Indice

1.Introduzione

1.1 Scopo

Lo scopo di questo documento è quello di definire attività, responsabilità e procedure relative alla gestione della configurazione del software (SCM) per il progetto *nome_progetto*. In particolare, il documento descrive le modalità di utilizzo della piattaforma **Github** come sistema di versionamento, controlle delle modifiche e collaborazione tra i membri del team.

1.2 Ambito

Il piano SCM si applica a tutti i componenti software sviluppati nel progetto, inclusi codice sorgente, documentazione e qualsiasi altro file.

1.3 Riferimenti

- Documentazione Github Ufficiale: https://docs.github.com/en

1.4 Definizioni e Acronimi

- SCM: Software Configuration Management.
- CI/CD: Continuous Integration / Continuous Deployment ● Branch: linea di sviluppo separate nel repository Git.
- Issue: Metodo per notificare ed organizzarsi con i colleghi su come risolvere eventuali problemi o dare nuove idee che poi possono essere implementate.
- PR (Pull Request): richiesta, post analisi di un issue ed eventuale implementazione per risolverlo, di fondere le modifiche apportate con il branch “precedente” (creando un branch *issue* da *mio_branch*, quando eseguo PR essa si fonde in *mio_branch*).
- Kanban board: tabella interattiva per organizzare e gestire al meglio il carico di lavoro

2.Gestione SCM

2.1 Organizzazione

Il nostro team di sviluppo è composto da 4 membri che devono collaborare tramite la repository posta sulla piattaforma Github, su cui è centralizzata la gestione della configurazione del progetto.

2.2 Ruoli e Responsabilità

- Configuration Manager: (Vitali Michele) responsabile dell'approvazione delle modifiche e della pubblicazione delle versioni ufficiali. Supervisiona l'applicazione delle regole SCM.
- Sviluppatori: (Tutti) realizzano le feature proposte creando branch dedicati e sottomettendo PR.
- Revisori: (Tutti) analizzano le PR prima di effettuare un merge nel branch principale. Per un fatto di sicurezza e qualità del codice è stato scelto che per eseguire un merge tramite PR si deve avere l'approvazione di almeno altri 2 membri del gruppo.

2.3 Autorità

Sempre per garantire stabilità nelle versioni è stato scelto che il merge sul branch main è riservato solo al Configuration Manager. Allo stesso tempo però ogni membro del gruppo può ovviamente creare branch e sottomettere PR che, dopo attenta revisione, potranno essere accorpate al branch di sviluppo denominato ***dev***.

3. Attività SCM

3.1 Identificazione degli Elementi di Configurazione

Gli elementi sottoposti a controllo di configurazione sono:

- Codice sorgente (directory **/src**)
- Documentazione (directory **/docs**)
- File di configurazione (directory **/config**)

Inoltre quando si deve creare un nuovo branch per implementare qualcosa si deve seguire questa convenzione:

- **feature/*nome_feature*** , per nuove feature.
- **bugfix/*nome_bug*** , per correzioni di eventuali problemi/bug trovati nel codice.
- **hotfix/*nome***, per correzioni urgenti (poco usato).

Infine le versioni considerate stabili e fruibili agli utenti verranno codificate con il formato **vX.Y.Z** (ad esempio v1.0.0), dove **X** indica la versione di base mentre **Y** e **Z** servono per indicare che nella versione **X** sono stati eseguiti dei fix/migliorie del codice.

3.2 Controllo di Configurazione

Qualsiasi modifica che si voglia accorpare alla versione attuale deve essere innanzitutto effettuata tramite branch dedicato seguendo le linee guida appena descritte per poi essere approvata tramite PR.
Come già detto le PR devono essere approvate da almeno 2 membri del gruppo (oltre lo stesso che l'ha creata) prima del merge.
I merge devono essere eseguiti nel branch ***dev*** e solo dopo un altro stadio di testing/revisione il **Configuration Manager** eseguirà il merge nel branch **main** qualora la versione venisse considerata stabile e distribuibile.

3.3 Contabilità dello Stato

Ogni nuova versione che verrà rilasciata al pubblico deve essere accompagnata da una **Release Note** nel repository di Github dove si descrive nel dettaglio:

- **Elenco delle modifiche**
- **Eventuali bug risolti ● Nuove funzionalità**

3.4 Audit di Configurazione

A fine sprint verranno condotti degli audit interni per verificare:

- **Coerenza tra branch e naming convention**
- **Tracciabilità PR**
- **Conformità alle linee guida**

4. Orari SCM

1. L'identificazione e tagging delle versioni avverrà al termine di ogni sprint.
2. Le PR devono essere chiuse entro la fine dello sprint corrente.
3. Gli audit SCM si terranno ogni circa 2 settimane, in concomitanza con la revisione dello stato di avanzamento del progetto ed eventuale pianificazione e organizzazione di nuove feature/fix.
4. I release di versioni stabili avverranno ogni qualvolta si raggiungerà una milestone del progetto precedentemente definita.

5. Risorse SCM

Come strumenti verranno usati principalmente:

- **Github**, come piattaforma principale per versioning, collaborazione, code review ed inoltre attraverso le funzioni al suo interno come ***Projects*** o ***Actions*** anche per gestione attività ed automatizzazione di test e deploy (per garantire CI/CD).
- **Eclipse**, come IDE di sviluppo Come risorse umane il progetto dispone di:
- **1 Configuration Manager**

- **5 Sviluppatori** (4 più il configuration manager che anch'esso svilupperà) Le risorse informatiche invece sono:
  - **Repository remoto su Github**
  - **Repository locali su PC dei membri**

6. Manutenzione del piano SCM

Il presente documento verrà revisionato ad ogni modifica significativa nel proceso di sviluppo o negli strumenti utilizzati.
Le modifiche allo stesso possono essere proposte da qualsiasi membro del team, ma per la loro approvazione serve l'okay di almeno 2 membri del gruppo (come per le PR).
Ogni versione aggiornata del piano SCM verrà tracciata tramite controllo di versione nel repository ***/docs***.

