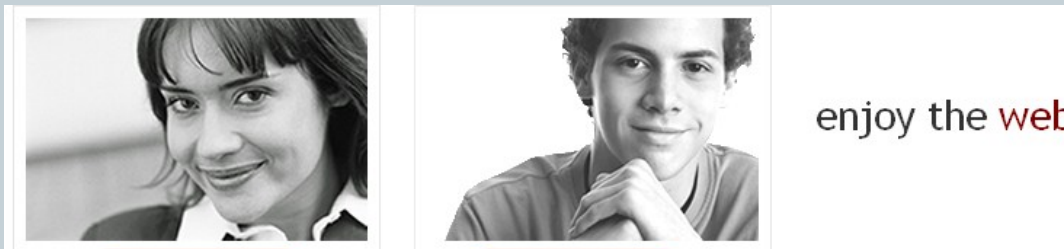


Small Software Factories

Sviluppare software in piccole realtà per grandi clienti

Software Configuration Management



Software Configuration Management

Concetti base

- (da Wikipedia) Il configuration management ha lo scopo di controllare e gestire le attività (sia documentali sia implementative) che portano alla produzione di software.
- Gestisce gli input/output direttamente o indirettamente legati alla costruzione di un prodotto software.
- Correla tra di loro i vari oggetti archiviati relativamente ad un prodotto software, mantenendo allo stesso tempo traccia delle varie versioni degli oggetti e della loro applicabilità.
- La gestione è di tipo formale ovvero nel processo vengono seguite procedure definite in precedenza tramite opportuni moduli di gestione.

Software Configuration Management

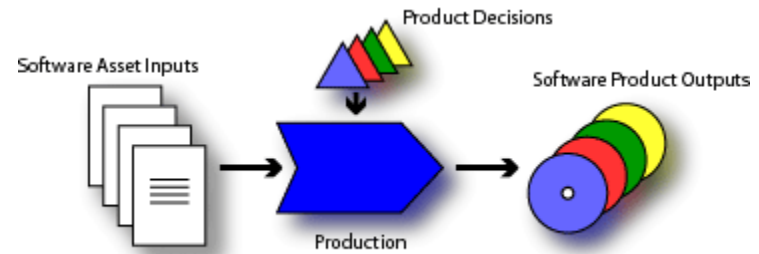
Perché introdurre la metodologia

- Gestione strutturata dei “software assets” aziendali.
- Salvaguardia degli oggetti da perdite e/o modifiche accidentali.
- Abilitazione dello sviluppo collaborativo di software
- Abilitazione dello sviluppo collaborativo in organizzazioni territorialmente distribuite.
- Raccolta dei semilavorati nei vari stadi intermedi del processo di sviluppo software.
- Facilitare e supportare i procedimenti di integrazione del software.
- Tracciamento delle attività di implementazione.
- Supporto ai processi di produzione automatizzata e ripetitiva.
- Facilitare e supportare le attività personalizzazione dei prodotti (varianti per cliente, mercato, piattaforma, ...)

Software Configuration Management

Gestione delle varianti di prodotto

- Ovvietà: i costi di manutenzione di un prodotto sono direttamente proporzionali al numero di versioni supportate.
- Le versioni si moltiplicano in funzione di:
 - Stato di rilascio (test, pre-produzione, produzione)
 - Evoluzione naturale del prodotto con l'aggiunta di nuove funzionalità e successivi rilasci.
 - Personalizzazioni per:
 - Cliente
 - Piattaforma
 - Sistema operativo
 -
- Obiettivo primario: minimizzare il numero di versioni supportate.
- Quando è necessario mantenere più versioni:
 - La sola gestione delle versioni via SCM non è più sufficiente.
 - E' necessario intervenire sull'architettura del software e ricorrere a pratiche di SCM più complesse (vedi *Software Product Lines e Variant Management*).
 - Queste pratiche sono supportate da procedure e strumenti di SCM.



Obiettivi dell'implementazione

- La procedura di Software Versioning viene implementata per:
 - Mantenere un archivio del software di proprietà dell'azienda
 - Mantenere un copia di tutte le versioni installate presso i clienti
 - Mantenere copia di tutte le variazioni effettuate sul codice per poter permettere di annullare qualsiasi modifica.
 - Tracciare gli autori delle modifiche
 - Supportare il processo di sviluppo software:
 - Abilitare lo sviluppo in parallelo tra più gruppi o semplicemente più sviluppatori e tra più stati (es. test, stage, produzione)
 - Supportare il processo di trasferimento del codice (nuove implementazioni, patch, ...) tra le versioni
 - Abilitare l'accesso da remoto al repository.
- Portafoglio prodotti:
 - Singolo prodotto e personalizzazione per cliente
 - Progetti speciali basati sul prodotto e no
- Canale di distribuzione:
 - Singola versione per cliente
 - Distribuzione dell'intero prodotto compilato
 - Rare patch di emergenza.

