# Evoluzione del software e gestione del cambiamento

# Indice

# 7. Evoluzione del software e gestione del cambiamento

> *Round up the usual suspects.*
> Scena finale del film *Casablanca* [1]

**Scopo della lezione**

- introduzione alla *gestione dei rischi*
- introduzione alla *gestione delle configurazioni software*
- cenni su alcuni *sistemi di controllo di versione*

L'analisi dei rischi, come si è visto (Sez. 5.2.5), gioca un ruolo importante in modelli evolutivi del processo produttivo del software quali il ciclo di vita a spirale (v. Fig. 5.2), dove occupa una fase distinta in ciascuna iterazione del processo. D'altra parte, come anticipato in Sez. 6.4.1, la pianificazione del controllo del processo produttivo può prevedere meccanismi di controllo basati sull'analisi dei rischi. Questa lezione offre innanzitutto una succinta introduzione alla più generale tematica della *gestione dei rischi* , in cui l'analisi dei rischi si inquadra.
Valutare le possibili fonti di rischi e sottoporle ad un controllo costante e pianificato risulta conveniente nella produzione di siti Web per la rapidità dell'evoluzione del contesto in cui sono operanti. Si riscontrano dunque gli stessi vantaggi associati ad una attenta gestione dei rischi nella produzione di software secondo modelli evolutivi. I vantaggi essenziali, rivelati da molti studi su quest'ultima, sono: riduzione della probabilità di fallimento del progetto, riduzione dei ricicli nello sviluppo del prodotto, ottimizzazione dei costi e nell'uso delle risorse, soprattutto di quelle umane.
Il cambiamento rilevante alla gestione dei rischi è inerentemente del genere che *si subisce* , e dunque la sua gestione mira ad anticipare e risolvere i problemi che il suo manifestarsi comporta. Di sorta affatto diversa è il cambiamento inerente al processo produttivo, che dunque *si desidera* , in quanto manifestazione della *crescita e maturazione* del prodotto, dallo stadio iniziale di idea embrionale a quello di sistema (di servizi) in funzione, ed oltre ancora, attraverso la sua cosiddetta "manutenzione", di cui si è detto in precedenza (in Sez. 5.2.1).
La tematica della *gestione delle configurazioni* è sostanzialmente centrata sul problema della *continuità* , del prodotto o servizio in corso di sviluppo e/o manutenzione, *attraverso il cambiamento* . Afferiscono a questa tematica studi di metodi e strumenti per il supporto di: controllo di versione, versioni concorrenti e multiple, tracciamento degli errori, analisi delle dipendenze, integrazione e propagazione delle modifiche.
In questa lezione:

- **Gestione dei rischi** (Boehm 1989) **in due fasi:**
  - **Valutazione dei rischi** (Sez. 7.1)
    che comprende le attività di *identificazione, analisi* e *classificazione* dei rischi.
  - **Controllo dei rischi** (Sez. 7.2)
    che comprende le attività di *pianificazione del controllo, risoluzione* e *rilevamento* dei rischi.

- **Gestione delle configurazioni software** (Sez. 7.3)
  Si presenta una introduzione a problemi e metodi di gestione delle configurazioni, in cui si inizia con una definizione dell'oggetto di interesse, per estrarne gli aspetti più rilevanti. Si precisa quindi il concetto di *derivazione di prodotti* da una configurazione software, mentre si differiscono alla sezione successiva gli argomenti relativi al *controllo di versione* , che costituisce il nucleo centrale della tematica in questione, per trattarli nel contesto di rapidi cenni a strumenti di supporto a questo.
- **Sistemi di controllo di versione: RCS, CVS, SVN** (Sez. 7.4)
  I sistemi di controllo di versione di cui si dà cenno costituiscono essi stessi un esempio di progressiva evoluzione del software, nelle rispettive soluzioni offerte alle principali problematiche del controllo di versione, in particolare a quelle dell'*identificazione delle versioni* e dell'*accesso concorrente* .
- **Proposte di lavoro** (Sez. 7.5)
  La serie di proposte di lavoro avanzate nelle due lezioni precedenti prosegue con ulteriori proposte, che integrano le precedenti rispetto agli argomenti di questa lezione.

