SVN-Anbindung

Wiki-Versionen

Redaktionell arbeiten wir z. B. Neuerungen in der Branchversion im Intranet ein. Nur absolute Basics werden in Vorversionen, dann i. d. R. in der Trunk-Version angepasst. Diese werden dann bis zum Branch durchgemerged. Durch Veröffentlichung des Servicepack 1 zu Version 20.21 Ende Juni 2021 wurde z. B. die Version 20.21.1 zur neuen Trunk-Version und die neue Branchversion 20.21.1 wurde eingerichtet. Optional: Eine Angabe der Kalenderwoche nach der Programmversion (letzer Upload von KW…) auf der Startseite könnte dem Anwender helfen, zu entscheiden, welche Informationen ggf. zusätzlich per Patchnotes zu berücksichtigen sind, wenn er eine aktuellere Version einsetzt.

Internet und Intranet

Redaktionell möglich wäre z. B. eine allgemeine Wiki-Struktur mit Unterkategorien wie [[20191:start|20.19.1]] [[20192:start|20.19.2]] [[2020:start|20.20]]

2020:start|zu Namespace 2020__ Klärungsbedarf Namespacemanagement/Hyperlinks/Workflow bei Versionswechsel

Aber eine solche übergeordnete Seite werden wir in der Kundenversion **nicht** anbieten, wir steigen in der jeweiligen Version dann direkt auf der aktuellen Versionsseite ein. Abhängig von der Programmversion wird die jeweilige Hilfeseite geöffnet. Die trunk-Version nutzt eine vereinfachte Adressierung, zu der dann intern ein Redirekt auf die Versionsseite erfolgt: ⇒ https://wiki.crem-solutions.de/ Dies unterstützt die Einrichtung eines dauerhaft sinnvollen Webseitenlesezeichen im Browser des Kundens.

Die Branchvariante besteht nur im Intranet! Die Branchversion halten wir bis zu einem bestimmten Zeitpunkt mit der Trunk-Version parallel und entwickeln sie dann gezielt weiter. Hierzu nutzen wir eine Versorgung via TortoiseSVN und mergen nach bestimmten Regeln. Die Branchversion des iX-Wiki wird in der Aufbauphase **nicht** online gestellt. Dies ist der jeweiligen Trunkversion vorbehalten.

Z. B. ist ⇒ iX-Wiki Onlineversion 20.21.0 nur für den exklusiven Teil der Kunden relevant, welche schon die Version 20.21.0 einsetzen, während iX-Wiki Onlineversion 20.20.0 allen Kunden mit 20.20.0 bereitgestellt wird. (Stand dieser Konstellation: 01.08.2021). Wer den Pfad kennt, kann natürlich in seinem Browser auch eine andere Onlinevariante des iX-Wiki ansteuern, dies wird technisch nicht unterbunden.

Intranet	**Internet**	**aktuelle Ebene**
http://srv-dev/xampp/DokuWiki/Version.20.22.2/doku.php	keine Online-Variante	Branch
http://srv-dev/xampp/DokuWiki/Version.20.22.1/doku.php	https://wiki.crem-solutions.de/Version.20.20.1/doku.php	Trunk
http://srv-dev/xampp/DokuWiki/Version.20.22.0/doku.php	https://wiki.crem-solutions.de/Version.20.20.0/doku.php	Trunk-1
http://srv-dev/xampp/DokuWiki/Version.20.21.3/doku.php	https://wiki.crem-solutions.de/Version.20.21.3/doku.php	Trunk-2
http://srv-dev/xampp/DokuWiki/Version.20.21.2/doku.php	https://wiki.crem-solutions.de/Version.20.21.2/doku.php	Trunk-3
http://srv-dev/xampp/DokuWiki/Version.20.21.1/doku.php	https://wiki.crem-solutions.de/Version.20.21.1/doku.php	Trunk-4
http://srv-dev/xampp/DokuWiki/Version.20.21.0/doku.php	https://wiki.crem-solutions.de/Version.20.21.0/doku.php	outdated
http://srv-dev/xampp/DokuWiki/Version.20.20.3/doku.php	https://wiki.crem-solutions.de/Version.20.20.3/doku.php	outdated
http://srv-dev/xampp/DokuWiki/Version.20.20.0/doku.php	https://wiki.crem-solutions.de/Version.20.20.0/doku.php	outdated

Datenhaltung iX-Wiki und SVN

Versionskontrolle allgemein

Durch die Verwendung von TortoiseSVN (Subversion) werden bestimmte Dateien aus dem Wiki-Verzeichnis über `Add` und `Commit` gesichert. Aufgrund der Abfragen und den damit verbunden Merge-Prozessen ist eine bestimmte Abfolge im Umgang mit TortoiseSVN für die iX-Wiki-Varianten zu beachten. Hauptansprechpartner für die SVN-Konfiguration ist SMV.

