Problema:
Workflow per lo sviluppo su più piattaforme contemporaneamente, tipo java e android o java ed eclipse.

In questo scenario, lo sviluppatore deve rigenerare ad ogni modifica il file tuprolog.jar e sostituirlo nel progetto in cui lo deve provare.
Ad esempio, Mercurio, doveva scrivere le modifiche ai class loader nella versione Java, rigenerare tuprolog.jar e inserirlo nella cartella libs di android così da poterlo provare sull'emulatore.
Qui il problema sta nel fatto che, c'è il rischio che qualcuno committi il tuprolog.jar che genera lui ma quel commit non riguarda solo il proprio branch ma anche gli altri progetti perché di fatto si sta modificando il tuprolog.jar referenziato anche dagli altri.
In pratica, il tuprolog.jar scaricato tramite external è un link a quello presente nei tags, quindi se io lo modifico e committo, non sto committando sul mio branch ma nei tags.

Considerazione generali
- Il punto infatti è che qualunque soluzione è buona quando ci si prende confidenza e la si usa semi-quotidianamente, per cui anche il dover avere un file da rinominare o un task ant da far andare o delle properties da cambiare qui piuttosto che là è gestibile perché dopo un paio di giorni diventa abitudine; con tuprolog però non funziona così perché non abbiamo una vera squadra fissa di sviluppo che faccia solo quello o abbia quello come main task del suo operare quotidiano.
- l'utilizzo troppo massiccio di Ant espone a una dipendenza perenna verso dei task anche per le operazioni più normali, il che equivale alla quasi certa garanzia che al primo commit o altra operazione fatta "per abitudine SVN" in modo diretto e fuori task Ant si vada verso stati di inconsistenza variegati e difficili da prevedere.
- se decidessimo di cambiare servizio di hosting questo potrebbe aiutare: http://www.svnhostingcomparison.com/

~~Idea 1:~~
definire le autorizzazioni sui vari path.

PRO:
- semplice e pulita (soluzione più ovvia)

CONTRO:
- bisogna cambiare servizio di hosting

Stato: Scartata perché google code ha disabilitato questa funzionalità.

~~Idea 2:~~
pre-commit hooks.

~~Scartata perché sono script che girano lato cliente e quindi non abbiamo la certezza che gli sviluppatori li attivino correttamente.~~
In realtà gli script eseguono sul server ma google code non offre la possibilità di installarli.

PRO:
- si ha molto controllo perché in quegli script potremmo farci di tutto

CONTRO:
- ~~gli sviluppatori devono configurare il loro client correttamente altrimenti gli script sono inutili~~
- ~~alcuni tool non hanno la possibilità di configurare tali script (Subclipse)~~
- bisogna cambiare servizio di hosting

Stato: Scartata perché google non permette di configurare tali script.

Idea 3:
specificando il numero di revisione al quale si fa riferimento nell'external, se si prova a committare quel file ci si becca un errore e il tutto viene bloccato.
Un esempio:
- revision 1: tagghiamo la 2.9 e quindi generiamo tuprolog.jar in tags/REL-2.9.0
- revisione 2: modifichiamo l'external in android e mettiamo
    svn:externals
    -r1 ^/2p/tags/REL-2.9.0/build/archives/tuprolog.jar libs/tuprolog.jar
- lo stesso per gli altri progetti

Quel "-r1" che mettiamo nell'externals è quello che ci protegge, se provo a committare tuprolog.jar becco un errore perché io sono ad una revisione "più avanti" e non posso modificare qualcosa che è ad una revisione "più indietro".

PRO:
- si ha la certezza che il file tuprolog.jar non venga modificato inavvertitamente
- relativamente semplice

CONTRO:
- c'è un pelo di lavoro da fare in più quando si aggiornano gli externals. Prima bisognava solo cambiare un numero (da 2.8 a 2.9 per esempio), adesso bisogna annotarsi il numero di revisione in cui è stato fatto il tag e modificare anche quello.
- se c'è la necessità di dover cambiare il tuprolog.jar che si trova nei tags, senza avanzare il numero di versione (in passato è capitato per risolvere dei bug “sottobanco”), si è obbligati a modificare i numeri di revisione in tutti gli externals. Altrimenti loro continuerebbero a referenziare il vecchio jar.