# 7.1 Valutazione dei rischi nel progetto del software

consta delle attività di

- **identificazione dei rischi**
- **analisi dei rischi**
- **classificazione dei rischi**

## 7.1.1 Identificazione dei rischi

**Tecniche di identificazione dei rischi**
Si incomincia con il far raccolta delle possibili fonti di rischi che potrebbero manifestarsi nello sviluppo del progetto di interesse. Risultano utili a questo scopo quattro tecniche di uso frequente.

- **Liste di verifica (checklist)**
  Sono dei promemoria di possibili rischi, compilati sulla base di esperienze pregresse nell'organizzazione produttiva o da esperti del settore applicativo in cui si colloca il prodotto.
- **Analisi dei criteri decisionali**
  Una fonte frequente di rischi si ha nel fatto che molte decisioni rilevanti al progetto possono essere prese da soggetti esterni al gruppo di sviluppo, e dettate da criteri decisionali estranei agli obiettivi propri del progetto, ad esempio criteri di tipo politico, o troppo orientati al mercato, o basati su obiettivi di breve termine e di corto respiro.
- **Analisi delle premesse culturali**
  Il bagaglio culturale delle persone coinvolte in un progetto ne condiziona le decisioni. L'analisi della loro formazione culturale e delle loro ipotesi di fondo può dunque rivelare possibili situazioni di rischio, specialmente con riguardo alle persone dotate di potere decisionale.
- **Supplemento di analisi**
  L'esperienza insegna che molti rischi che si manifestano nello sviluppo di un progetto hanno origine in "aree oscure" di documenti delle fasi iniziali, quali la specifica dei requisiti e il piano operativo. Un supplemento di analisi di tali documenti può rivelare potenziali fonti di rischi.

## 7.1.2 Analisi dei rischi

**Scopo**

- **valutare la probabilità che ciascuno dei rischi identificati si manifesti**
- **stimare la perdita causata in tal caso**

Il prodotto di questi due fattori è detto

**indice di esposizione al rischio ER(e) dell'evento e**
**ER(e) = P(e) * Perdita(e)**

dove il fattore `Perdita(e)` è espresso in termini monetari o in altra misura di risorse, ad esempio unità di tempo-persona, a cui sia associabile un costo.

**Tecniche di analisi dei rischi**

Pur scontando un indubbio margine di soggettività di giudizio e di incertezza nella stima di entrambi i fattori, gli indici ER risultano utili alla

- **Analisi delle decisioni**
  una delle tre principali tecniche di analisi dei rischi, che consiste nello stimare gli indici ER di decisioni alternative, e prendere quindi quella di indice minimo.
- **Analisi dei diagrammi PERT**
  (introdotti in Sez. 6.2.3) può essere condotta in vari modi, a seconda di quanta informazione è presente nel diagramma: nel caso più semplice, si identificano come fonti di maggiori rischi le attività più critiche, quali quelle il cui eventuale ritardo causa necessariamente ritardo nel completamento del progetto, o quelle con maggiore *fan-in* (numero di attività da cui dipendono direttamente) o *fan-out* (numero di attività che da esse dipendono direttamente).
- **Analisi what-if dei modelli di stima dei costi**
  quali i COCOMO (ai quali si è accennato in Sez. 6.1.4 e 6.2.5). Questa tecnica è basata sul fatto che i modelli di stima dei costi e dei tempi dei produzione dipendono da vari parametri della stima, secondo date formule empiriche: si può dunque studiare *che cosa* accade *se* detti parametri variano, e considerare più rischiosi gli eventi corrispondenti alle variazioni di parametri che causano i mutamenti più significativi dei costi e tempi stimati dal modello.

