**NOTA: NEL SEGUENTE ESERCIZIO HO PROVATO AD IMMERGERMI NEL RUOLO DI UN RELEASE MANAGER, INVENTANDO E SIMULANDO UN CONTESTO DI PROGETTO.**

BUSINESS: GUCCI

TASK: IMPLEMENTAZIONE DI DUE NUOVE FEATURES PER L’E-COMMERCE APPLICATION

* Implementazione sezione per la personalizzazione degli abiti
* Implementazione sezione per un camerino virtuale

RISORSE DEL TEAM:

* 10 analisti programmatori divisi equamente in 2 team +2 team leaders
* 5 testers +1 team leader

VERSIONE ATTUALE: 2.1.1

VERSIONE PREVISTA DOPO IL RELEASE SENZA GESTIONE DI HOTFIX : 3.1.1

**Operazioni preliminari (ma assolutamente non secondarie):**

* Analisi dettagliata della richiesta del cliente, tramite lo studio di un documento discorsivo redatto dagli analisti funzionali;
* Successiva analisi tecnica da parte dei team leaders (con eventuale collaborazione dei team stessi), in modo da valutare le tempistiche di sviluppo, testing e messa in produzione;
* Accordare le tempistiche col cliente.

**Effettuate le operazioni preliminari, suddivido i compiti in macro aree:**

* Team sviluppatori 1: sviluppo prima feature;
* Team sviluppatori 2: sviluppo seconda feature;

Richiedo ai team leaders di suddividere il lavoro tra i componenti dei loro team nella maniera più efficiente.

Inoltre richiedo un report settimanale sull’andamento delle operazioni di sviluppo, in modo da poter gestire sul nascere eventuali problemi, e contestualmente erogare un feedback al cliente sull’andamento del progetto.

**Dopo queste fasi di managing dei team, definisco un git flow:**

* Ovviamente trattandosi di implementazione di nuove funzionalità, saranno già presenti all’interno della repository remota i branch master e develop (il primo mantiene una copia del progetto, pulito e funzionante, con i tag relativi alle diverse versioni rilasciate, in modo da registrare una cronostoria del progetto in toto; il secondo mantiene la stessa copia del progetto in master, ma essendo un branch di sviluppo e di supporto al master non necessita del tag relativo alla versione);
* Ogni funzionalità verrà implementata staccando dal develop un branch feature con i seguenti nomi:
  + - FeaturePersonalizzaAbitoLogic per sviluppare la prima funzionalità in elenco;
    - FeatureCamerinoVirtualeLogic per sviluppare la seconda.
* Data la complessità delle due nuove funzionalità, ma anche in virtù della loro analogia, stacco da ognuno dei precedenti feature branches due sotto-branch:
  + - FeaturePersonalizzaAbitoLogic:
* FeaturePersonalizzaGrafic
  + - FeatureCamerinoVirtualeLogic:
* FeatureCamerinoGrafic;

E’ chiaro quindi che nei sub-branch con la digitura Grafic verranno implementate le componenti grafiche relative alle due funzionalità, mentre nei branches padre Logic la parte di logica di gestione.

* Il branch di release avrà la dicitura Release-[tag-versione], in modo da identificare chiaramente nella cronologia la versione del software “generata” dalla stessa release.
* Un eventuale hotfix sarà nominato HotFix-[tag-versione], con motivazione analoga al punto precedente.

**Definito il git flow, determino le procedure di commit e di integrazione dei branch:**

Preventivamente richiedo agli sviluppatori di effettuare il minor numero di commit possibile (e quindi di organizzare i file da committare in gruppi ), per mantenere una cronologia pulita

* I due branch di tipo Grafic, prima di pushare le proprie commit nei relativi branch padre (Logic), effettueranno una merge degli stessi, e gli sviluppatori responsabili risolveranno all’interno del loro branch eventuali conflitti. Dopo di che effettueranno una pull-request al branch relativo di tipo Logic. Questa procedura dovrebbe assicurare l’assenza di conflitti in questa fase di integrazione.
* Il secondo step è quello di integrare i due branch, FeaturePersonalizzaAbitoLogic e FeatureCamerinoVirtualeLogic nel branch di develop. La mia strategia è la seguente:

1. Entrambi i team effettuano (possibilmente insieme) il merge del develop nei loro branch di competenza, per risolvere nel loro ambiente di sviluppo eventuali conflitti con la versione del progetto attualmente in uso.
2. Successivamente, solo il team responsabile del FeaturePersonalizzaAbitoLogic effettuerà una pull-request sul branch develop.
3. Solo dopo l’accettazione della pull-request, il team responsabile del branch FeatureCamerinoVirtualeLogic effettuerà una nuova merge del develop, per controllare gli eventuali conflitti che l’integrazione del primo branch potrebbe generare.
4. Dopo la risoluzione dei conflitti anche il secondo team effettua una pull-request.

* Anche se il precedente punto dovrebbe aver eliminato la possibilità di errori, in develop il team tester effettuerà tutta la serie di test necessaria per la verifica dell’effettivo funzionamento del software con le due nuove features integrate.

**Fase di release**

Solo a questo punto verrà creato il branch Release-3.1.1 (relativamente al mio esempio), ed inizia la fase di rilascio. Qui nuovamente i tester effettueranno una nuova serie di test, che comprendono anche:

1. Regression test, per valutare l’effettivo funzionamento di vecchie features che potrebbero essere compromesse dall’integrazione delle nuove.
2. UAT, serie di test effettuati al cliente, che verifica l’effettivo funzionamento delle nuove features, le accetta oppure può richiedere alcune modifiche.

Se i test risultano tutti positivi e se il cliente ha accettato l’implementazione delle nuove funzionalità, la nuova versione del software può essere effettivamente rilasciata. Il merge del branch di release deve essere effettuato sia sul branch master, ed accompagnato da un opportuno tag relativo alla versione (secondo il mio esempio, e in assenza di hotfix, sarà 3.1.1), sia sul branch di develop, per aggiornare anche questo alla nuova versione del software, senza però che questo necessiti del tag di versionamento.

A questo punto inoltre potranno essere eliminati definitivamente i branch di features e contestualmente anche il branch di release.

**Gestione di un hotfix**

Per quanto la procedura di integrazione e testing possa essere meticolosa, non è detto che una volta rilasciata la nuova versione questa sia effettivamente funzionante. Può accadere invece che l’applicazione non svolga i suoi compiti principali; questa è chiaramente una situazione estremamente grave e critica.

Personalmente indirizzerei tutte le risorse del team alla risoluzione di questa condizione, in gergo hotfix, (che nel mio esempio nominerei HotFix-[3.1.2]). **Il cliente è al primo posto per un consulente, quindi in questa condizione non esistono ne giorni festivi ne orari proibitivi**.

N.B. il branch di hotfix è l’unico che viene staccato direttamente dal master, dopo la risoluzione del problema si dovrà effettuare il merge di tale ramo sia sul master, con l’aggiornamento del tag della versione (che in questo caso sarebbe 3.1.2), ed ovviamente anche sul branch develop per pulirlo dalla presenza di questo grave bug (nel merge sul develop, come per la release, non è necessario associare un tag di versionamento). Successivamente potrà essere eliminato anche il ramo di hotfix.

**Cose da fare (e da non fare), al fine di preservare l’integrità del software**

Oltre che definire una logica efficiente, sia per il git flow che per le commit e le operazioni di merge (come specificato nei punti precedenti), risulta molto importante effettuare i Regression tests.

Inoltre NON SI DEVE mai eliminare il ramo develop. Teoricamente sarebbe sempre possibile ricrearlo con l’ultima versione funzionante del software, ma contestualmente ogni operazione di sviluppo dello stesso andrebbe persa (anche se solo temporaneamente, in quanto su sistemi distribuiti come Git ogni utente mantiene una copia locale dello sviluppo a cui sta lavorando, ma sicuramente è condizione di rallentamento del progetto).

**Cos’è una pull-request?**

Ponendo il caso che abbia sviluppato una nuova funzionalità sul branch Feature, dopo il push in remoto (che avviene sul corrispettivo branch Feature remoto), chiedo al mio referente, tramite una pull-request, se accetta il mio sviluppo, e in caso positivo integrerà lo stesso sul branch develop remoto.