Stato: testata

Idea 4:
Avere il jar direttamente nei branch o nel trunk quindi non si utilizzerebbero gli externals.

Mettiamo il jar in una cartella nel trunk dei progetti che lo richiedono (android, eclipse, .net).
A questo punto i progetti sono auto-contenuti, hanno già tutto quello che serve per sviluppare e così sono anche i branch che derivano da quel trunk.
In questo caso gli sviluppatori possono sovrascrivere il file tuprolog.jar, ed eventualmente anche committarlo, tanto quella modifica sarà locale al loro branch e non avrà effetti sugli altri progetti.

PRO:
- risolviamo il problema e non dobbiamo neanche dire niente di particolare agli sviluppatori, possono fare quello che vogliono perché i branch sono veramente isolati.
- togliendo tutti gli externals, l'update di tutto il repo dovrebbe velocizzarsi.

CONTRO:
- ci sarebbero molti jar in giro nel repo mentre con gli externals sono solo nei tags e poi vengono referenziati;
- non si capirebbe da dove viene quel jar, cioè da quale versione è stato generato, mentre guardando gli externals si capisce subito la versione e volendo si ha ancora il codice di quella versione proprio nel tag.
- bisogna garantire che ad ogni commit sul trunk (che abbia effetto sul motore) sia rigenerato e ricaricato sul repo il tuprolog.jar. Google code offre solo i post-commit hooks che potrebbero essere utilizzati in questo caso.
- nel caso in cui si rilasciasse una nuova versione di tuprolog quando ci sono attivi dei branch (di solito raro ma possibile) sarebbe necessario rigenerare il tuprolog.jar e andarlo a copiare nei vari branch. Una gestione simile era necessaria anche con gli externals, solo che in quel caso si modificava un numero nella proprietà e basta.

Stato: non testata

Idea 5:
Usare sempre gli externals ma copiare il file nella working copy con un altro nome, ad esempio tuprolog_ext.jar.
Poi nella guida scriviamo che per poter utilizzare il progetto è necessario copiare e incollare questo file con il nome tuprolog.jar (oppure rinominarlo). In entrambi i casi il file tuprolog.jar verrà visto come un nuovo file, non linkato a quello nei tags e dato che abbiamo messo una property svn:ignore nella cartella libs, anche se si committa quel file non sarà caricato sul server.

Questa soluzione non risolve completamente il problema ma lo rende solo un po' più difficile da manifestarsi. Non lo elimina perché se qualcuno genera il nuovo tuprolog.jar, poi lo rinomina in tuprolog_ext.jar e lo copia dentro libs siamo punto e a capo, se committa, commuta sui tags.

A favore di questa idea c'è il fatto che i progetti eclipse referenziano tuprolog.jar

quindi appena si fa il checkout si ha un errore poiché la libreria non viene trovata, quindi dovrebbe venire in mente che quel file va almeno rinominato. (Sperando che non venga in mente di modificare il reference in Eclipse :)).

PRO:
- semplice.

CONTRO:
- non ci da la sicurezza piena, volendo essere “perfidi” ce la si fa lo stesso a fare casini.

Stato: testata

Idea 6:
Togliere tutti gli externals e scrivere un target Ant che faccia l’export di tuprolog.jar dentro la cartella libs.
Siccome il file viene esportato non ha un link con il repo quindi posso fare quello che voglio.
In pratica è come se avessimo gli externals perché nello script Ant ci sarà scritto quale versione vogliamo copiare (2.8, 2.9, ecc.) ma è come se fossero “manuali” perché siamo noi a dover lanciare il target per tirare giù il jar e questo ci “scollega” dal repo.