## 7.1.3 Classificazione dei rischi

**Scopo**

**disporre i rischi, precedentemente identificati, in ordine di rilevanza o criticità rispetto al conseguimento degli obiettivi del progetto**

I risultati dell'analisi dei rischi sono i dati di riferimento per stabilire quest'ordine, tuttavia, a causa del margine di incertezza ad essi associato, detti risultati solitamente sono solo il punto di partenza per la classificazione, che tiene conto del giudizio di esperti e delle valutazioni basate su

esperienza e previsioni dei protagonisti del processo produttivo.

# 7.2 Controllo dei rischi nella produzione del software

consta delle attività di

- **pianificazione del controllo dei rischi**
- **risoluzione dei rischi**
- **rilevamento dei rischi**

## 7.2.1 Pianificazione del controllo dei rischi

**Scopo**

**produrre un piano di intervento e di controllo per rilevare tempestivamente e fronteggiare l'eventuale manifestarsi, nel corso del progetto, di ciascuno dei rischi precedentemente studiati e classificati**

**coefficiente di riduzione del rischio CRR(t,e)**
**della tecnica t di riduzione del rischio per l'evento e**
**CRR(t,e) = (ER(e) - $ER_{dopo}$ (t,e)) / C(t)**

- **Analisi costi-benefici**
  In primo luogo, in base alla classificazione risultante dalla fase precedente, occorre decidere di quali rischi cercare di ridurre la probabilità che si verifichino e, per ciascuno di essi, con quale *tecnica di riduzione del rischio* .
  A questo scopo torna utile il concetto di *coefficiente di riduzione del rischio* `CRR(t,e)` (v. riquadro sopra), che esprime il rapporto beneficio/costo per l'adozione di una data tecnica `t` di riduzione del rischio dell'evento `e`, dove
  **beneficio:** i due indici di esposizione al rischio che compaiono al numeratore sono rispettivamente riferiti alla situazione prima dell'adozione della tecnica `t` e a quella dopo l'adozione di detta tecnica, mentre
  **costo:** `C(t)` è il costo di adozione della tecnica in questione (ovviamente espresso nella stessa unità di costo in cui si esprimono le perdite associate all'evento e nei due indici di esposizione al rischio in questione).
- **Riclassificazione dei rischi**
  In secondo luogo, avendo deciso quali tecniche di riduzione del rischio adottare, si produce una *nuova classificazione dei rischi* che tenga conto dei mutati indici `ER` , in virtù della riduzione di probabilità dei rischi per i quali si sono adottate le suddette tecniche.
- **Integrazione nel piano operativo**
  Infine, si pianificano le attività di rilevamento e risoluzione dei rischi di cui appresso, e si procede alla loro integrazione nel piano operativo.

## 7.2.2 Risoluzione dei rischi

**Scopo**

**eseguire le attività di intervento previste dal piano di controllo dei rischi:**

- **preventive**
- **in itinere**

per far fronte ai rischi studiati. Possono aver luogo come attività preliminari all'effettivo sviluppo del prodotto, come ad esempio accade per attività formative previste per la risoluzione preventiva di rischi di insufficiente conoscenza di specifiche tecnologie, o possono aver luogo durante lo sviluppo, nel caso che il rischio paventato si manifesti effettivamente.

## 7.2.3 Rilevamento dei rischi

**Scopo**

**eseguire le attività di osservazione, misura e monitoring previste dal piano di controllo dei rischi**

Tali attività hanno evidentemente luogo durante il processo di sviluppo del prodotto. Una fra esse consiste nel mantenere una *tabella di monitoraggio* o

**hit list dei rischi**

che effettivamente si manifestano, classificati in ordine di gravità, anzianità di presenza nella tabella, etc., con ulteriori informazioni sullo stato di ciascun rischio, ovvero delle misure intraprese per fronteggiarlo.

