Evoluzione e manutenzione

Una definizione

La manutenzione del sw è la modifica di un prodotto sw dopo la consegna per correggere difetti, migliorare le prestazioni o altri attributi, o per adattare il prodotto a un ambiente diverso (ANSI / IEEE, 1983)

Leggi dell'evoluzione del sw (Lehman 1980, Lehman e Belady 1985)

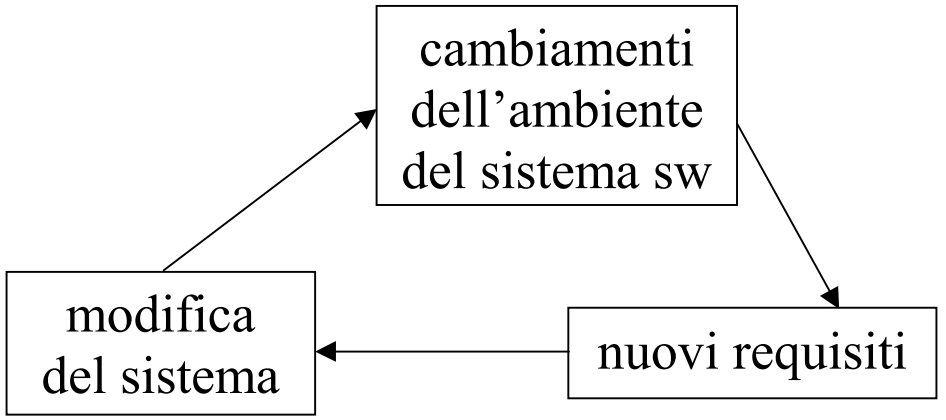
Sono cinque ipotesi derivate dall'analisi (misurata) della crescita di numerosi sistemi sw di grandi dimensioni

- *Cambiamento continuo*: ogni programma in uso o sottostà a cambiamenti continui o diventa progressivamente meno utile; il processo di cambiamento o di decadenza continua finché non diventa più conveniente sostituire il programma

Es. motivi di cambiamento: nuovi vincoli, interazione con altri sistemi, aumento del numero degli utenti, *porting* su altre piattaforme, riscrittura in nuovi linguaggi

Interpretazione:

la manutenzione del sw è inevitabile perché il processo di evoluzione del sw è ciclico



Leggi dell'evoluzione del sw (cont.)

♦ *Complessità in aumento*: la complessità del sw in evoluzione tende continuamente a crescere, a meno che non vengano pianificate delle attività esplicitamente mirate a ridurla

Interpretazione:

- La struttura di un sistema sw è in continuo degrado (vedi sistemi legacy)
- L'aumento di complessità è spesso dovuto all'uso di patch
- La manutenzione preventiva serve per ridurre il degrado

Leggi dell'evoluzione del sw (cont.)

♦ *Evoluzione dei grandi programmi*: l'evoluzione di un programma è un processo auto-regolante con tendenze e invarianti (fra release), relative agli attributi del sistema, determinabili statisticamente.

Es. attributi del sistema: intervallo di tempo fra due release, numero di errori rilevati in una release

Interpretazione:

- È la legge più dibattuta, che ha stimolato la ricerca di "verità universali"
- Tanto più grandi sono le dimensioni del sistema, tanto minore è il cambiamento fra una release e la successiva perché un grosso cambiamento introdurrebbe molti nuovi errori che limiterebbero l'utilità dello stesso dopo la consegna

Leggi dell'evoluzione del sw (cont.)

♦ *Conservazione della stabilità organizzativa (tasso di lavoro invariante)*: durante la vita attiva di un programma, il tasso globale di attività (sviluppo) è statisticamente invariante.

Interpretazione:

- Non si verificano fluttuazioni ampie o selvagge negli attributi organizzativi, quali la produttività
- È supportata da (Brooks 1975): a un certo punto, le risorse e l'uscita (del gruppo di lavoro) raggiungono un livello ottimale e aggiungere più risorse non cambia l'uscita in modo sostanziale → sostiene implicitamente che la maggior parte dei progetti vengono condotti in condizioni di "saturazione"
- Conferma che le grandi squadre di sviluppo sono improduttive perché sopraffatte dall'onere della comunicazione

Leggi dell'evoluzione del sw (cont.)

