Progetto del Software

Appunti redatti

Iacopo Ruzzier

Ultimo aggiornamento: 3 marzo 2025

Indice

Introduzione

Il corso tratta metodologie di progetto del software, diversificate per le varie necessità e solitamente riconducibili a poche famiglie → strutturazione e automatizzazione del processo

vedremo

- documentazione (su sw di larga scala si lavora in molti) - come usare, modificare, testare, rilasciare i nostri artefatti sw; licensing
- requisiti, KPI, test-driven design
- tool di collaborazione
- design tools (UML, E/R d., ...)
- design pattern
- strutturazione codice (programmazione avanzata)
- alcuni use-cases

Parte I
Collaborative tools - Git & friends

1 VCS, git

vcs: sistema che traccia le modifiche fatte ad un progetto e permette di ritornare a stati precedenti

- permette lo sviluppo collaborativo (modifiche "firmate")
- obbliga a seguire flussi di sviluppo noti
- fornisce set di strumenti per automatizzare testing, integrazione, sviluppo (CI/CD)
- modo semplice per scrivere documentazione
- permette di concentrarsi sulla scrittura del codice "senza pensieri"

1.1 git

vcs più usato, basato su repo distribuiti (2005 torvalds, per supportare sviluppo kernel linux): a partire da repo remote contenenti la codebase ("origin") gli utenti clonano la repo in locale, la mantengono aggiornata tramite pull, la aggiornano tramite push

2 Workflow base

progetto come **sequenza di commit**: snapshot del codice in un dato istante, con commit identificati da hashcode, contenenti riferimento al commit precedente, e commento obbligatorio
→ i commit tracciano cambiamenti incrementali della codebase, con granularità a discrezione dei programmatori

- prima di un commit devo aggiungere i file nella staging area
- rinominare file significa elimino+riaggiungo

```
git clone <URL>
git log #log di tutti i commit fatti
git diff #per visualizzare le differenze tra due commit
git add <file>[<file>...] #staging
git commit # -m "msg"
git commit -a # bypassa staging, ma non considera i file appena aggiunti
              # non ancora tracciati
git commit --amend # ritorno al commit precedente se ho dimenticato
   qualcosa
git push [origin] [master]
git pull [origin] [master]
```

IMG schema git

pulling e auto-merging: la storia locale è aggiornata in base al timestamp del commit. in particolare viene modificata automaticamente includendo sia i cambiamenti locali che quelli remoti (merge in automatico)

2.1 Merge e conflitti

git lavora sulla singola riga → conflitto se viene modificata da più utenti diversi, con repo locale che resta in stato conflicting in caso di **merge conflicts** → vanno risolti localmente e va fatto il merge manuale (flag appropriati)

consigli:

- pull frequenti
- assicurarsi che il codice funzioni (testing automations)
- commit piccoli, suddivisi per area di lavoro (codice, makefile, script) → forza a mantenere lo spazio di lavoro pulito e strutturato

```
git checkout/reset # unstaging/deletion modifiche o commit locali
git revert # per ritornare ad un commit specifico
git cherry-pick # essendo i commit incrementali (tracciano la
# differenza rispetto al genitore), possiamo applicare la stessa
# logica ma rispetto ad un commit diverso
```

3 Struttura tipica di un progetto con versioning

tipicamente molto rigida

- branch principale contenente ultima versione rilasciata (e intera cronologia commit)
- branch multiple corrispondenti a sottoprogetti specifici
- libertà totale sulle branch, tipicam. regole aziendali (`develop`,`bugfix/`,`hotfix/`,`features/`,`.pb_<smth>`)
- push su main branch non permesso → fork di main o dev, e poi PR (gestita dal maintainer)
- regole di accesso e vari ruoli utente (a liv. repo e branch)

flow tipico:

1. dev clona una branch della repo remota aggiornata
2. dev inizia a lavorare, nuovi commit in lcoale, nel mentre commit nuovi anche in remoto
3. una volta pronto, il dev fa una pull da remoto, con la responsabilità di rendere consistenti i propri commit con la cronologia principale (implica retesting)
4. dopo il merge, si crea il "final commit", si pusha sul cloud e si fa una PR
5. (tipicamente) la richiesta viene accettata, e le modifiche sono applicate alla branch `Developer`
6. ultimo pull per rendere consistente la copia locale, e (tipicamente) eliminiamo l'altra branch