Da innerhalb von DokuWiki, dem von iX-Wiki verwendeten Wikisystem, ebenfalls eine Versionskontrolle über das php-System operiert, ist zwischen Versionskonflikten in TortoiseSVN und Versionskonflikten in DokuWiki zu unterscheiden. Da wir nur bestimmte Dateien des DokuWikisystems in TortoiseSVN betrachten und nur für diese eine Versionkontrolle vorsehen, ist es möglich, dass spezielle DokuWikifunktionen hierdurch unterbrochen oder nicht optimal ausgeführt werden können. Solange diese Funktionen ebenfalls nur eine Wiki-interne Versionkontrolle betreffen, sehen wir das als unkritisch an. Im laufenden Betrieb sind bislang praktisch keine Versionskonflikte in Dokuwiki aufgetreten.

Welche Dateien oder Verzeichnisse von iX-Wiki unterliegen in TortoiseSVN einer Versionskontrolle?

- Baumstrukturen aus dem Vereichnis data\pages ⇒ Unterverzeichnisse (namespaces) und Wikiseiten (data\pages *.txt)
- Baumstrukturen aus dem Vereichnis data\index ⇒ Suchindex der Volltextsuche (data\index *.idx)
- Baumstrukturen aus dem Vereichnis data\media ⇒ Medienverzeichnisse und ihre Inhalte (data\media*.*)

Meta-Daten und Indexierungen sowie Verlauf über die Verzeichnisse `attic cache index locks media_attic media_meta meta tmp` sind nicht zwingend erforderlich. Sie unterstützen das DokuWiki-interne System eines Repositories und der Indexierung. So werden index-Dateien (relevant für die Volltextsuche) auch bei der Versionierung mit SVN berücksichtigt. Indexierungen können nachträglich anhand der pages-Seiten administrativ über den Searchindex-Manager erzeugt werden. Diese zusätzliche Indexierung sollte aufgrund von Erfahrungen bzgl. unvollständig indexierter Seiten vor einer Veröffentlichung durchgeführt werden.

Versionsspezifische Sicherung

Welche Dateien sollten trunk/branch-spezifisch sein?

- Seiten: data\pages
- Suchindex zu Seiten: data\index
- Medien: data\media
- Plugins in Testphase
- Neuerungen in Branch (inhaltliche Dateiunterschiede ⇒ Differenzierung ist via TortoiseSVN und update/commit-Workflow zu managen

Für eine versionsspezifische Sicherung sind auf jeden Fall `data\pages`, `data\media` relevant zzgl. der Inhalte der Verzeichnisse `conf` und `lib`. Die Konfigurationsdateien aus dem Verzeichnis `conf` sowie der Plugins `lib\plugins` beinhalten ggf. individualisierte Daten, die ebenfalls gesichert werden müssen und im Zweifelsfall versionsspezifisch (trunk oder branch) sind. Wird z. B. ein Plugin im Trunk installiert und liefert spezifische Codes im Seitentext, können diese Codes in der Branch-Variante nur dann sicher interpretiert werden, wenn das gleiche Plugin dort ebenfalls mit der gleichen Konfiguration eingerichtet ist! Daher ist die Liste der verwendeten Plugins sinnvoll, um den Überblick über die das DokuWiki ergänzenden Features zu behalten. Plugins sollten nach einer Testphase möglichst immer in beiden Versionen parallel installiert werden.

SVN-Workflow

Nur Branch-Bearbeitung

Bei sauberer Trennung und reinem Bearbeiten von iX-Wiki in der Branch-Ebene reduziert sich der SVN-Workflow auf:

1. **Banch-Commit** (ggf. mit Add neuer Strukturen) von sämtlichen Inhalten (Version 20.20.2\)
2. Prüfung auf **Merge-Konflikte** (s. ggf. hierzu erzeigte E-Mails vom SVN-System im Rahmen der Verarbeitung)
3. ggf. **Konfliktauflösung**

Wurden durch Umstrukturierung im iX-Wiki Seiten gelöscht, werden diese (und ggf. auch hierzu automatisch angelegte Index-Dateien) beim commit als `missing` angezeigt, da SVN die Löschabsicht nicht automatisch erkennt. Die zu löschenden Dateien sind im commit-Dialog manuell mit der `delete`-Anweisung zu belegen, sodass sie nicht durch SVN wieder angelegt werden, sondern auch dort offiziell gelöscht sind.