## 7.2.4 Struttura organizzativa della gestione dei rischi

Una possibile struttura organizzativa per la gestione dei rischi è basata su
**tre ruoli** (Boehm 1989) **:**

- **Responsabile**
  ha i compiti di: pianificazione delle attività di gestione dei rischi, verifica della loro corretta attuazione, integrazione di dette attività nel ciclo di vita del software.
- **Gruppo di controllo**
  è diretto dal responsabile della gestione dei rischi, e vi partecipano i responsabili degli altri gruppi del progetto; si riunisce periodicamente per valutare i risultati dell'applicazione delle tecniche di risoluzione dei rischi ed aggiornare la tabella di monitoraggio dei rischi.
- **Gruppo di lavoro**
  è costituito dai responsabili del progetto e da rappresentanti degli utenti (o del committente), ed ha il compito di valutare l'impatto dei rischi identificati e delle relative tecniche di controllo e risoluzione sulle attività di sviluppo.

# 7.3 Gestione delle configurazioni del software

**concetti, problemi, metodi e strumenti rilevanti alla manutenzione del software, ovvero alla sua evoluzione nel ciclo di vita**

## 7.3.1 Concetto di configurazione software

In prima approssimazione:

**l'insieme dei moduli software e di informazione ad essi correlata (specifiche, documentazione di progetto, di collaudo, di utente, etc.) che consente di costruire una versione funzionante di un prodotto software, includendo in questa categoria anche le applicazioni Web.**

Il concetto di configurazione così caratterizzato risulta evidentemente di interesse per prodotti software di una certa complessità, per i quali cioè la procedura di costruzione non si riduca alla semplice traduzione di un programma sorgente nel corrispondente codice eseguibile, ma richieda una maggiore articolazione in operazioni quali: consultazione di documentazione, determinazione dell'insieme di moduli sorgenti (ad es. da estrarre da una libreria) necessari alla costruzione, determinazione dei traduttori richiesti e delle rispettive direttive di compilazione, generazione dei corrispondenti moduli oggetto, costruzione dell'eseguibile da questi ultimi (previa determinazione di relative direttive), eventuale collaudo del funzionamento del prodotto.

## 7.3.2 Definizione di gestione delle configurazioni

A fronte di questo concetto di configurazione, una definizione di *gestione delle configurazioni* è **in prima approssimazione, proposta da** (Babich 1986) **:**

**insieme delle tecniche e delle metodologie che consentono di identificare, organizzare e controllare le modifiche apportate ad un programma sviluppato da un gruppo di programmatori, allo scopo di ottimizzare la produttività e ridurre il numero di errori.**

Inerenti alla definizione proposta sono dunque i seguenti aspetti.

1. **Modifiche del software**
   Come abbiamo anticipato all'inizio della lezione, la gestione delle configurazioni è centrata sul problema della continuità del software attraverso il cambiamento. La definizione data limita lo scopo del cambiamento alla manutenzione correttiva, e quello della gestione delle configurazioni al corrispondente risultato (riduzione degli errori) con ottimizzazione della produttività, ma la pratica degli anni più recenti rivela ampiamente l'applicabilità di metodi di gestione delle configurazioni alla manutenzione adattativa e perfettiva, e la sua conseguente utilità ad altri obiettivi di qualità che trascendono quello proposto, segnatamente

   

   **conseguimento della massima soddisfazione degli utenti**

2. **Coordinamento del lavoro di un gruppo di produttori**
   considerati come autori sia di sorgenti originali sia (e soprattutto) di successive modifiche. Frequentemente si riconosce proprio in un coordinamento insoddisfacente l'origine di gravi inconvenienti quali incapacità di gestire correttamente lo *sviluppo concorrente* di diverse parti del software da diversi autori, o di controllare l'evoluzione del prodotto a fronte di manutenzione correttiva e perfettiva.
3. **Identificazione delle modifiche apportate**
   qui intesa in un senso "sintattico" di *meccanismo di denominazione univoca.* La necessità di un tale meccanismo è tanto elementare quanto lo è quella di designare univocamente l'oggetto di qualsiasi discorso.

