann.
2. Die Arbeit eines KM-Admins verändert sich massiv. Teils wird dabei aus einem Full-Time-Job eine Aufgabe, die man nebenher erledigen kann. Hier sollte man sich frühzeitig Gedanken über mögliche Konsequenzen machen. Die Gründe für die Abnahme der Arbeit liegen in der durch die Entwickler durchgeführten Integration und den nur rudimentär ausgebildeten KM-Prozess. Bei der Verwendung eines Werkzeugs wie Polarion verändert sich dies evtl. wieder.
Dies hat auch menschliche Auswirkungen. Meist führt die Umstellung zuerst einmal dazu, dass viel Aufwand in die Konfiguration und Definition des KM Prozesses investiert wird, auch wenn dies auch Sicht eines erfahrenen Subversionnutzers wenig Sinn macht. Kritisch ist, wenn ein KM-Admin der kommerziellen Software seit Jahren ausschließlich dies gemacht hat.
3. Die Konzepte von Source Integrity und Subversion sind teils nur mit Mühe aufeinander abbildbar. Dies führt zu den folgenden Notwendigkeiten:
 - a. Optimierte Prozesse von Source Integrity müssen in Subversion umgestellt werden.
 - b. Einige Feature (Cherry Picking) sind nur mit deutlichem Mehraufwand umsetzbar. Bei einer Veränderung der Methodik sinkt aber eher der Aufwand.
4. Die ähnliche Nomenklatur von Begriffen in Source Integrity und Subversion führt dazu, dass das Verständnis zu Beginn erschwert wird. Dies kann durch Workshops und mit der Zeit wachsende Glossare entschärft, aber nie völlig ausgeräumt werden.
5. MKS Experten bemängeln bei Subversion den Wust an Werkzeugen, der von unterschiedlichen Herstellern kommt, und nicht einfach in heterogenen Umgebungen aufgesetzt werden kann. Beispiele hier sind:
 - a. TortoiseSVN fragt hin und wieder nach, ob es auf eine neue Version umstellen soll. Dies ist nachgerade gefährlich.
 - b. So schön TortoiseSVN ist, gibt es kein vergleichbares Werkzeug für Unix, Linux oder Mac OS X. MKS stellt ein Werkzeug auf allen Plattformen bereit.
 - c. Der Windows Explorer, der Repo Browser, einige Dialoge und speziell die Integration von Diff- und Merge-Werkzeugen macht nicht immer einen guten Eindruck.

Quellenverzeichnis

[Red Book] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato: Version Control with Subversion.
Site. <http://svnbook.red-bean.com/>