Ist im lokalen Work-Verzeichnis das iX-Wiki integriert, werden die Aktualisierungen dort durch das normale SVN-Update vorgenommen.

Trunk- und Branch-Bearbeitung

Wir vermeiden eine parallele Bearbeitung von iX-Wiki-Inhalten in Trunk- und Branchebenen. Sinnvoller ist es, eine iX-Wiki-Version bis zu einer Deadline zu pflegen und ab dann nur im Branch fortzusetzen. Sind dennoch iX-Wiki-Änderungen im Trunk erforderlich, hat sich gezeigt, dass es sinnvoll ist, nach Aktualisierungsprozessen im Trunk kurzfristig zu aktualisieren, um die Anzahl möglicher Konflikte überschaubar zu halten.

1. **Banch-Commit** (ggf. mit Add neuer Strukturen, immer vor dem Trunk-Commit auszuführen!)
   Dies kann in zwei Schritten erfolgen
   1. Commit neuer Seiten (z. B. Version20.20.2\data\pages)
   2. Commit von ‚non-pages'-Inhalten (Version 20.20.2\): data-Unterverzeichnisse attic, index, media, media_attic, media_meta, meta
2. **Trunk-Commit** (ggf. mit Add neuer Strukturen)
   Dies kann in zwei Schritten erfolgen:
   1. Commit neuer Seiten (z. B. Version20.21.3\data\pages)
   2. Commit von ‚non-pages'-Inhalten (Version 20.21.3\): data-Unterverzeichnisse attic, index,

media, media_attic, media_meta, meta.

3. Prüfung auf **Merge-Konflikte** (s. ggf. hierzu erzeigte E-Mails vom SVN-System im Rahmen der Verarbeitung)
4. ggf. **Konfliktauflösung**
5. **Branch-Update**
6. Prüfung auf **Merge-Konflikte** (s. ggf. hierzu erzeigte E-Mails vom SVN-System im Rahmen der Verarbeitung)
7. ggf. **Konfliktauflösung**

Zum Dateivergleich auf Dateiebene der txt-Dateien kann auch der **Total Commander** genutzt werden. Dieser bietet die Dateifunktion `Compare By Content…`. Sind die verglichenen Dateiversionen von Trunk / Branch einer Datei identisch, erhält man eine Hinweismeldung. Bei Unterschieden öffnet sich ein geteiltes Fenster mit den Möglichkeiten zu den jeweiligen Unterschieden zu springen und Dateiinhalte abzugleichen.

Ein Merge-Konflikt sollte letztlich mit SVN-Tools bereinigt werden. Für Wikidateien bedeutet das i. d. R. einen Vergleich von Textdateien. Dieser ist einfach zu realisieren. Der Vergleich von Worddateien gestaltet sich schwieriger, da hier nicht mit einem Editor zeilenweise verglichen werden kann. Ggf. kann hier über SVN das Konfliktdokument in Word mit Versionsvergleich bearbeitet werden. Beim anschließenden Commit sollte vom Redakteur in der Commitinfo dann auf den gelösten Mergekonflikt hingewiesen werden.

Die Bearbeitung der Konflikte ist in der Dokumentation von TortoiseSVN grundlegend beschrieben. Weiter unten einige Auszüge hieraus (zur SVN-Version 1.14, Kapitel 4.6):

TortoiseSVN-Eigenschaften (Beispiele)

TortoiseSVN-Eigenschaften TRUNK

Abrufbar über die SVN-Properties
Stand: 24.11.2020

- https://srventw01.crem-solutions.de/svn/ix_rep/trunk/ix_root_minus1/ix_dev/doku/DokuWiki

TortoiseSVN-Properites TRUNK
WC Revision: 158738
Depth: fully recursive

Property	Value	inherited from
svn.ignore	_dummy .htaccess dont-panic-if-you-see-this-in-your-logs-it-means-your-directory-permissions-are-correct.png dont-panic-if-you-see-this-in-your-logs-it-means-your-directory-permissions-are-correct.xcf vendor lib inc bin COPYING	
svn.ignore	Alter Stand Rest- Allgemeine Dokumente.zip	https://srventw01.crem-solutions.de/svn/ix_rep/trunk/ix_root_minus1/ix_dev/doku
svn.mergeinfo	/trunk/ix_root:121037-126502	https://srventw01.crem-solutions.de/svn/ix_rep/trunk/ix_root_minus1
svn.mergeinfo	/trunk/ix_root/ix_dev:59966-60349*,121037-126502,130826 /trunk/ixroot_minus1/ixplus/NCrem.BL/BusinessObjects/PlusSachkonten/SachkontenChangeHelper.cs:130825-130826	https://srventw01.crem-solutions.de/svn/ix_rep/trunk/ix_root_minus1/ix_dev
tsvn:logtemplate	Auslöser Inhaltliche Änderung Doku Testhinweise Betroffene Module Technische Änderung	https://srventw01.crem-solutions.de/svn/ix_rep/trunk/ix_root_minus1/ix_dev