Software Version Control

Terminologia

- **Item:** oggetto di cui vengono gestite le versioni (es. file).
- **Change:** rappresenta una specifica modifica ad un documento sottoposto al controllo di versione.
- **Repository:** archivio delle modifiche.
- **Commit:** operazione di sincronizzazione tra le directory di lavoro ed il contenuto del repository.
- **Change list (o changeset):** identifica un insieme di modifiche fatte in un singolo commit.
- **Update:** copia le modifiche fatte sul repository nella propria directory di lavoro (può essere visto come l'operazione contraria al commit).
- **Merge / Integrazione:** unisce modifiche concorrenti in una revisione unificata.
- **Revisione :** una revisione è una versione in una catena di modifiche.
- **Conflitto:** un conflitto si presenta quando diversi soggetti fanno modifiche allo stesso item. Non essendo il software abbastanza “intelligente” da decidere quale tra le modifiche è quella 'corretta', si richiede ad un utente di risolvere il conflitto.

Aree della fabbrica

- Gli elementi in lavorazione devono poter essere identificati univocamente e tracciati durante il loro percorso nella fabbrica
- L'identificatore viene assegnato dal gestore dei work item
- Le procedure di gestione delle varie aree tracciano le operazioni svolte usando l'identificatore
- SCM funziona come magazzino di semilavorati e prodotti finiti



Pattern per il controllo di versione

Introduzione

- Branch: per ogni prodotto sviluppato od in fase di sviluppo viene creata nel repository un'area di archiviazione (branch) in cui memorizzare l'insieme di oggetti che costituiscono il prodotto.
- Branch owner & policy
 - *Regola: ogni branch ha un responsabile ed una politica di gestione.*
 - La politica di gestione determina quale tipologia di oggetti può essere memorizzato nel branch.
 - Il responsabile del branch definisce la politica di accesso e controlla che sia rispettata.
- Il concetto di “finito”
 - *Assunto: “finito” = “rilasciabile”*
 - Un elemento (modulo, funzione, script ..) è rilasciabile quando ha passato “unit”, “integration” e “functional” tests.
 - Se un elemento è rilasciabile allora in qualsiasi momento un cliente potrebbe dire “Bene, andiamo in produzione ora” senza che nessuno nel gruppo di sviluppo possa dire “si ma aspetta!!”
 - Attenzione: finito = “regression tested”
- Lo sviluppo deve procedere senza tempi di attesa anche se due sviluppatori operano sullo stesso modulo (vedi librerie condivise). Configurare il tool di gestione delle versioni per l'opzione copy-modify-merge.

Pattern per il controllo di versione

Aree di lavoro (workspace)

- Ogni sviluppatore dispone di un'area riservata sulla propria postazione di lavoro in cui creare i componenti software, correggerli e provarli (debug).
- Possiamo considerare quest'area come un ramo del repository esteso alla stazione di lavoro.
- Anche questo ramo ha una sua politica di gestione:
 - *L'area di lavoro è privata e non può essere condivisa (nessun tipo di filesystem sharing, un'area di lavoro per ogni sviluppatore).*
 - *Si opera solo all'interno delle aree di lavoro (nessuna copia temporanea su cui sviluppare).*
 - *Sincronizzare frequentemente (tutte le mattine) il proprio lavoro con quello degli altri membri del gruppo.*
 - *Salvare (check-in) frequentemente (tutte le sere) il lavoro svolto.*

Pattern per il controllo di versione

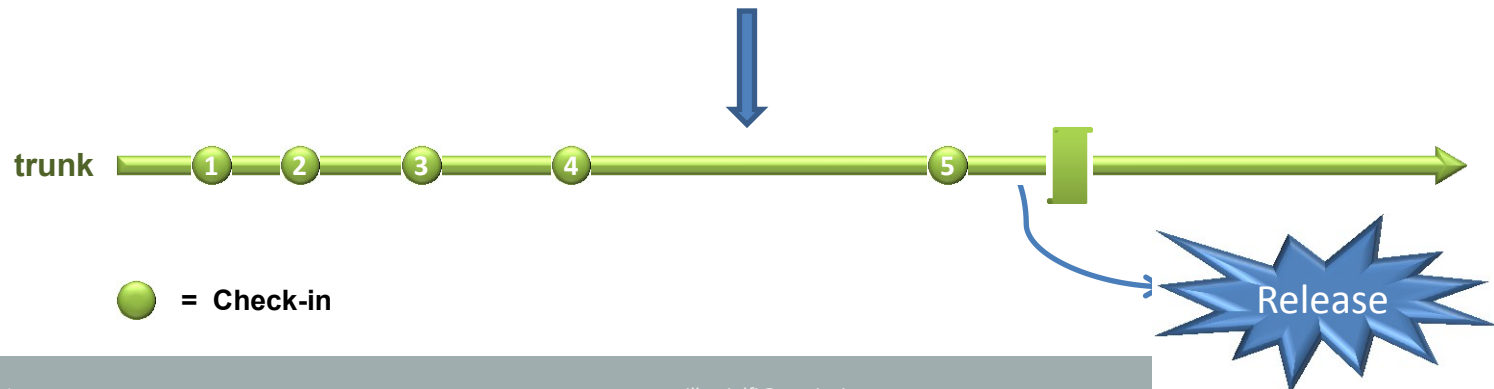
Ramo principale (trunk branch)

- Quando un elemento “finito” deve essere depositato nel repository.
- Deve esistere un ramo del repository in cui depositare l’elemento finito per un successivo rilascio in produzione. Questo è il ramo dei prodotti finiti.
- Un qualsiasi ramo può essere quello dei prodotti finiti. Il ramo principale del repository (altrimenti detto trunk, mainline,..) è un buon punto di partenza.