7. Extra

7.1 Guida all'uso di Github

In questa guida verranno descritte le procedure che ogni membro del team dovrà usare per utilizzare nel modo corretto Github e garantire ordine, tracciabilità e assenza di conflitti e/o file persi, seguendo le best practices di Git e Github.
Inoltre in questa guida si da per scontato di aver già installato Github Desktop.

7.1.1 Clonazione del repository

Prima di tutto per ottenere il progetto sul nostro PC locale dobbiamo clonare il repository remoto su di esso.
Questa operazione è da svolgersi **una singola volta** all'inizio!

```
git clone https://github.com/Michele-Vitali/ABLV_IngSW.git
```

Verrà così creata una cartella locale con tutto il codice e la storia del progetto nella directory dove ci troviamo (ad es. Desktop).

7.1.2 Regole di lavoro

Per evitare conflitti e problemi sulle versioni stabili **non si lavora mai direttamente sul branch main**.
Ogni modifica verrà fatta su un **nuovo branch** a partire dal branch di sviluppo **dev** seguendo questa naming convention:

- **feature/*nome_feature***, per nuove funzionalità (ad es. feature/login_page)
- **bugfix/*nome_bug***, per correzioni (ad es. bugfix/fix_login_page)
- **hotfix/*nome***, per correzioni urgenti (ad es. hotfix/deploy_error)

Si incoraggia l'uso di nomi chiari e coerenti con ciò che il branch tratta, così da far capire nell'immediato agli altri membri con cosa si ha a che fare.

7.1.3 Creazione di un branch

Prima di iniziare a fare qualsiasi modifica, si deve **aggionrare sempre** il repository locale eseguendo un **pull** tramite Github Desktop così da ottenere i file aggiornati all'ultima modifica applicata.
Solo dopo si crea un **nuovo branch** a partire da **dev** per iniziare a lavorare sulla modifica.

7.1.4 Salvare e caricare le modifiche

Quando vengono eseguite delle modifiche e si ritiene utile il caricamento delle stesse sul repository remoto allora si può eseguire un **commit** sempre tramite Github Desktop utilizzando:

- Titolo, deve essere breve e far intendere al volo cosa si è modificato.
- Descrizione, qui si deve descrivere bene i cambiamenti effettuati ed il motivo; inoltre se ritenuto necessario si dovrebbe spiegare ancora meglio delle modifiche apportate che potrebbero risultare difficile da comprendere.

7.1.5 Pull Request

Quando si è soddisfatti della propria modifica allora si può eseguire una **Pull Request** tramite Github web.
Qui si deve prima di tutto comparare le modifiche e stare attenti di selezionare la direzione della PR, che deve essere **base: *branch_precedente* <- compare: *nuovo_branch*** (dove branch_precedente dovrà sempre essere **dev** tranne quando il Configuration Manager decide di fare il merge con il main). Si devono scegliere, così come quando si esegue un commit, un **titolo** ed una **descrizione** del lavoro svolto che siano chiari e concisi. N.B.:
Come titolo della PR si deve mettere ***titolo_pr keyword #numero_issue***
Dove:

- titolo_pr, indica il titolo chiaro della stessa
- keyword, deve essere una tra quelle proposte qui: https://docs.github.com/en/issues/tracking-your-work-with-issues/using-issues/linking-a-pull-request-to-an-issue#linking-a-pull-request-to-an-issue-using-a-keyword.
  Così avviene il link tra issue e PR, così che una volta approvata la PR l'issue verrà chiuso in automatico eventualmente.
- #numero_issue, indica il numero che Github assegna in automatico all'issue, così che quando eventualmente chiude sa quale issue deve prendere.

Dopo tutto ciò, si devono assegnare dei **revisori** alla propria PR per l'approvazione delle modifiche (per semplicità si può assegnare tutti i membri e poi i primi due che controllano approveranno, commenteranno o rifiuteranno le modifiche).

Solo dopo eventuale approvazione quindi verrà effettuato il merge nel branch **dev** e sta al Configuration Manager poi effettuare il merge nel **main** una volta ritenuto idoneo.

7.1.6 Best practices

1. Prima di eseguire qualsiasi modifica ricorda di aggiornare il repository locale tramite **pull**.
2. Per evitare al meglio conflitti e/o file persi si dovrebbe creare un branch per risolvere o implementare **funzionalità** "**piccole**" man mano, ovvero che non richiedano troppo tempo, così da non perdere (e far perdere) eventuali modifiche che intanto altri possono eseguire.
3. Non si deve **modificare/eliminare file** che altri potrebbero star modificando in quel momento per implementare le loro modifiche; se proprio si ritiene necessario, concordare prima con gli altri membri del gruppo ogni azione eseguita su file di "possibile disastro".
4. Non si devono **spostare o rinominare directory** senza aver avuto prima l'approvazione degli altri membri del gruppo.
5. In caso di eventuali conflitti **informare sempre** gli altri membri del gruppo per decidere quale contenuto sia corretto (e quindi da tenere) e quale no. Dopo approvazione dei membri allora procedere con il **merge**.
6. Dopo aver approvato una PR il **branch** responsabile delle modifiche dovrebbe venire **cancellato** in **automatico** da Github, ma per garantire una buona pulizia della repository controllare comunque che non ci sia più e nel caso cancellarlo manualmente.