TortoiseSVN-Eigenschaften BRANCH

Stand: 24.11.2020

- https://srventw01.crem-solutions.de/svn/ix_rep/trunk/ix_root_minus1/ix_dev/doku/DokuWiki

TortoiseSVN-Properites BRANCH
WC Revision: 161853
Depth: fully recursive

Property	Value	inherited from
svn.ignore	_dummy .htaccess dont-panic-if-you-see-this-in-your-logs-it-means-your-directory-permissions-are-correct.png dont-panic-if-you-see-this-in-your-logs-it-means-your-directory-permissions-are-correct.xcf vendor lib inc bin COPYING	
svn.ignore	Alter Stand Rest- Allgemeine Dokumente.zip	https://srventw01.crem-solutions.de/svn/ix_rep/branches/ix_root/ix_dev/doku
svn.mergeinfo	/trunk/ix_root:121037-126502	https://srventw01.crem-solutions.de/svn/ix_rep/branches/ix_root
svn.mergeinfo	/trunk/ix_root/ix_dev:59966-60349*,121037-126502,130826 /trunk/ixroot/ixplus/NCrem.BL/BusinessObjects/PlusSachkonten/SachkontenChangeHelper.cs:130825-130826	https://srventw01.crem-solutions.de/svn/ix_rep/branches/ix_root/ix_dev
tsvn:logtemplate	Auslöser Inhaltliche Änderung Doku Testhinweise Betroffene Module Technische Änderung	https://srventw01.crem-solutions.de/svn/ix_rep/branches/ix_root/ix_dev

Konflikte in TortoiseSVN auflösen

Bei Mergekonflikten in der Error-E-Mail auf `Antworten` klicken und in dem Mail-Editor dann mit `Strg + F` nach „conflict(s)“ suchen. Dort sind dann die Dateien benannt, welche konfliktbehaftet sind. Nachfolgende Infos wurde aus der SVN-Hilfe entnommen...

Ab und an werden Sie einen Konflikt erhalten, wenn Sie Ihre Arbeitskopie aktualisieren oder zu einer anderen URL wechseln. Es gibt zwei Arten von Konflikten:

- Dateikonflikte: Ein Dateikonflikt entsteht, wenn zwei (oder mehr) Entwickler dieselben Zeilen einer Datei geändert haben.
- Baumkonflikte: Ein Baumkonflikt entsteht, wenn ein Entwickler eine Datei oder einen Ordner umbenannt, verschoben oder gelöscht hat, den ein anderer Entwickler ebenfalls umbenannt, verschoben, gelöscht oder bearbeitet hat.

Dateikonflikte

Ein Konflikt tritt dann auf, wenn mehrere Personen die gleichen Zeilen in einer Datei verändert haben. Da Subversion nichts über Ihr Projekt weiß, überlässt es in solchen Fällen Ihnen, den Konflikt aufzulösen. Die sich in Konflikt befindlichen Bereiche sind in der Resolve-Ansicht folgendermaßen markiert:

```
<<<<<<< Dateiname
    Ihre Änderungen
=======
    Code aus dem Projektarchiv
```

```
>>>>>>> Revision
```

Außerdem werden für jede Datei in Konflikt drei weitere Dateien in Ihrem Verzeichnis erstellt:

filename.ext.mine

Dies ist die Datei, so wie Sie war, bevor Sie Ihre Arbeitskopie aktualisierten. Das heißt, es ist Ihre eigene Originaldatei inklusive der Änderungen, die Sie selbst vorgenommen haben.

filename.ext.rOLDREV

Dies ist die Datei, wie Sie ursprünglich war, ohne jegliche Änderungen, auch ohne die Änderungen, die Sie selbst an der Datei vorgenommen haben.

filename.ext.rNEWREV

Dies ist die Datei, wie sie im Projektarchiv gerade aktuell ist, d. h., diese Datei hat die Änderungen von den anderen Mitarbeitern bereits integriert, jedoch noch nicht die Ihren.