Assunto: trunk è il ramo dei prodotti finiti.

- Politica di gestione di trunk
 - *Gli elementi possono essere rilasciati in qualsiasi momento.*

Nell’esempio seguente viene eseguito un rilascio contenente cinque elementi.



Pattern per il controllo di versione

Quando creare nuovi rami (branch)

- Il più raramente possibile.
 - Più rami = Maggiori costi.
 - Più rami = Maggiori conflitti = Maggiori tempi di sviluppo
- Creare un nuovo ramo solo quando c'è qualcosa da memorizzare e non c'è altro ramo del repository che possa essere usato senza violarne la politica di accesso.
- Motivi per nuovi branch:
 - Nuovi rilasci (release): la release deve essere mantenuta nel tempo anche dopo il rilascio di successive release.
 - Nuovi metodi di rilascio (service packs, fix, ..)
 - Più gruppi di sviluppo sullo stesso prodotto
 - Sviluppo di nuove funzionalità

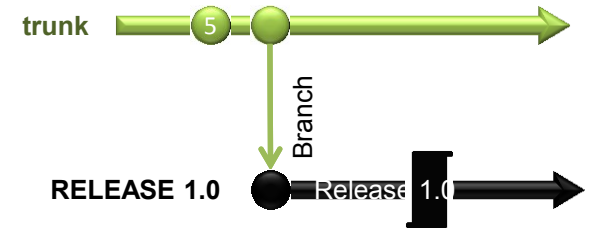
Pattern per il controllo di versione

branch for release

- Ogni rilascio ha bisogno di un'area di parcheggio per essere salvaguardato e mantenuto nel tempo.