PRO:
- gestione molto simile a quella necessaria per gli externals, e forse più semplice, perché il numero di versione sarà nello script Ant. Quindi prima modificavamo le svn properties adesso modifichiamo build.xml.
- sappiamo da dove viene il jar perché basta guardare lo script ant (prima guadavamo le svn properties quindi è la stessa cosa)
- possiamo cambiare il jar sottobanco come facevamo con gli external perché l’export non viene fatto da una revisione particolare ma dalla HEAD, cioè dall’ultima revisione presente sul repo.
- togliendo tutti gli externals, l’update di tutto il repo dovrebbe velocizzarsi.

CONTRO:
- gli sviluppatori devono lanciare il target ant per avere il progetto configurato e funzionante

Stato: non testata

~~Idea 7:~~
Utilizzare i lock.
Dopo aver targato una nuova versione si acquisisce un lock sul file tuprolog.jar, a questo punto chi prova a committarlo si becca un errore perché non è l’utente che ha acquisito il lock e perché non possiede il token che viene generato durante l’operazione di locking (che invece il proprietario del lock possiede nella sua working copy).

PRO:
- semplice da realizzare, si modificherebbe il task ant che tagga per fargli acquisire anche il lock
- si continuano ad utilizzare gli externals
- il file non rimane lockato per sempre e il proprietario del lock non è l'unico che può sbloccarlo. Infatti è possibile forzare l'operazione di unlock anche se non si è il proprietario: `svn unlock --force http://svn.example.com/repos/project/raisin.jpg`
- possiamo modificare il file sottobanco: togliamo il lock, sostituiamo il file, riacquistiamo il lock.

CONTRO:
- c'è un po' di gestione se dobbiamo sostituire il file perché bisogna togliere il lock e poi riacquisirlo.
- volendo uno studente potrebbe togliere il lock e committare lo stesso il file.

Stato: Testata e scartata perché google code non supporta il locking, bisognerebbe cambiare servizio di hosting (https://code.google.com/p/support/issues/detail?id=1349).

~~Idea 8:~~
Utilizzo della property svn:needs-lock.
Mettendo questa proprietà su tuprolog.jar quando viene fatto il checkout svn cerca di rendere il file read-only utilizzando i permessi del filesystem. In questo modo chi tenta di modificare il file non può farlo ed è necessario che acquisisca il lock su quel file. Solo a quel punto infatti svn rende il file read/write.
Questo meccanismo viene di solito utilizzato insieme al locking per evitare che qualcuno modifichi un file nella sua working copy mentre il lock su quel file e acquisito da un altro utente e quindi il primo non potrebbe poi committare. Con questa property ci si assicura che sia possible modificare il file solo se effettivamente si ha poi la possibilità di committare.

PRO:
- semplice da realizzare
- si continuano ad utilizzare gli externals
- possiamo modificare il file sottobanco

CONTRO:
- se sostituiamo il file bisogna ricordarsi di rimettere la property altrimenti salta tutto
- volendo uno studente potrebbe acquisire il lock e committare lo stesso il file

Stato: testata e scartata: se provo a sostituire il file tuprolog.jar nella mia working copy il client svn mi chiede di acquisire il lock e se acquisissi il lock potrei committare il file.
Siccome google code non supporta il locking (vedi idea 7), il client svn non mi fa sostituire il file e quindi non posso utilizzare il nuovo jar.
Però, se sostituisco il file direttamente dal Sistema Operativo, cioè senza passare da eclipse, lo posso utilizzare ma lo posso anche committare, quindi non sto proteggendo niente.

~~Idea 9:~~
Configurare opportunamente i progetti.
Pensandoci, il problema si manifesta solo quando un developer deve lavorare contemporaneamente su due progetti (se lavoro solo android non cambierò mai il tuprolog.jar e quindi il problema non esiste) quindi avrà entrambi i progetti configurati nell'ide (nello stesso workspace con eclipse), quindi l'idea è quella di linkare direttamente i due progetti nell'ide. In questo modo non è più necessario rigenerare tuprolog.jar e copiarlo perché c'è reference diretto tra i sorgenti.
Più in pratica:
due progetti, uno per java e uno per android linkati ai rispettivi branch. Nel progetto android si toglie la dipendenza da tuprolog.jar e si aggiunge un link al progetto java. In questo modo le modifiche vengono commitatte direttamente sui relativi branch e non c'è più il rischio di committare turpolog.jar nei tag perché non si usa più, si referenzia direttamente il sorgente.