Sie können entweder einen externen Konflikteditor per `TortoiseSVN → Konflikt bearbeiten` aufrufen oder Sie können den Konflikt mit einem beliebigen Texteditor manuell auflösen. Sie müssen entscheiden, wie der Code aussehen soll, die notwendigen Änderungen vornehmen und die Datei speichern. Mit einem Konflikteditor wie TortoiseMerge oder einem der anderen beliebten Programme ist das im allgemeinen einfacher, da diese die betroffenen Dateien normalerweise in einer Drei-Fenster-Sicht anzeigen und Sie sich keine Gedanken über die Konfliktmarken machen müssen. Wenn Sie einen Texteditor verwenden, müssen sie manuell nach Zeilen suchen, die mit «««< beginnen.

Anschließend müssen Sie Subversion noch mitteilen, dass Sie den Konflikt aufgelöst haben. Dies geschieht mit dem Befehl `TortoiseSVN → Konflikt aufgelöst`. Bitte beachten Sie, dass dieser Befehl nicht den Konflikt selbst löst, sondern nur Subversion mitteilt, dass Sie selbst den Konflikt bereits gelöst haben. Der Befehl macht nichts weiter als die drei zusätzlich erstellten Dateien `filename.ext.mine` und `filename.ext.r*` zu löschen, damit sie Ihre Änderungen in das Projektarchiv übertragen können.

Wenn ein Konflikt zwischen Binärdaten besteht, versucht Subversion nicht, die Daten selbst zusammenzuführen. Die lokale Datei bleibt unverändert (exakt so, wie sie Ihrer letzten Änderung entspricht) und Sie erhalten `Dateiname.ext.r*` Dateien. Wenn Sie Ihre eigenen Änderungen verwerfen wollen, tun Sie das mit dem Befehl `Rückgängig`. Wenn Sie Ihre Version beibehalten und die Version im Projektarchiv überschreiben wollen, verwenden Sie den Befehl `Konflikt aufgelöst` und übertragen anschließend die Daten ins Projektarchiv.

Sie können den Befehl `Konflikt aufgelöst` für mehrere Dateien verwenden, indem Sie den übergeordneten Ordner markieren und `TortoiseSVN → Konflikt aufgelöst…` aus dem Kontextmenü wählen. Dies öffnet einen Auswahldialog, in dem alle konfliktbehafteten Dateien aufgelistet sind. Wählen Sie die Dateien, die Sie als aufgelöst markieren wollen.

Eigenschaftskonflikte

Ein Eigenschaftskonflikt entsteht dann, wenn zwei oder mehr Entwickler dieselbe SVN-Eigenschaft

verändert haben. Wie bei Dateikonflikten können nur Entwickler solche Konflikte auflösen.

Wenn eine der Änderungen die andere überschreiben soll, wählen Sie entweder `Mit der lokalen Eigenschaft auflösen` oder `Mit der Eigenschaft aus dem Projektarchiv auflösen`. Wenn die Änderungen zusammengeführt werden müssen, wählen Sie `Revisionseigenschaft bearbeiten`, tragen dort den gewünschten Wert ein und markieren den Konflikt als aufgelöst.

Baumkonflikte

Ein Baumkonflikt entsteht, wenn ein Entwickler eine Datei oder einen Ordner umbenannt, verschoben oder gelöscht hat, den ein anderer Entwickler ebenfalls umbenannt, verschoben, gelöscht oder bearbeitet hat. Es gibt verschiedene Ursachen für Baumkonflikte und alle erfordern unterschiedliche Vorgehensweisen, um den Konflikt aufzulösen.

Wenn eine Datei in Subversion lokal gelöscht wird, wird sie auch aus dem lokalen Dateisystem gelöscht. Das bedeutet, dass kein überlagertes Symbol angezeigt werden kann, wenn sie Teil eines Baumkonfliktes ist und dass Sie den Konflikt nicht mit Hilfe eines Rechtsklicks auflösen können. Verwenden Sie stattdessen den `Auf Änderungen prüfen` Dialog, um den `Konflikt bearbeiten` zu können.

TortoiseSVN kann dabei helfen, Änderungen zusammenzuführen, aber es kann zusätzliche Arbeit erforderlich sein, um die Konflikte aufzulösen. Bedenken Sie, dass nach eine Aktualisierung die BASE der Arbeitskopie dem Inhalt des Projektarchivs entspricht. Wenn Sie eine Änderung nach dem Aktualisieren rückgängig machen, wird das Objekt in den Status des Projektarchivs zurückversetzt und nicht in den Zustand, in dem Sie begonnen haben, Ihre eigenen Änderungen durchzuführen.
</nodisp>

Baumkonflikt: Lokal gelöscht, eingehende Änderung beim Aktualisieren

1. Entwickler A verändert die Datei Foo.c und überträgt die Änderung ins Projektarchiv.
2. Entwickler B benennt gleichzeitig die Datei `Foo.c` in seiner Arbeitskopie in `Bar.c` um oder löscht `Foo.c` bzw. den Elternordner.