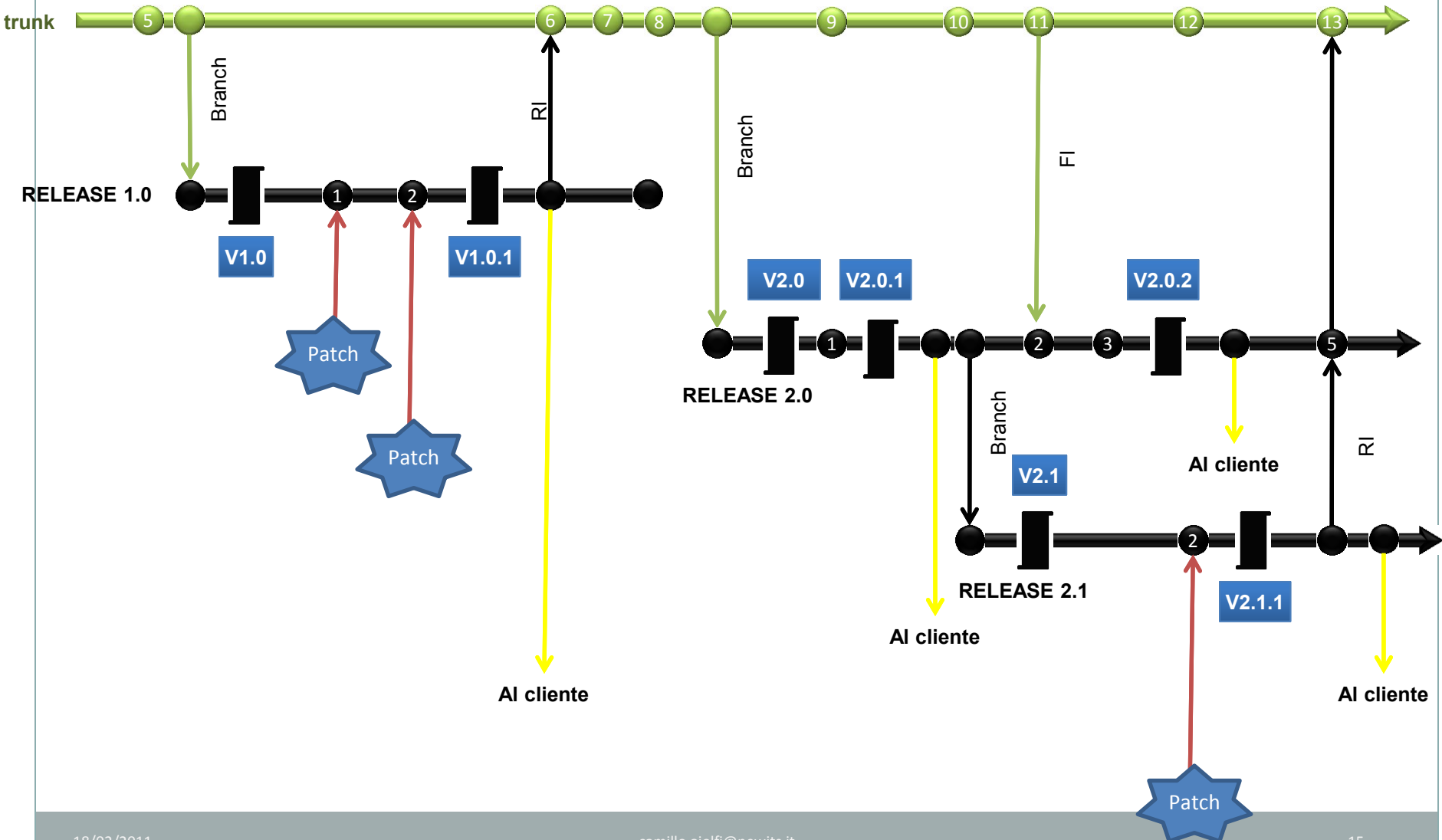
Branch = Release

- Viene definito un nuovo rilascio per:
 - Nuovo pacchetto destinato a più clienti
 - Nuovo pacchetto destinato ad un singolo cliente
 - Nuovo pacchetto destinato ad un nuovo ambiente (test, stage, produzione,...)
- Ad ogni nuovo rilascio viene creato un branch
- La creazione di un branch è immediatamente seguita dalla creazione di un identificativo (tag, label, ..) che ne fissa l'immagine del contenuto all'istante della creazione. Se il contenuto del branch verrà successivamente modificato tale immagine non subirà cambiamenti.
- Il normale sviluppo continua su trunk.
- Il software nel branch appena creato viene rigorosamente testato.
- Le correzioni (patch, hotfix) vengono applicate alla release/al branch.
- Se opportuno vengono riportate su trunk (Merge - Reverse Integration)
- A test ultimato viene applicato un nuovo tag ed il software rilasciato al cliente.
- Durante la vita della release si possono scoprire problemi su trunk da riportare eventualmente sul branch (Merge = Forward Integration)