## 7.3.3 Derivazione di prodotti da una configurazione software

**Raramente la realizzazione di un prodotto software parte dal nulla.**
Non solo, ad es. nelle metodologie evolutive, può essere disponibile un prototipo, magari inizialmente sviluppato solo a scopo di validazione dei requisiti di utente, ma generalmente:
**Sempre più spesso risulta disponibile codice in gran quantità, riusabile per fornire funzionalità diverse da quelle originarie, a fronte di limitate modifiche.**
Una elevata disponibilità di software riusabile può essere frutto di lungimiranti strategie dell'organizzazione produttiva, ad esempio quando, come regola generale, si privilegia la costruzione di librerie software ad alto grado di riusabilità rispetto a soluzioni *ad hoc* , ma può anche beneficiare del crescente patrimonio di software *open source* nel pubblico dominio, quando le politiche dell'organizzazione produttiva rispetto alla questione delle licenze d'uso del software, o le caratteristiche della distribuzione del prodotto, siano compatibili con l'inclusione di componenti open source nel prodotto stesso.
Quali che siano le ragioni che permettono un elevato riuso di software già esistente nella realizzazione di un nuovo prodotto, alla evidente convenienza economica di questo processo si accompagna però il

**rischio:**

importare nel prodotto eventuali difetti nascosti nel software preesistente

**Problemi**
di maggior rilievo alla tematica qui in considerazione sono quelli di come garantire:

1. **tracciamento degli errori**
   riscontrati in un nuovo prodotto, ma causato da difetti o errori di adattamento di software preesistente, riusato nel prodotto;
2. **determinazione delle dipendenze fra moduli nuovi e moduli riusati**
   ai fini dell'analisi delle stesse, quale strumento di tracciamento degli errori di cui al punto precedente;
3. **completezza della manutenzione correttiva del software riusato**
   integrazione e propagazione delle modifiche di manutenzione correttiva a software riusabile, in *tutte* le istanze del suo riuso.

**Esauriente documentazione della derivazione del prodotto**
è la chiave di volta su cui si reggono le soluzioni ai problemi suddetti. Con tale termine si designa

**derivazione di un prodotto**

**insieme di informazioni che permette di ricostruire la storia della creazione ed evoluzione del prodotto**

**ad esempio:**

- **l'elenco delle configurazioni relative alle sue successive versioni, e**
- **per ciascuna di queste:**
  - **procedure utilizzate per creare gli oggetti derivati (moduli oggetto ed eseguibili)**
  - **autori delle creazioni**
  - **date di creazione**
  - **scopo della creazione**
  - **strumenti e documentazione di eventuali collaudi**
  - **etc.**

# 7.4 Sistemi di controllo di versione

**Motivazioni**

**Distinguere versioni diverse di uno stesso documento, modulo software o prodotto, risulta utile per due categorie di motivazioni affatto diverse:**

- **per distinguerlo in stadi diversi di sviluppo o manutenzione**
- **per distinguerne forme diverse in quanto adatte a contesti diversi**

Esempi della prima categoria si riscontrano naturalmente nell'evoluzione sia dei semilavorati durante il processo di sviluppo, sia dei prodotti a seguito di manutenzione correttiva e perfettiva.
Riguardo alla seconda categoria, uno stesso documento di testo, ad esempio, può essere disponibile in formati di visualizzazione o stampa diversi, ciascuno adatto alle specifiche caratteristiche di un determinato programma di visualizzazione o stampa, rispettivamente. Un prodotto software potrà contenere parti che sono indipendenti dall'architettura hardware e parti che ne dipendono; ad esempio, un modulo di emulazione software di operazioni aritmetiche complesse, superfluo nelle architetture in cui tali operazioni sono eseguite da un coprocessore aritmetico, essenziale nelle altre.