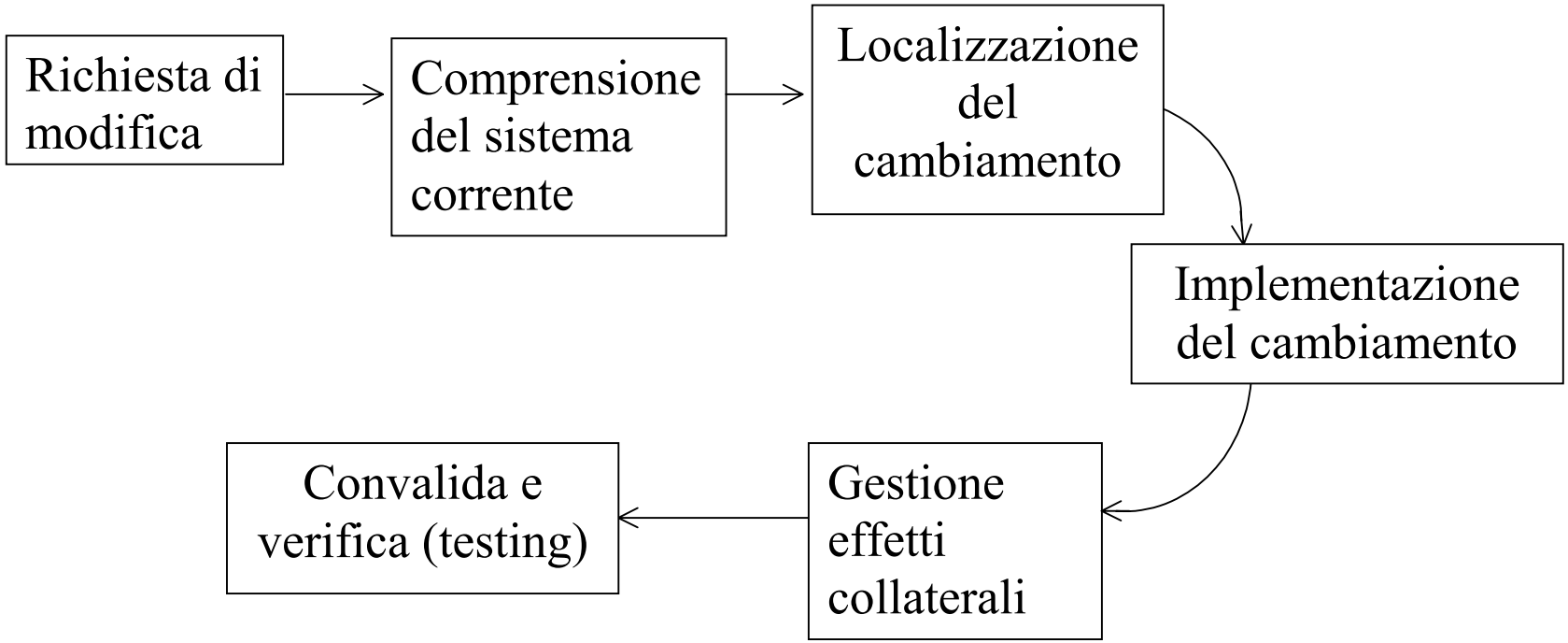
- *Conservazione della familiarità (complessità percepita)*: durante la vita attiva di un programma in evoluzione, il contenuto di ciascuna release (cambiamenti, aggiunte, cancellazioni) è statisticamente invariante.

Evoluzione del sw e declino del sistema

Si può decidere di sostituire il sistema anziché manutenerlo se si risponde positivamente ad alcune delle seguenti domande:

- Il costo della manutenzione è troppo elevato?
- L'affidabilità del sistema è inaccettabile?
- È difficile adattare il sistema entro una scadenza ragionevole?
- Le prestazioni del sistema non rispettano i vincoli?
- Le funzionalità del sistema sono di utilità limitata?
- È possibile fare lo stesso lavoro meglio, più rapidamente o spendendo meno usando altri sistemi?
- Il costo di manutenzione dell'hw è tale da giustificare la sostituzione con hw nuovo più economico?

Attività del processo di manutenzione



Attività del processo di manutenzione (cont.)