Pattern per il controllo di versione

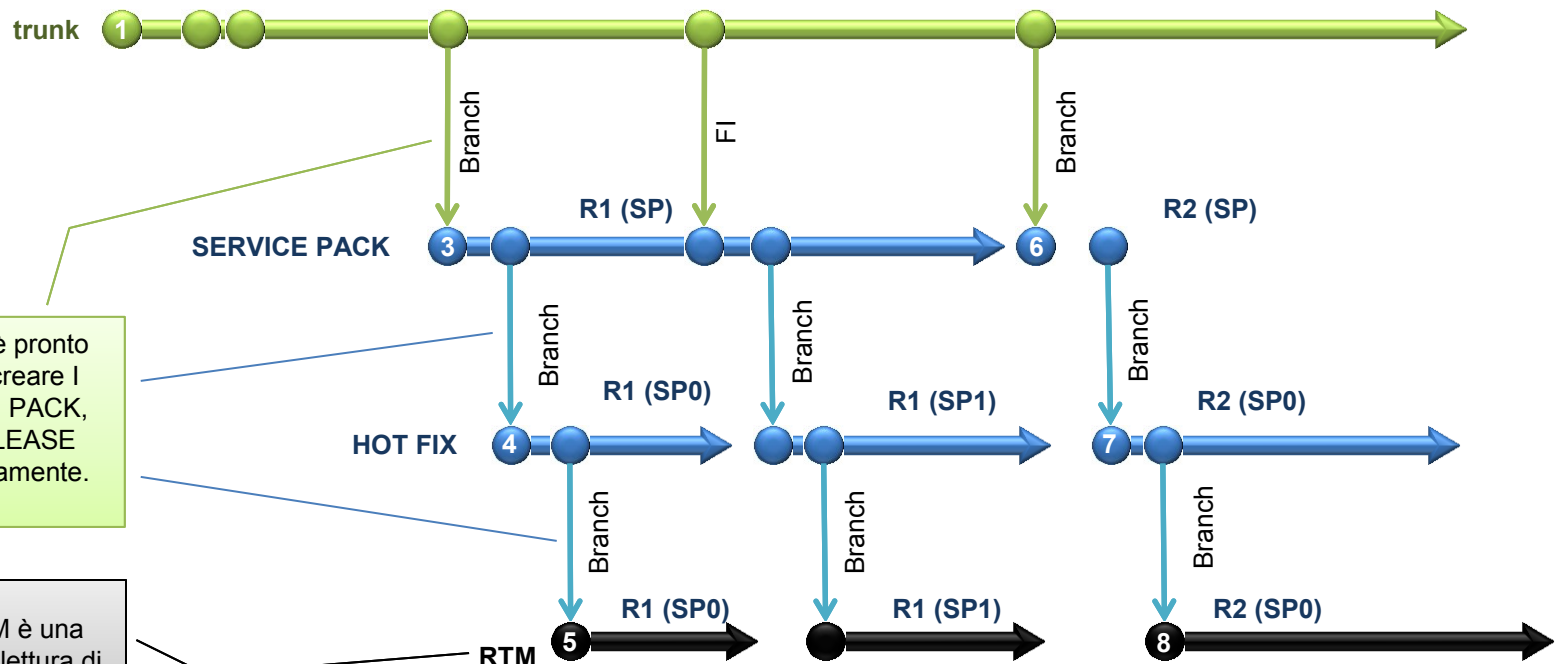
branch for release - schema base



Pattern per il controllo di versione

branch for release - schema alternativo

- In funzione dello schema di servizio offerto si devono creare rami dedicati ai vettori di rilascio previsti (service pack, hotfix, release)
- Ricordare: + rami = + tempo = + costi
- Ricordare: lo schema di servizio va ripetuto per ogni variante (cliente, piattaforma, ...)



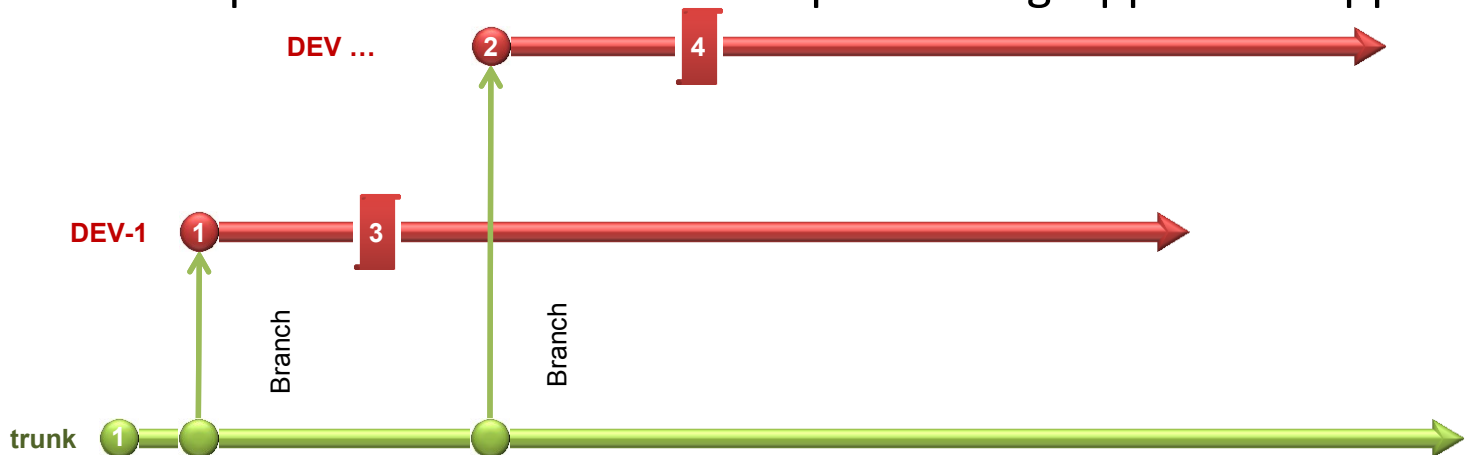
Quando trunk è pronto per il rilascio creare i rami SERVICE PACK, HOT FI e RELEASE contemporaneamente.

Il ramo RTM è una copia in sola lettura di quanto rilasciato

Pattern per il controllo di versione

branch for development - rami di sviluppo

- Lo sviluppo del prodotto continua su trunk.
- Quando un nuovo sviluppo non può essere realizzato senza violare la politica di trunk si crea un nuovo ramo.
- Esempio:
 - Sviluppo di un oggetto di durata superiore ad un giorno.
 - La politica dei workspace richiede di salvare il lavoro tutti i giorni.
 - Non si può salvare l'oggetto in trunk senza violarne la politica visto che l'oggetto non è "finito".
 - Si deve creare un ramo temporaneo di sviluppo.
- L'inizio di uno sviluppo deve essere marcato con un tag.
- Sul nuovo ramo possono salvare tutti i componenti il gruppo di sviluppo.