## 7.4.1 Concetti di versione

Dalle motivazioni precedentemente esposte emergono

**due casi del concetto di versione**

1. *revisione* :
   **un modulo è una revisione di un altro modulo se lo sostituisce in tutte le nuove configurazioni create a partire dal momento in cui esso è reso disponibile; questo caso corrisponde all'evoluzione correttiva o perfettiva del modulo;**

2. *variazione* :
   **un modulo è una variazione di un altro modulo se, a partire dal dal momento della sua disponibilità, lo sostituisce solo in alcune delle nuove configurazioni, quelle cioè adatte a contesti in cui siano verificate le condizioni specifiche per le quali è realizzata la variazione.**

**Concetti rispettivamente correlati:**

1. **Versioni concorrenti**
   Il primo caso è quello di maggior interesse all'evoluzione del software. Tutti i sistemi di controllo di versione dei quali si dà cenno più avanti permettono lo *sviluppo concorrente,* in modi diversi, ma questo presenta problemi di particolare rilievo quando le versioni concorrenti sono, per l'appunto, revisioni operate in parallelo da autori diversi. Tali sistemi forniscono, fra l'altro:
   - **meccanismi di identificazione univoca delle revisioni**
   - **memorizzazione incrementale delle revisioni**
     basata sulla memorizzazione delle differenze tra revisioni consecutive, per l'efficienza d'uso delle risorse di memoria
   - **opportunità di documentare le modifiche**
     che ciascuna revisione presenta rispetto alla versione precedente
   - **meccanismi di blocco (ingl.** *lock* **)**
     per evitare modifiche simultanee di uno stesso modulo da parte di autori diversi. Di quest'ultimo aspetto ci occupiamo nella sezione che segue.
2. **Famiglia di prodotti: insieme di prodotti ottenibili da configurazioni che differiscono solo per variazioni dei moduli in essi.**
   Sembra evidente la necessità di gestire una famiglia di prodotti in modo tale che l'eventuale revisione di un modulo che non ha variazioni si applichi consistentemente a tutti i prodotti nella famiglia. Ciò si ottiene organizzando i moduli in una libreria centralizzata dove ciascuno di essi è univocamente identificato. La revisione dei moduli viene operata sulla libreria, mentre ciascun prodotto viene costruito estraendo i moduli della sua configurazione dalla libreria.

## 7.4.2 Controllo di versione nello sviluppo concorrente

**Problema: in assenza di un sistema di controllo di versione con un qualche meccanismo di blocco, l'accesso concorrente ad un'area condivisa, in cui autori diversi producono revisioni di documenti o moduli software, può generare errori nel risultato o perdita di modifiche.**

Il primo caso si ha quando le modifiche vengono effettuate direttamente sull'area condivisa, a causa dell'interferenza di modifiche su uno stesso modulo, in cui ciascuna delle due non tiene conto dell'altra.

**Soluzione:** *sandbox + lock*

La soluzione al problema precedente consiste nel prevedere un'area privata per ciascun autore, il quale opera le modifiche su una *copia locale* del modulo, prelevata dall'area (o libreria) condivisa. Solo quando un autore avrà completato la revisione nella sua area privata (ingl. *sandbox,* nella terminologia dei sistemi qui in esame), potrà accedere all'area condivisa per sostituire il modulo con la revisione prodotta.
Se sul modulo nell'area condivisa è attivato un *lock* (più precisamente, uno *strict lock* ) per tutta la durata del lavoro di revisione, non c'è possibilità di conflitto ed il problema è risolto. Ciò però comporta, di fatto, l'impossibilità di parallelizzare il lavoro su uno stesso modulo, anche quando due o più autori convengano di operare su parti distinte e separate del modulo.