Comprensione del sistema corrente

- Assorbe dal 50% al 90% del tempo dedicato alla manutenzione
- In termini temporali, è basata 3.5 volte più sul codice che su altri documenti (spesso ciò deriva dalla inattendibilità della documentazione)

Localizzazione del cambiamento

È supportata dall'analisi automatica del codice

Gestione effetti collaterali

- Gli effetti collaterali di una modifica del codice sulle altre parti del programma prendono il nome di *ripple effects*
- La loro gestione è supportata dall'analisi del codice, con generazione automatica delle dipendenze

Dipendenze

- Dipendenze associate alla programmazione in piccolo:
  - Flussi di dati
  - Flussi di controllo

- Dipendenze associate alla programmazione in grande:
  - Importazione / esportazione di risorse
  - Accesso a entità
  - Ereditarietà / associazioni / aggregazioni

Attività sede del cambiamento richiesto	**Artefatti da modificare di conseguenza**
Analisi dei requisiti	Specifica dei requisiti +
Design di sistema	Specifica dell'architettura Specifica tecnica del design +
Design di programma	Specifica del design del programma +
Implementazione di programma	Codice del programma Documentazione di programma +
Testing di unità	Piani di test Script di test +
Testing di sistema	Piani di test Script di test +
Consegna del sistema	Documentazione utente Materiale per l'addestramento Documentazione per l'operatore Guida di sistema Guida del programmatore

N.B. La manutenzione di norma non comporta grosse modifiche architetturali

Manutenzione: una classificazione

- Manutenzione correttiva (21%): correzione dei guasti che quotidianamente vengono scoperti
- Manutenzione adattiva (25%): modifiche secondarie necessarie come conseguenza di modifiche primarie, cambiamenti dell'hw o dell'ambiente
- Manutenzione perfettiva (50%): cambiamenti che migliorano qualche aspetto del sistema (es. chiarificazione della documentazione, aumento della copertura dei test, ecc.)
- Manutenzione preventiva (4%): cambiamento di qualche aspetto del sistema al fine di prevenire i malfunzionamenti (es. miglioramento della gestione degli errori, introduzione di controlli di tipi, ecc.); tipicamente si effettua quando si scopre, e quindi corregge, un guasto effettivo o potenziale che non si è ancora manifestato sotto forma di malfunzionamento

Manutenzione: responsabilità

- La manutenzione assorbe la quantità prevalente delle risorse impiegate nel ciclo di vita del sw (stima per il decennio corrente, riportata da Pfleeger: 80% dei costi)
- Essa è condotta da un team, comprendente analisti, progettisti e programmatori (questi ultimi hanno un ruolo più vasto che non nello sviluppo perché è richiesta una conoscenza profonda del codice)
- Il team è composto prevalentemente da persone che non hanno partecipato allo sviluppo ed, eventualmente, da una o più persone che vi hanno partecipato: una squadra nuova è più obiettiva

Sistemi ereditati (legacy systems)

Sistemi per i quali l'attività di manutenzione è diventata prevalente su ogni altra; caratteristiche:

- Sono stati implementati diversi anni fa
- La loro tecnologia (linguaggi di programmazione, stile di codifica, hw) è diventata obsoleta
- Sono stati mantenuti per un lungo periodo
- La loro struttura si è deteriorata e non facilita la comprensione del codice
- La loro documentazione è diventata obsoleta
- Contengono regole di business che non sono documentate altrove
- Non possono essere sostituiti facilmente
- Rappresentano un grosso investimento per l'azienda
- Gli autori originali non sono più disponibili
- La loro evoluzione costituisce una sfida particolare

Manutenzione dei legacy system

Obiettivo: migliorare la qualità del sw contenendo i costi; approcci principali:

- Redocumentation (mediante analisi statica del codice)
- Restructuring / refactoring = trasformazione di codice mal-strutturato in codice ben-strutturato
- Reverse engineering = creazione del design e delle specifiche a partire dal codice
- Re-engineering = reverse engineering + modifica di specifiche e design + forward engineering → creazione di un nuovo sistema basato su specifiche e design rivisitati

Redocumentation

- La documentazione prodotta può essere grafica o testuale
- Non esistono metodi consolidati di ridocumentazione
- Tipicamente il processo inizia sottoponendo il codice a uno strumento di analisi statica, che produce in uscita:
  - ✓relazioni di chiamata fra componenti
  - ✓gerarchia delle classi
  - ✓tavole di interfaccia dei dati
  - ✓dizionario dei dati
  - ✓diagrammi di flusso dei dati
  - ✓diagrammi di flusso del controllo
  - ✓pseudocodice
  - ✓cammini di test
  - ✓riferimenti incrociati di componenti e variabili