Pattern per il controllo di versione

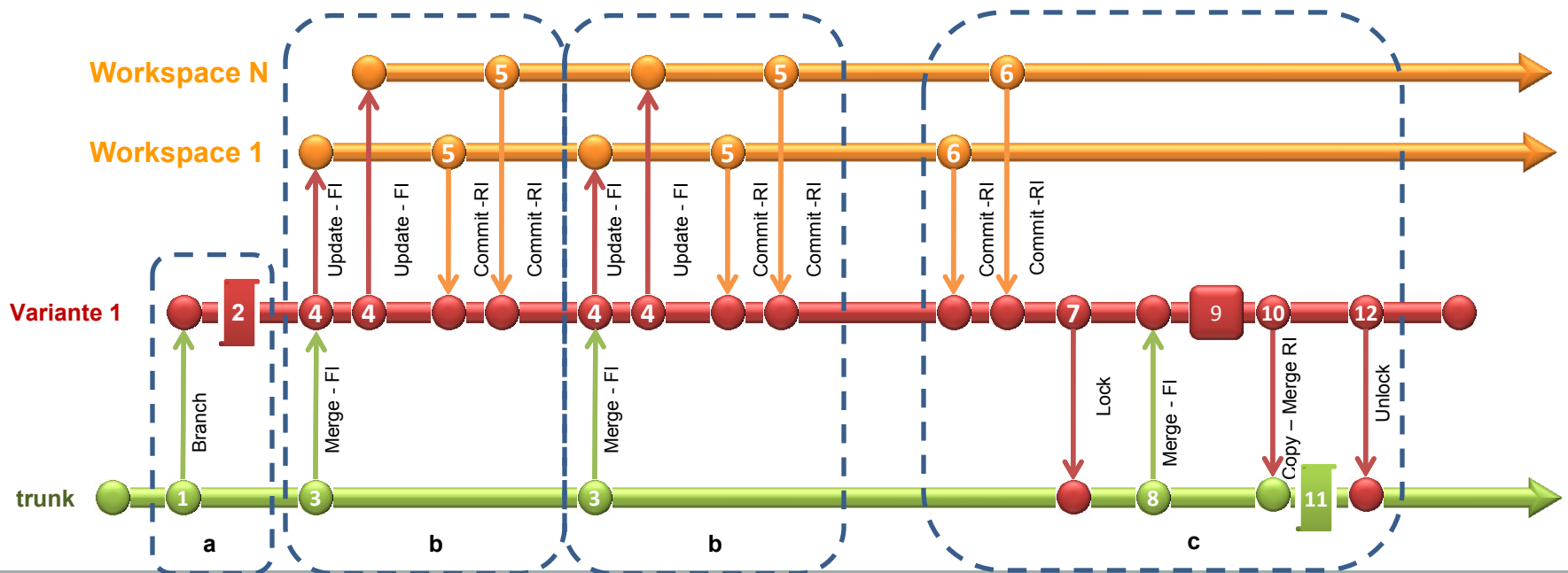
branch for development - branch policy

- La politica di gestione del ramo di sviluppo condiziona ed è condizionata dalla metodologia.
- Prendiamo in prestito un obiettivo di SCRUM ed introduciamo una piccola variazione:
L'implementazione di una nuova funzionalità di un prodotto software deve essere suddiviso in unità elementari sviluppabili (fino alla fase di Unit Testing) in una giornata di lavoro.
- Politica di gestione:
 - *Dopo ogni check-in il ramo deve essere compilabile senza errori.*
 - *Gli item salvati in un ramo di sviluppo sono "Unit Tested"*
- Questo condiziona la metodologia costringendo a suddividere l'architettura dei moduli software in unità elementari (classi, funzioni, procedure, scripts, ..) realizzabili e testabili nel corso di una giornata.
- Se si elimina il vincolo "Unit tested" la politica è meno stringente permettendo, ad esempio, il rilascio di scheletri di classe (class skeleton).

Pattern per il controllo di versione

branch for development – operatività

- Creazione nuovo ramo per lo sviluppo di una nuova variante di prodotto (es. personalizzazione per un cliente). Al ramo viene applicata una etichetta.
- Operazioni periodiche. Come da politica dei rami di sviluppo, il periodo è pari ad un giorno.
- Rilascio della variante “finita”. Il ramo viene chiuso.



Pattern per il controllo di versione

branch for development – operatività (continua)

a) Creazione iniziale del ramo.

1. Branch. Viene creato il nuovo ramo di sviluppo. L'operazione è istantanea e va fatta all'inizio delle operazioni.
2. Tag. Viene applicata una etichetta per identificare lo stato originale del ramo.

b) Operazioni periodiche

3. Merge. Il responsabile del ramo di sviluppo provvede a sincronizzarlo con trunk; vengono raccolti i rilasci degli altri team di sviluppo. Si procede alla risoluzione di eventuali conflitti.
4. Update. Ogni sviluppatore sincronizza la propria area di lavoro con il ramo di sviluppo; vengono raccolti i rilasci degli altri team di sviluppo e degli altri membri del team. Si procede alla risoluzione di eventuali conflitti.
5. Commit. La produzione periodica viene rilasciata sul ramo di sviluppo compatibilmente con la politica del ramo stesso (es. gli item non testati non si rilasciano).

c) Rilascio

6. Commit. Vengono rilasciati gli ultimi sviluppi da parte di tutti i membri del team.
7. Lock. Ove ammesso dalle procedure e dagli strumenti usati, il ramo trunk viene bloccato per impedire ulteriori rilasci da parte di altri team.
8. Merge pre rilascio. Vengono recuperati eventuali rilasci di altri team. Si risolvono gli ultimi conflitti.
9. Build e test finale (regression test). Si costruisce il prodotto software completo e si procede con il test finale.
10. Merge – Copy - Reintegrate. Rilascio del prodotto “finito” su trunk; viene copiato il ramo di sviluppo su trunk.
11. Tag. Il rilascio viene etichettato con una opportuna label.
12. Unlock. Se si è bloccato trunk si procede con lo sblocco.