*strict lock* **o** *soft lock* **?**

Il meccanismo di *soft lock,* in alternativa allo *strict lock,* è più tollerante: segnala che del lavoro di modifica è in corso su (una copia locale de)l modulo, ma non ne blocca il trasferimento in altre aree private. È dunque possibile concordare di operare simultaneamente modifiche su parti separate dello stesso modulo. A tale scopo occorre dunque definire opportune norme d'uso del sistema, e garantirne il rispetto, affinché l'accesso concorrente all'area o libreria comune proceda in modo coordinato e coerente.
Ciascuna delle due o più revisioni così effettuate contiene solo le modifiche apportate dal suo autore: l'integrazione di modifiche effettuate in parallelo da più autori deve essere effettuata manualmente. Poiché la revisione di un documento non cancella fisicamente la versione precedente, risulta possibile effettuare revisioni parallele come indicato, per poi procedere all'integrazione quando tutte le revisioni parallele saranno state compiute. A tale scopo, i sistemi di controllo di versione qui in esame offrono l'operazione di *merge* , che effettua l'integrazione di modifiche su documenti di testo effettuate in revisioni differenti e su parti separate del documento. Ne introduciamo brevemente le caratteristiche principali.

## 7.4.3 Il sistema RCS

**Revision Control System** (RCS)

- **memorizzazione incrementale inversa**
- **blocco per default**
- **integrazione su richiesta esplicita**

Il sistema RCS nasce nella prima metà degli anni '80 in ambiente Unix come evoluzione del *Source Code Control System* (SCCS). Successive variazioni di RCS lo hanno reso disponibile anche per altri sistemi operativi. I principali miglioramenti riguardano l'interfaccia di utente e l'efficienza d'uso delle risorse di memoria per un più rapido reperimento delle versioni più recenti. Quest'ultimo progresso è ottenuto memorizzando integralmente la versione più recente, mentre per le versioni precedenti si memorizzano le differenze in ordine inverso (ingl. *reverse delta* ), cioè le differenze che ciascuna versione presenta rispetto a quella successiva.
Le funzioni di RCS si applicano a librerie di file di testo. A ciascun modulo nella libreria è associato un *albero delle revisioni* , nel quale diramazioni da uno stesso nodo rappresentano linee di sviluppo concorrente. Un meccanismo di *lock* regola l'accesso concorrente, secondo una politica di blocco per default, mentre una richiesta di *soft lock* deve essere esplicita.

Coerentemente con la suddetta politica, l'integrazione di più linee concorrenti di sviluppo è resa possibile attraverso richiesta esplicita di un'operazione di *merge.*
RCS fornisce inoltre funzioni di identificazione automatica delle revisioni, con registrazione automatica di informazioni utili alla derivazione di prodotti (ad es. autore, data, stato della revisione, etc.), e permette di esaminare le differenze tra revisioni diverse.

## 7.4.4 Il sistema CVS

**Concurrent Versions System** (CVS)

- **gestione delle configurazioni in progetti software,** *repositories* **CVS in rete**
- **integrazione per default**
- **blocco su richiesta esplicita**

Il sistema (CVS) (*Concurrent Versions System* ) costituisce una ulteriore e più sofisticata evoluzione di RCS, ed è oggi lo strumento più diffuso per la gestione delle configurazioni in progetti di sviluppo di software *open source.* La diffusione della collaborazione in rete ha portato all'affermazione di CVS come sistema *client-server* e allo sviluppo sia di software *client* sia di interfacce Web per l'esplorazione di *repositories* CVS.
Le funzionalità essenziali di CVS sono le stesse di RCS, però si abbandona l'approccio "blocco per default" in favore del più "ottimistico" approccio caratterizzabile come "integrazione per default". Cioè, al fine di operare una revisione, non è più necessario ottenere un *lock,* e in caso di modifiche concorrenti, il sistema proverà ad effettuarne il *merge,* senza necessità di intervento esplicito degli autori a questo scopo. Naturalmente, quando serva, è possibile porre uno *strict lock* sulla revisione in corso, nel qual caso l'integrazione con altre revisioni della stessa versione potrà effettuarsi attraverso una richiesta esplicita di *merge,* come in RCS.