Restructuring

- È basata su metodi consolidati di ristrutturazione del codice
- È effettuata da strumenti automatici che analizzano staticamente il codice, generano una rappresentazione interna (ad es. mediante grafi) dello stesso e provvedono a
  - eliminare il codice morto (es. rami con condizioni impossibili)
  - semplificare le strutture di controllo (“spaghetti code”)
  - eliminare i cloni

Spaghetti code: esempio

```
Start:   Get (Time-on, Time-off, Time, Setting, Temp, Switch)
         if Switch = off goto off
         if Switch = on goto on
         goto Cntrld
off:     if Heating-status = on goto Sw-off
         goto loop
on:      if Heating-status = off goto Sw-on
         goto loop
Cntrld:  if Time = Time-on goto on
         if Time = Time-off goto off
         if Time < Time-on goto Start
         if Time > Time-off goto Start
         if Temp >= Setting then goto off
         if Temp < Setting then goto on
Sw-off:  Heating-status := off
         goto Switch
Sw-on:   Heating-status := on
Switch:  Switch-heating
loop:    goto Start
```

Logica di controllo strutturata

```
loop
  -- The Get statement finds values for the given variables from the
  -- system's environment.
  Get (Time-on, Time-off, Time, Setting, Temp, Switch) ;
  case Switch of
    when On => if Heating-status = off then
            Heating-status := on ; Switch-heating ;
          end if ;
    when Off => if Heating-status = on then
            Heating-status := off ; Switch-heating ;
          end if;
    when Controlled =>
      if Time >= Time-on and Time < = Time-off then
        if (Temp >= Setting or Time = Time-off) and Heating-status = on
          then Switch-heating; Heating-status = off;
        elsif (Temp < Setting or Time = Time-on) and Heating-status = off
          then Heating-status := on ; Switch-heating;
        end if;
      end if ;
  end case ;
end loop ;
```

Semplificazione delle condizioni

Condizione complessa
if not (A > B **and** (C < D **or not** (E > F)))...

Condizione semplificata
if A <= B **or** (C>= D **and** E > F)...

Ristrutturazione del design

- Modularizzazione
- Riorganizzazione dei dati

Refactoring

- Identificazione di astrazioni condivise
- Conversione ereditarietà / aggregazione
- Specializzazione delle operazioni nei sottotipi

Reverse engineering

- È realizzata da strumenti automatici che incarnano metodi consolidati, tipicamente includendo l'analisi statica del codice
- Funziona bene se le aspettative sono modeste

Re-engineering

- È un argomento di ricerca
- Attualmente è basata sulla combinazione di trasformazioni semi-automatiche e interventi umani

Fattori che influenzano l'entità dello sforzo di manutenzione

Fattori non tecnici

- Disponibilità e avvicendamento del personale
- Durata attesa del sistema (tanto più è lunga, tanta più cura richiede la manutenzione)
- Dipendenza dall'ambiente
- Affidabilità dell'hw
- Novità dell'applicazione

Fattori tecnici

- Qualità del design (modularità ecc.)
- Qualità del codice
- Linguaggio di programmazione
- Qualità della documentazione
- Qualità del testing
- Tipo di applicazione (i sistemi in tempo reale e altamente sincronizzati sono più difficili da modificare)
- Novità dell'implementazione