Eine Aktualisierung der Arbeitskopie von Entwickler B resultiert in einem Baumkonflikt:

• Die Datei `Foo.c` wurde aus der Arbeitskopie gelöscht, ist aber gleichzeitig als Baumkonflikt markiert.

• Wenn ein Konflikt nicht von einem Löschen, sondern von einem Umbenennen herrührt, ist die `DateiBar.c` als hinzugefügt markiert, enthält aber nicht die Änderungen von Entwickler A.

Entwickler B muss sich nun entscheiden, ob er die Änderungen von Entwickler A übernehmen möchte. Im Fall des Umbenennens kann er die Änderungen an `Foo.c` in `Bar.c` zusammenführen. Für einfache Löschungen kann er das Objekt mit den Änderungen von Entwickler A beibehalten und das Löschen verwerfen. Oder er kann, indem er den Konflikt ohne weitere Aktionen als aufgelöst markiert, die Änderungen von Entwickler A verwerfen.

Der Konfliktbearbeitungsdialog ermöglicht es, Änderungen zusammenzuführen, wenn das Original der umbenannten Datei `Bar.c` gefunden werden kann. Wenn es mehrere in Frage kommende Dateien gibt, wird eine Schaltfläche für jede Datei angezeigt, was die Auswahl der zutreffenden Datei erlaubt.

Baumkonflikt: Lokal geändert, eingehendes Löschen beim Aktualisieren

1. Entwickler A benennt die Datei `Foo.c` in `Bar.c` um und überträgt die Änderung ins Projektarchiv.
2. Entwickler B verschiebt die Datei `Foo.c` in seiner Arbeitskopie.

Oder im Fall eines verschobenen Ordners …

1. Entwickler A benennt den Elternordner `FooOrdner` in `BarFolder` um und überträgt die Änderung ins Projektarchiv.
2. Entwickler B verschiebt die Datei `Foo.c` in seiner Arbeitskopie.

Eine Aktualisierung der Arbeitskopie von Entwickler B führt zu einem Baumkonflikt. Für einen einfachen Dateikonflikt:

- Die Datei `Bar.c` wird als normale Datei zur Arbeitskopie hinzugefügt.
- Die Datei `Foo.c` ist als *hinzugefügt mit Historie* markiert und hat einen Baumkonflikt.

Für einen Ordnerkonflikt:

- `BarOrdner` wird als normaler Ordner zur Arbeitskopie hinzugefügt.
- Der Ordner `FooOrdner` ist als *hinzugefügt mit Historie* markiert und hat einen Baumkonflikt.

Die Datei `Foo.c` ist als *verändert* markiert.

Entwickler B muss nun entscheiden, ob er die Reorganisation durch Entwickler A übernehmen will und seine Änderungen in der entsprechenden Datei in der neuen Struktur zusammenführt oder ob er einfach die Änderungen von A rückgängig macht und die lokale Datei beibehält.

Um ihre lokalen Änderungen mit der Umstrukturierung zusammenzuführen, muss Entwickler B zuerst herausfinden, zu welchem Dateinamen die Konfliktdatei `Foo.c` im Projektarchiv umbenannt/verschoben wurde. Dies kann unter Verwendung des Log-Dialogs geschehen. Dann verwendet sie die Schaltfläche, welche die zutreffende Quelldatei zeigt, um den Konflikt aufzulösen.

Wenn Entwickler B entscheidet, dass die Änderungen von A falsch waren, muss er die `Als gelöst markieren`-Schaltfläche im Konfliktbearbeitungsdialog verwenden. Dies markiert den Konflikt der/des Datei/Ordners als aufgelöst, aber die Änderungen des Entwicklers A müssen von Hand entfernt werden. Auch hier hilft der Log-Dialog dabei, die Verschiebungen nachzuverfolgen.

Baumkonflikt: Lokal gelöscht, eingehende Löschung beim Aktualisieren

1. Entwickler A benennt die Datei `Foo.c` in `Bar.c` um und überträgt die Änderung ins Projektarchiv.

2. Entwickler B benennt `Foo.c` in `Bix.c` um.

Eine Aktualisierung der Arbeitskopie von Entwickler B resultiert in einem Baumkonflikt:

- Die Datei `Bix.c` ist als *hinzugefüggt mit Historie* markiert.
- Die Datei `Bar.c` ist als *normal* markiert.
- Die Datei `Foo.c` ist als *gelöscht* markiert und hat einen Baumkonflikt.

Um diesen Konflikt aufzulösen, muss Entwickler B zunächst herausfinden, wie die konfliktbehaftete Datei `Foo.c` im Projektarchiv umbenannt wurde. Dies kann mit Hilfe des Log-Dialogs geschehen.