PRO:
- noi non dovremmo fare niente, rimangono gli externals

CONTRO:
- facciamo affidamento sugli sviluppatori, chi non segue la nostra guida potrebbe comunque committare il file
- quando si reintegra bisogna fare attenzione a non fare il merge delle properties del progetto altrimenti i singoli progetti (android, eclipse) non sarebbero più autocontenuti. Ossia chi sviluppa solo per una piattaforma continua ad usare turpolog.jar e non deve linkare un altro progetto.
- forse il problema rimarrebbe per chi sviluppa contemporaneamente per java e .net, da vedere bene.

Stato:
- testata per java e android: tutto ok e facile da impostare
- testata per java ed eclipse: ok ma
    - il progetto java deve essere "convertito in plugin" (non so bene questo cosa voglia dire ma serve ad eclipse per poterlo referenziare in un altro progetto che implementa un plugin vero). Questo può essere fatto una volta per tutte e poi gli sviluppatori non dovrebbero più farlo.
    - il progetto del plugin è un po' complesso da impostare per utilizzare il progetto java e non il jar, niente di clamoroso ma più difficile di quello android
- testata per java e .net: non funziona perché non c'è un modo di linkare un progetto eclipse ad uno visual studio. Chi sviluppa la OOLibrary usa visual studio e il progetto dipende da tuprolog.dll che viene generato con IKVM quindi non c'è modo di linkare la OOLib al progetto del motore in java.

Idea 10:
Tenere i branch degli studenti (o eventualmente tutti i branch) su un repository diverso, diciamo repo2.
Siccome non si può fare un branch diretto tra due repo diversi, è necessario creare il branch nel repo1 e poi importarlo nel repo2.

A questo punto si darebbe ai developer il permesso di scrittura solo sul repo2 e visto che il file tuprolog.jar risiederà sempre sul repo1 (verrà copiato nella working copy tramite externals) si dovrebbero scongiurare i commit erronei, dato che i developer non hanno i permessi di commit sul repo1.

A questo punto, quando lo studente ha finito il lavoro sul suo branch servirebbe un merge dal repo2 al repo1 così che sia possibile reintegrare il lavoro nel trunk (che si trova nel repo1). Purtroppo però il merge diretto tra due repo diversi non può essere fatto (http://svn.haxx.se/users/archive-2006-04/0285.shtml) ma si può fare passando per una working copy:
1) checkout del branch dal repo1
2) merge di quella working copy con il branch sul repo2
3) commit della working copy sul repo1
4) elimina branch sul repo2

Inoltre credo che così facendo si perda la storia di tutti i commit dello studente sul repo2, ossia, nel repo1, tutto il suo lavoro sarà visto come un unico commit, mentre sul repo2 si avrebbero i vari commit intermedi. Questo potrebbe anche non essere un punto a sfavore in realtà.

PRO:
- gli externals dovrebbero funzionare ugualmente (non l'ho testato ma ne sono abbastanza convinto, mi sembra di averlo letto nella guida di svn che è una cosa possibile).
- non dando accesso agli studenti al repository principale dovremmo essere al sicuro da un commit accidentale di tuprolog.jar, si beccherebbero un errore perché non hanno i premessi di fare commit.

CONTRO:
- non è possibile fare un branch (svn copy) direttamente da un repo all'altro, si ottiene questo errore "svn: E200007: Source and destination URLs appear not to point to the same repository.". Bisogna quindi esportare il branch dal repo1 ed importarlo nel repo2.
- anche il merge non può essere fatto direttamente tra repo diversi bisogna passare tramite una working copy (vedere sopra).
- in pratica bisogna fare un passo in più sia per creare un branch che per reintegrarlo nel trunk.

Stato: Non testata completamente.