## 7.4.5 Il sistema Subversion

Proposto in continuità evolutiva rispetto a CVS, il sistema *Subversion* (SVN) ne propone una serie di miglioramenti, volti al superamento di alcune limitazioni rivelate dall'esperienza pratica d'uso di CVS. Il riquadro presenta un sommario di alcune delle principali innovazioni. Per una panoramica più completa si rinvia alla fonte citata.

**Subversion** (SVN)

- **controllo di versione esteso a directory, copie, ridenominazioni e meta-dati**
- **atomicità del** *commit* **delle modifiche (identificazione delle revisioni per** *commit,* **invece che per file)**
- **servizio in rete con Web-server** *Apache* **e supporto del protocollo** *WebDAV/DeltaV* **basato su HTTP (ma con opzione di server indipendente, se desiderata)**
- **architettura** *client-server* **nativa**
- **controllo di versione di file di qualsiasi tipo, non solo testo, senza perdita di efficienza (algoritmo di analisi delle differenze di file binari)**

# 7.5 Esercizi: gestione dei rischi, gestione delle configurazioni

**La proposta di lavoro sugli argomenti introdotti in questa lezione è in continuità con quelle delle due lezioni precedenti**

## 7.5.1 Premessa

ipotesi di lavoro: v. Sez. 5.3.1

## 7.5.2 Esercizio 1: identificazione dei rischi

Identificare i rischi più rilevanti, secondo il proprio giudizio, ed argomentarne la rilevanza.

## 7.5.3 Esercizio 2: tecniche di riduzione del rischio

Per ciascuno dei rischi identificati nell'esercizio precedente, descrivere una o più tecniche di riduzione del rischio (quando almeno una tale tecnica esista).

## 7.5.4 Esercizio 3: pianificazione del controllo dei rischi

A partire dalle risposte date ai due esercizi precedenti, elaborare un piano di gestione dei rischi ed integrarlo nella bozza di piano operativo prodotta nella risposta all'esercizio 5 in Sez. 6.5.6.

## 7.5.5 Esercizio 4: derivazione di prodotti

Nell'ipotesi che versioni successive della documentazione contenuta nel sito, di cui all'ipotesi di lavoro, siano rese disponibili in forma di archivio compresso, posto sotto controllo di versione, dove l'ultima versione sia per ipotesi l'archivio compresso della directory con le ultime versioni dei sorgenti del sito, proporre norme di documentazione, delle modifiche di ciascuna revisione dell'archivio rispetto alla versione precedente nonché delle modifiche di ciascuna revisione dei sorgenti, utili alla *derivazione* del sito (in un senso analogo a quello dato a tale termine per prodotti software in Sez. 7.3.3).

### 7.5.6 Esercizio 5: controllo di versione nello sviluppo concorrente

Nell'ipotesi enunciata nell'esercizio precedente, proporre norme di uso del meccanismo di *lock* di uno dei sistemi di controllo di versione introdotti in Sez. 7.4 che consentano la revisione in parallelo di documenti distinti dell'archivio da parte di più autori.

### 7.5.7 Esercizio 6: piano di gestione delle configurazioni

A partire dalle risposte date ai due esercizi precedenti, elaborare un piano di gestione delle configurazioni ed integrarlo nella bozza di piano operativo prodotta in risposta all'esercizio 5 in Sez. 6.5.6.

## 7.6 Note

1. Fai raccolta dei soliti sospetti.

## 7.7 Bibliografia


- **Babich, W.A..**, 1986. *Software Configuration Management: Coordination for Team Productivity*. Reading, MA, USA: Addison-Wesley.
- **Boehm, B.**, 1989. *Tutorial: Software Risk Management*. Washington D.C., USA: IEEE Computer Society Press.
- **RCS**. *Revision Control System*. Web: http://www.cs.purdue.edu/homes/trinkle/RCS.
- **CVS**. *Concurrent Versions System*. Web: http://ximbiot.com/cvs.
- **SVN**. *Subversion project*. Web: http://subversion.tigris.org.