Danach muss sich Entwickler B entscheiden, welchen neuen Dateinamen von `Foo.c` er übernehmen möchte, den eigenen oder den von Entwickler A vergebenen Namen.

Nachdem Entwickler B den Konflikt manuell aufgelöst hat, muss der Baumkonflikt mit der Schaltfläche im Konflikteditor als *aufgelöst* markiert werden.

Baumkonflikt: Lokal fehlend, eingehende Änderung beim Aktualisieren

1. Entwickler A verändert im Stamm die Datei `Foo.c` um und überträgt die Änderung ins Projektarchiv.
2. Entwickler B, der auf einem Zweig arbeitet, benennt `Foo.c` in `Bar.c` um und überträgt die Änderung ins Projektarchiv.

Das Zusammenführen der Änderungen von Entwickler A im Stamm mit der Arbeitskopie von Entwickler B führt zu einem Baumkonflikt:

- Die Datei `Bar.c` befindet sich bereits mit dem Status *normal* in der Arbeitskopie.
- Die Datei `Foo.c` ist als *fehlend* markiert und hat einen Baumkonflikt.

Um diesen Konflikt aufzulösen, muss Entwickler B den Dateikonflikt im Konflikteditor als *aufgelöst* markieren, wodurch er aus der Konfliktliste entfernt wird. Danach muss er entscheiden, ob er die fehlende Datei `Foo.c` aus dem Projektarchiv in die Arbeitskopie kopiert, die Änderungen von Entwickler A an `Foo.c` in die umbenannte Datei `Bar.c` überträgt oder die Änderungen ignoriert, indem er den Konflikt als aufgelöst markiert und nichts weiter unternimmt.

Beachten Sie, dass, wenn Sie die fehlende Datei aus dem Projektarchiv kopieren und danach den Konflikt als aufgelöst markieren, ihre Kopie wieder entfernt wird. Sie müssen den Konflikt erst auflösen und danach die Datei kopieren.

Baumkonflikt: Lokal bearbeitet, eingehende Löschung beim Zusammenführen

1. Entwickler A benennt im Stamm die Datei `Foo.c` in `Bar.c` um und überträgt die Änderung ins Projektarchiv.
2. Entwickler B, der auf einem Zweig arbeitet, modifiziert `Foo.c` und überträgt die Änderung ins Projektarchiv.

1. Entwickler A benennt im Stamm den Ordner `FooOrdner` in `BarOrdner` um und überträgt die Änderung ins Projektarchiv.
2. Entwickler B, der auf einem Zweig arbeitet, verändert `Foo.c` in seiner Arbeitskopie.

Das Zusammenführen der Änderungen von Entwickler A im Stamm mit der Arbeitskopie von Entwickler B führt zu einem Baumkonflikt:

- Die Datei `Bar.c` ist als *hinzugefügt* markiert.
- Die Datei `Foo.c` ist als *verändert* markiert und hat einen Baumkonflikt.

Entwickler B muss nun entscheiden, ob er die Reorganisation durch Entwickler A übernehmen will und seine Änderungen in der entsprechenden Datei in der neuen Struktur zusammenführt oder ob er einfach die Änderungen von A rückgängig macht und die lokale Datei beibehält.

Um ihre lokale Änderungen mit der Umstrukturierung zusammenführen zu können, muss Entwickler B zuerst herausfinden, in welchen Dateinamen die Konfliktdatei `Foo.c` im Projektarchiv umbenannt/verschoben wurde. Dies kann unter Verwendung des Log-Dialogs für die Zusammenführungsquelle erreicht werden. Der Konflikteditor zeigt nur das Protokoll der Arbeitskopie, da nicht bekannt ist, welcher Pfad bei der Zusammenführung verwendet wurde, sodass dieser selbst gesucht werden muss. Die Änderungen müssen dann per Hand zusammengeführt werden, da es gegenwärtig keine Möglichkeit gibt, diesen Prozess zu automatisieren oder auch nur zu vereinfachen. Sobald die Änderungen übertragen wurden, werden die Konfliktpfade überflüssig und können gelöscht werden.

Wenn Entwickler B entscheidet, dass die Änderungen von A falsch waren, muss er die *Als gelöst markieren*-Schaltfläche im Konfliktbearbeitungsdialog verwenden. Dies markiert den Konflikt der/des Datei/Ordners als aufgelöst, aber die Änderungen des Entwicklers A müssen von Hand entfernt werden. Wieder hilft hier der Log-Dialog der Quelle, die Verschiebungen nachzuverfolgen.