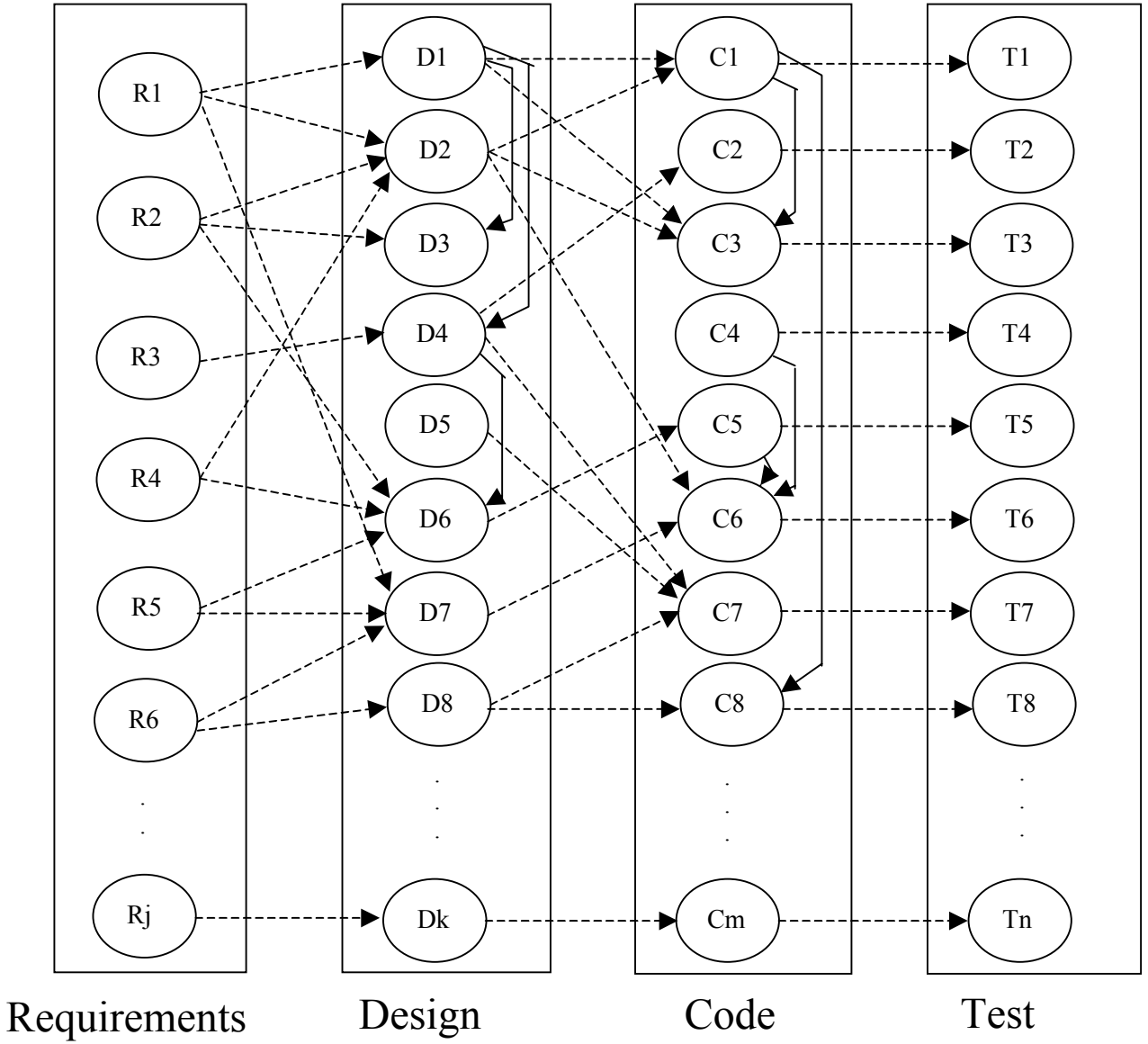
Analisi di impatto

Valutazione dei rischi (effetti su risorse, costi e scheduling) di una modifica proposta per ciascun artefatto coinvolto nella modifica; produce in uscita le seguenti misure:

- *Tracciabilità verticale* = relazioni di dipendenza tra le parti di un artefatto (es. tra i requisiti nel documento di specifica)
- *Tracciabilità orizzontale* = relazioni di dipendenza tra le parti di artefatti prodotti consecutivamente nello sviluppo del sw

Ogni modifica apportata non deve aumentare la complessità del grafo di tracciabilità

Grafo di tracciabilità complessivo



Problemi di sviluppo e manutenzione

- Doppia modifica: l'esistenza di copie/cloni dello stesso frammento di codice comporta che le correzioni vadano replicate
- Modifiche simultanee: programmatori diversi possono modificare concorrentemente librerie di codice condiviso, con possibilità di interferenza tra le modifiche
- Copie locali: programmatori diversi possono lavorare contemporaneamente su copie locali dello stesso modulo
- Tracciabilità: modifiche che non ripercorrono tutte le fasi di sviluppo possono portare ad artefatti inconsistenti tra di loro

↓

necessità di coordinare (identificare, organizzare e controllare) le operazioni di modifica degli artefatti prodotti nelle varie fasi e iterazioni di sviluppo e manutenzione del sw

Famiglie di prodotti

Spesso un'applicazione è in realtà costituita da una famiglia di prodotti, che si differenziano, ad es., per:

- ambiente operativo (hw: processore, sw: sistema operativo)
- ambito legislativo per cui sono concepiti
- funzionalità offerte

Repository condiviso = totalità dei moduli che servono per costruire i prodotti della famiglia + info circa la componibilità degli stessi (sfruttate da strumenti di costruzione)

Configurazione = collezione dei componenti di un sistema che rappresenta un prodotto (cioè una delle varianti) della famiglia

↓

necessità di controllare le differenze fra configurazioni
al fine di minimizzare rischi ed errori di sviluppo e testing

Gestione delle configurazioni

È la risposta ai due tipi di necessità evidenziati sopra; supporta:

- identificazione del sw: ogni modulo è identificato univocamente, assieme alla sua versione
- controllo delle configurazioni: le operazioni di modifica dei moduli di una certa configurazione sono disciplinate
- convalida e verifica delle configurazioni: ogni configurazione corrisponde ai requisiti espressi dall'utente per quella configurazione
- evoluzione delle configurazioni: è possibile avere informazioni sulla storia dei moduli

Evoluzione delle configurazioni nel tempo

- *Major release/ version*: riscritture o cambiamenti radicali; è di solito associata a una consegna completa del prodotto al cliente
- *(Minor) release*: modifiche limitate, spesso correttive; è di solito associata a rilasci parziali o a patch

Versione

Il termine ha un sovraccarico di significati:

- Configurazione di un'applicazione (sinonimo generico)
- Major release di un sistema (già visto nel lucido precedente) → sistema *n.m*, dove *n* è la versione e *m* la (minor) release
- Versione di un singolo modulo: evoluzione dovuta a interventi correttivi, adattativi o perfettivi

Terminologia secondo Sommerville

- *Version* An instance of a system which is functionally distinct in some way from other system instances
- *Variant* An instance of a system which is functionally identical but non-functionally distinct from other instances of a system
- *Release* An instance of a system which is distributed to users outside of the development team

Gestione delle configurazioni: strumenti di supporto

- Gestione delle versioni dei moduli: di ciascuna è memorizzata la forma completa oppure solo le variazioni (delta) rispetto a un modulo di riferimento
- Controllo dell'accesso concorrente ai moduli: meccanismi di lock garantiscono che un solo utente alla volta possa accedere alla stessa versione in scrittura mentre tutti possono accedervi in lettura
- Gestione delle informazioni che documentano ciascuna versione di un modulo:
  - nome e versione del tool usato
  - nome e versione del codice sorgente
  - autore del modulo
  - descrizione del modulo
  - descrizione delle modifiche apportate rispetto alla versione precedente
  - data di modifica
  - direttive di compilazione

Repository di gestione delle configurazioni

Contiene info per il controllo dei cambiamenti (problemi riscontrati, organizzazione che li ha scoperti, chi e quando li ha risolti, chi ha autorizzato il cambiamento, priorità del cambiamento, eventuale controllo di versione, ecc.)

A fronte di ciascun malfunzionamento segnalato dagli utenti, è necessario poter costruire esattamente la stessa versione dell'applicazione usata dal cliente per poter riprodurre la condizione di errore rilevata

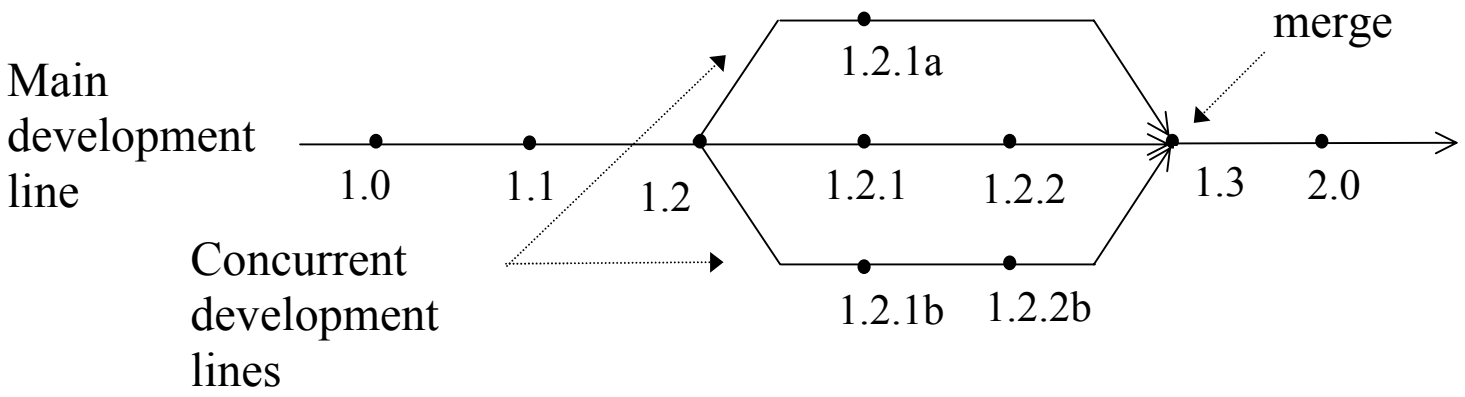
Repository di gestione delle configurazioni: accesso esclusivo a un modulo