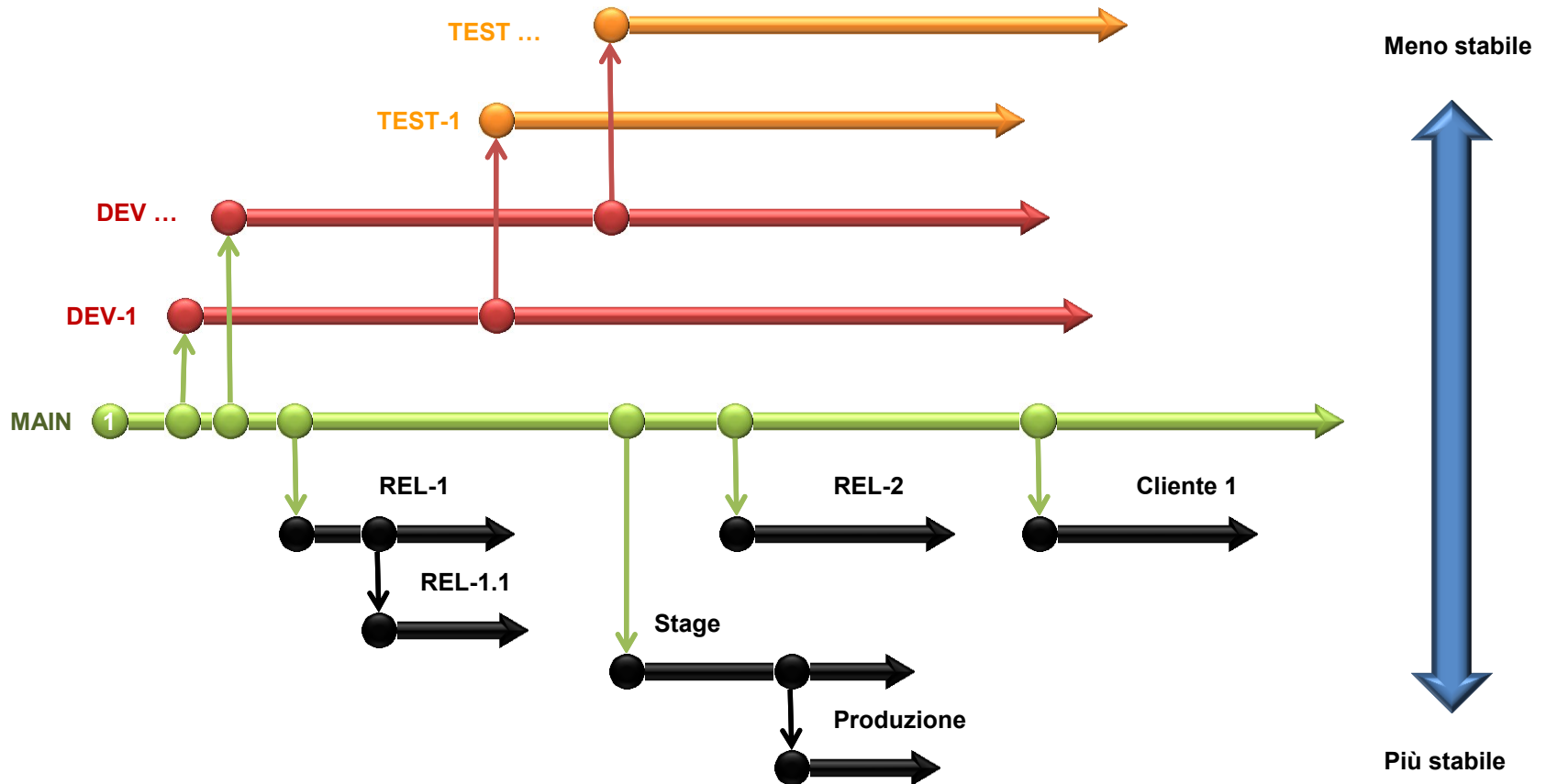
Pattern per il controllo di versione

Regole generali

- Risolvere i conflitti sul ramo meno stabile.
 - ES. I conflitti vanno risolti sui rami di sviluppo e non su trunk
 - Maggiore è la frequenza di merge e minore è la probabilità di conflitti complessi.
- Pubblicare il lavoro su trunk il più frequentemente possibile e non solo al momento del rilascio di nuove release.
 - Effetto collaterale: chi prima rilascia su trunk vince! Eventuali conflitti devono essere risolti da altri.
- Per quanto possibile limitare i rilasci (changeset) a singole unità elementari di lavoro (work-item) come patch, classi, singole pagine o parti di pagina.
- Tracciare i singoli rilasci mediante l'accoppiata changset-id e work-item-id in modo da poter sempre individuare i rilasci che contengono un particolare work-item. Per iniziare, un tool di bug-tracking è sufficiente.