Baumkonflikt: Lokal gelöscht, eingehende Löschung beim Zusammenführen

1. Entwickler A benennt im Stamm die Datei `Foo.c` in `Bar.c` um und überträgt die Änderung ins Projektarchiv.
2. Entwickler B, der auf einem Zweig arbeitet, benennt `Foo.c` in `Bix.c` um und überträgt die Änderung ins Projektarchiv.

Das Zusammenführen der Änderungen von Entwickler A im Stamm mit der Arbeitskopie von Entwickler B führt zu einem Baumkonflikt:

- Die Datei `Bix.c` ist als *normal* (unverändert) markiert.
- Die Datei `Bar.c` ist als *hinzugefügt mit Historie* markiert.
- Die Datei `Foo.c` ist als *fehlend* markiert und hat einen Baumkonflikt.

Um diesen Konflikt aufzulösen, muss Entwickler B zunächst herausfinden, wie die konfliktbehaftete Datei `Foo.c` im Projektarchiv umbenannt wurde. Dies kann mit Hilfe des Log-Dialogs für die Quelle geschehen.

Danach muss sich Entwickler B entscheiden, welchen neuen Dateinamen von `Foo.c` er übernehmen möchte, den eigenen oder den von Entwickler A vergebenen Namen.

Nachdem Entwickler B den Konflikt manuell aufgelöst hat, muss der Baumkonflikt mit der Schaltfläche im Konflikteditor als aufgelöst markiert werden.

Andere Baumkonflikte

Es gibt weitere Fälle, die einfach deshalb als Baumkonflikte gekennzeichnet werden, weil der Konflikt einen Ordner anstelle einer Datei betrifft. Wenn Sie z. B. einen Ordner des gleichen Namens sowohl zu trunk als auch zu branch hinzufügen und versuchen, diese Änderungen zusammenzuführen, erhalten Sie einen Baumkonflikt. Wenn Sie den Zielordner behalten wollen, markieren Sie den Konflikt einfach als aufgelöst. Wenn Sie den Quellordner behalten wollen, müssen Sie zunächst im Projektarchiv den Zielordner löschen und das Zusammenführen neu starten. Kompliziertere Situationen müssen Sie manuell auflösen.

lokale Merge_Unterverzeichnisse

Neben den Verzeichnissen, in welchen Dateivarianten zu finden sind, welch ggf. auch als Bestandteil eine Pakets (Patch oder Servicepack) an Kunden gehen, gibt es im lokalen work-Verzeichnis auch noch Hilfsverezichnisse für die Merge-Prozesse. Die Namen dieser Hilfsverzeichnisse beginnen mit `merge_` und werden dann mit Quelle und Zielebene ergänzt. Daher gibt es parallel zu den lokalen Versionsverzeichnissen

```
branch
trunk
trunk_minus1
trunk_minus2
trunk_minus3
trunk_minus4
```

die lokalen Hilfsverzeichnisse

```
merge_t_to_b
merge_t1_to_t
merge_t2_to_t1
merge_t2_to_t1
merge_t3_to_t2
merge_t4_to_t3
```

Diese Hilfsverzeichnisse müssen i. d. R. nicht separat per SVN-update oder SVN-commit bearbeitet werden. Sie werden durch interne Prozesse beim Commit von einem trunk-Verzeichnis genutzt, um die Inhalte pro Version in das übergeordnete Dateiverzeichnis zu transferieren.

Dennoch können hier Versionskonflikte auftreten, wenn Dokumente in unteren Ebenen angepasst werden und in einer Zwischenebene der merge-Prozess scheitert. Wenn SVN-resolve mit Versionsvvergleich hier nicht klappt, ist der Vorschlag zur Lösung wie folgt (Beispiel: Konflikt im merge_t_2_b):

1. Datei im Mergeverzeichnis (merge_t_b_mit SVN-resolve using mine zurücksetzen (das bedeutet, dass die Änderungen von t nicht übernommen werden).
2. SVN-commit mit dem Merge-Kommentar (Fehlerhinweis für Historie).
3. SVN-update auf Branch.
4. Betroffene Datei im Zielordner (hier: lokale Branch-Verzeichnis) öffnen und inhaltlich aktualisieren mit den Anpassungen aus der Quelldatei (hier trunk) trunk gemachten Änderungen.
5. SVN-commit auch mit dem Merge-Kommentar (hier: im lokalen Branch-Verzeichnis).



Dies mit einer inhaltlichen Komplettkopie nur sinnvoll, wenn in der übergeordneten Datei keine sonstigen versionsspezifischen Inhalte existieren (identische Inhalte in trunk und branch). Ansonsten ist die selektive Ergänzung der Inhalte erforderlich!