# Lezione #9

Mercoledì, 23 Ottobre 2013

*Tullio Vardanega, 11:30-13:15*

**Luca De Franceschi**

**Configurazione**, devo conoscere tutte le parti di un sistema, **C**onfiguration **M**anagement **D**atabase. Spesso nei team c'è una persona che si occupa della configurazione. In un progetto, tipicamente, l'albero della base di dati è molto complesso, nel nostro progetto è molto più semplice in realtà. Elementi di configurazione (**CI**, configuration items). In un sistema reale la base di dati è essenziale, perchè devo poter interrogarlo sui vari item. La configurazione è fatta di nodi che evolvono, perché ciascun nodo ha una storia ed essa viene modificata. Prima ho la configurazione (*system breakdown*, decomposizione del sistema). Gli elementi in questo albero sono i CI. Alcuni nodi costituiscono un raggruppamento che descrive un preciso stato del sistema (*baseline*) che rappresenta il "punto di copertura" di uno stadio da parte di un team. Avrò un numero non fisso di baseline, servono a darmi certezza di progresso. La prima certezza la voglio sui requisiti (*requirements baseline*). Una baseline è una raccolta di CI, ciascuna con la sua storia. Ho bisogno di uno strumento che mi dica da quali CI è costituita una baseline. Conseguentemente, una baseline va documentata. Esiste dunque un'attività aggiuntiva che si chiama **controllo di configurazione**. Questa attività è molto delicata perchè mi dice lo stato di avanzamento del prodotto. La modifica è un atto che deve essere formalmente approvato, perchè è un processo sui cui vogliamo iterare. Occorre introdurre un sistema rigido di gestione delle modifiche che vengano fatte a fronte di una **richiesta**. Si cambia la baseline cambiando una o più CI che cambiano la loro versione. Chi vuole cambiare deve avere un motivo per cui valga la pena, quindi serve un documento che va ad un'autorità delegata, per capire se il cambiamento è accettabile o no (autorità che si chiama "gruppo di decisione", *Change Management Board*, CMB). Questo gruppo valuta se il cambiamento è accettabile e lancia un ticket per il cambiamento. Chi fa queste azioni reimmette nel sistema il dato modificato. Fare un cambiamento produce un flusso di lavoro, e quindi comporta un costo. Normalmente il responsabile di progetto è parte di questo gruppo. La configurazione evolve attraverso il change management.

Il **versionamento** deve essere automatizzato. Si appoggia su una *repository*, che è un servizio aggiornato alla *best practice*. Devo usare un sistema di versionamento già esistente e moderno, supportato dalla tecnologia "*cloud*" (es. Git). Voglio che il versionamento e la configurazione siano insieme. Le modifiche le faccio sullo spazio locale e poi le reimmetto nella repository quando sono convinto che quello che ho fatto è buono, facendo un *update*. Deve essere funzionante in modo robusto al modo concorrente (*data race*). Questa cosa si risolve in due modi:

- **Locking**;
- procedura che assicura un invariante, che su un singolo CI lavori in ogni istante al più una persona. Applico questo invariante sui tickets.

Ogni singolo CI ha quindi una storia ramificata che non è necessariamente simmetrica con la versione del sistema. La storie possono essere lineari o ramificate quando ho delle varianti in cui ho più requisiti in contrasto. Il **branching** è una diramazione rispetto al **trunk** (tronco) principale. Devo essere capace di distinguere ciò che c'è nella diramazione da ciò che c'è sul sistema. No solo nel codice ci devono essere le modifiche ma anche nella repository devo avere un *log*. Nell'oggetto cambiato c'è scritto cosa e cambiato, quando e perchè.
**Strumenti di lavoro**, molto aldilà del semplice editor di programmi e compilatore...
**Procedure di progetti**, le attività sono sistematiche, disciplinate e quantificabili. Ciò che dà disciplina alle attività sono le norme sancite. Sono le regole che di diamo, e non è accettabile che di defletta da queste. Le norme sono l'essenza, salvaguardano la collaborazione. In un gruppo di lavoro mi devo affidare ad un modello rigido che si chiama appunto *norme*. Si distingue in due categorie:

- **Obbligatorie**;
- **Raccomandazioni**.

Una regola non si discute, una raccomandazione è invece opinabile. Le norme vanno scritte in un documento. Un documento che il gruppo dovrà avere sarà le "*Norme di progetto*", un insieme di regole a cui il gruppo

deve aderire. E' il fattore umano che rende le regole difficili e permeabili. Costruire questo filtro comporta un costo e una fatica iniziale, ma porta a miglioramenti poi. Le regole sono buone se sono facilmente applicabili e non vanno lasciate a discrezione degli esseri umani. Ci sono alcune pratiche di buonsenso per evitare errori:

- Nomi di tipi, di variabili e funzioni devono essere espressivi e distinguere di cosa stiamo parlando (es. norme *Javadoc*);
- Quanto codice posso scrivere in un solo file, strutturazione in moduli, file, directory... Il codice va frantumato in parti piccole, che consentano di rispettare un vincolo di linee di codice per file;
- Regolazione rispetto a come uso un linguaggio. Il programmatore non può prendersi la libertà di utilizzare come vuole un linguaggio.

La parte più importante di un pezzo di codice si chiama **modulo**. Questa cosa è l'intestazione (o *header*) che deve dirmi tutto di quel modulo (cosa fa, chi l'ha fatto, copyright, copyleft, avvertenze...). Serve una strategia forte per costringere i programmatori a lavorare come si conviene.

L'attività di **analisi dei requisiti** è importantissima, comporta molte competenze ed è complessa, per cui viene chiamata **Ingegneria dei requisiti**. Un requisito può essere visto da due lati:

- Dal lato di chi lo chiede (**vista cliente**);
- Dal lato di chi lo soddisfa (**Vista fornitore**)

Dovrò prima interrogare gli stakeholders e poi chiedermi cosa serve a me per soddisfare quei requisiti. Quando queste due cose sono soddisfatte avrò soddisfatto il **requisito utente**. Sui requisiti si effettua *breakdown*, spezzamento, perchè rende più facile verificare che siano soddisfatti.

- Assicurare che siano individualmente verificabili;
- Assicurare che siano individualmente effettuabili.

La **verifica** e la **validazione** formano la **qualifica**.

- La verifica va a controllare che nel fare il lavoro non abbiamo introdotto errori (*did I build the system right?*);
- La validazione va ad accertarsi che i requisiti siano soddisfatti (*did I build the right system?*).