Operazioni fondamentali

- `checkout [version] [lock] ⟨module⟩` : richiede una copia locale del modulo. Si può specificare quale versione si desidera. Se si intende modificare tale modulo, si dovrà chiedere l'accesso esclusivo (lock), che verrà concesso solo se nessun altro utente ne è già in possesso. Se non è richiesto il lock, il modulo è disponibile in sola lettura
- `checkin ⟨module⟩` : viene inserita nel deposito dei moduli una nuova versione del modulo e viene rilasciato il lock sullo stesso

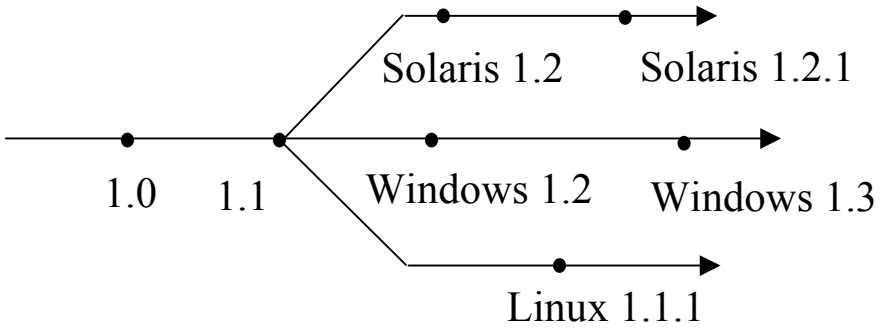
Repository di gestione delle configurazioni: accesso concorrente a un modulo

In alcuni casi può essere conveniente poter lavorare in parallelo sulla stessa versione di un modulo. Ciò richiede un'operazione finale di riconciliazione (merge) delle diverse versioni nel tronco principale di sviluppo



Repository di gestione delle configurazioni: accesso parallelo

In alcuni casi è necessario mantenere più versioni parallele attive, senza prevedere alcuna operazione di merge, in quanto esse sono relative, ad es., a diversi clienti, piattaforme, funzionalità offerte dal pacchetto. Al nome di tali versioni si può aggiungere un nome simbolico



Librarian / configuration control board

Gruppo costituito da rappresentanti di tutte le parti interessate, compresi clienti, sviluppatori e utenti, responsabile di pianificazione, gestione e rilascio di nuove versioni del sistema. Compiti:

- approvazione delle richieste di modifica
- amministrazione del tool di configuration management
- autorizzazione di check in / check out di moduli
- coordinamento dei merge tra tronchi concorrenti di sviluppo

Strumenti automatici di gestione delle configurazioni

- RCS, CVS (dominio pubblico)
- Clear Case (commerciale)
- …

Strumenti di costruzione dell'eseguibile

Uniti agli strumenti di configuration management, permettono di controllare le varianti del sistema e l'evoluzione temporale dei moduli componenti.
Consentono di:

- ricompilare solo i moduli modificati o dipendenti da moduli modificati
- specificare diverse varianti dell'applicazione, che vengono costruite usando i moduli necessari a ciascuna variante; a tal fine alcuni sono dotati di Module Interconnection Language (MIL) con cui descrivere la scomposizione logica del sistema, le relazioni tra componenti logiche e componenti fisiche e le dipendenze che vincolano la costruzione dell'eseguibile associato a ciascuna variante
- specificare direttive di compilazione diverse per le diverse varianti del sistema

Lo strumento più semplice di costruzione dell'eseguibile è *make*