Pattern per il controllo di versione

Vista d'insieme



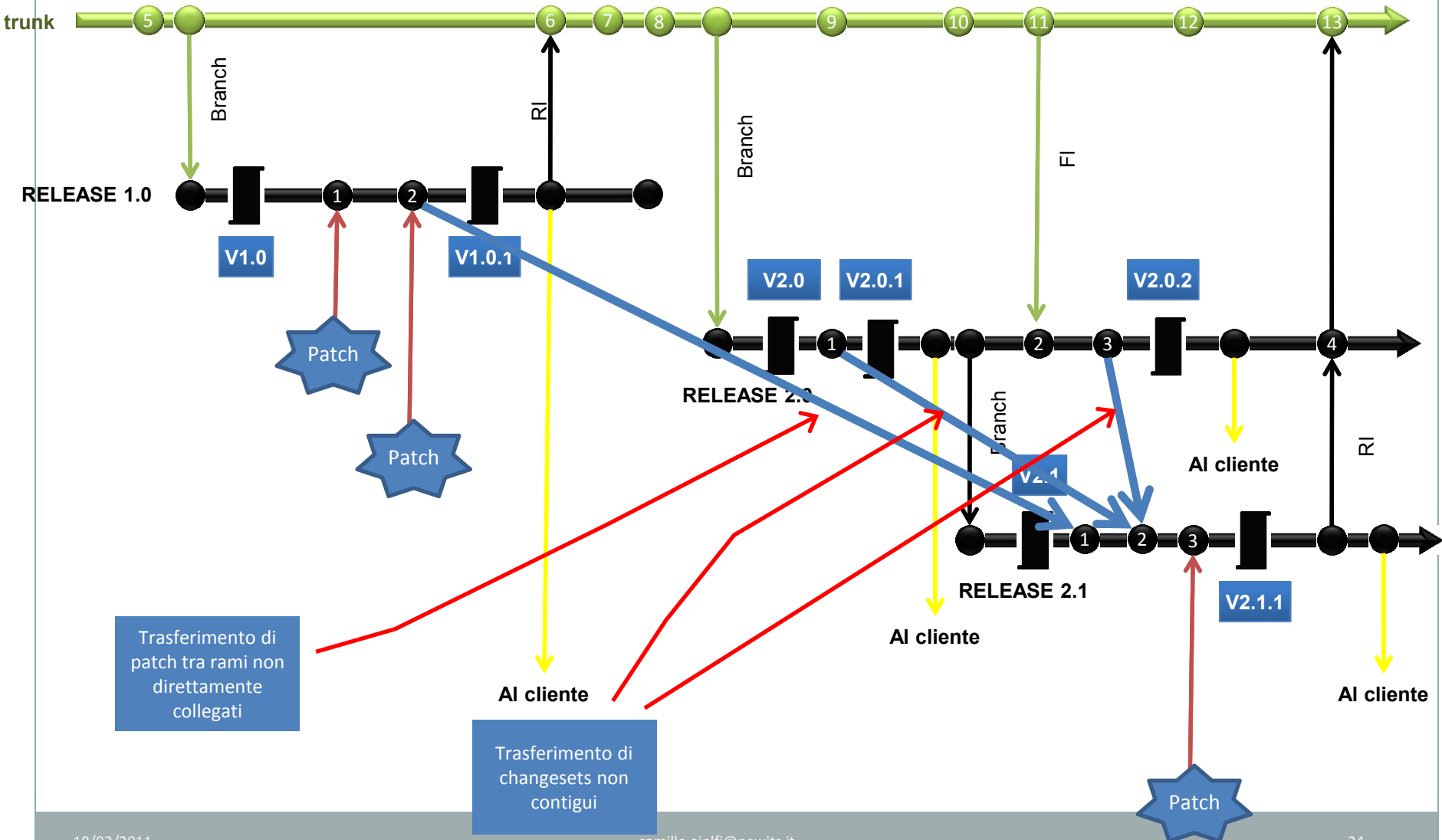
Pattern per il controllo di versione

Note sulle operazioni di merge

- Le operazioni di merge sono pericolose quanto tutte le altre operazioni di sviluppo.
- Ogni operazione di merge dovrebbe essere seguita da una fase di test.
- L'automazione dei test incrementa il tempo di scrittura del codice ma permette di eseguirli di frequente, aumenta la qualità del software e diminuisce il costo globale di sviluppo.
- I conflitti devono essere risolti ed il risultato deve essere testato.
- I conflitti aumentano e le operazioni di merge si complicano al crescere della distanza tra ramo di origine e ramo di destinazione sia in termini di tempo di aggiornamento che percorso di navigazione nell'albero.
- E' necessario minimizzare le operazioni di merge tra rami non direttamente collegati e le operazioni di merge di changeset non contigui (baseless merge, cherrypicking).

Pattern per il controllo di versione

Baseless merge - cerry picking



Prossimi passi

- Automazione dei test
- Passaggio alla gestione *Software Product Lines e Variant Management (riusabilità del software)*.

Riferimenti

- **InfoQ:** Agile version control with multiple teams
- **Microsoft:** Team Foundation Server 2008 branching guidance
- **Vari:** Version control with Subversion
- **Perforce Software:** High level best practices in Software Configuration Management
- **Perforce Software:** Branching and merging in the face of agile development, extreme programming, team collaboration, and parallel releases.
- **Stanford University:** Branching and merging with CVS