QA – Procedure di RELEASE

Editato da: Fabrizio Cornelli
Versione: 1.0.5
Definito nelle sue linee da: Marco Valleri, Daniele Milan, Fabrizio Cornelli, Fabio Busatto, Alberto Ornaghi

Questo documento descrive le procedure di Release e le metodologie di test che verranno adottate a partire dalla 9.2.3, e che coinvolgeranno tutti gli sviluppatori, i Fae e il gruppo QA di HT (Fabrizio Cornelli, Matteo Oliva e Marco Losito).

Le procedure qui descritte, così come quelle definite nel documento di crisi, saranno inserite in Test Rail.

1 Tipi di Release

Le fasi che portano dallo sviluppo alla release si articolano su diverse settimane a seconda del tipo.

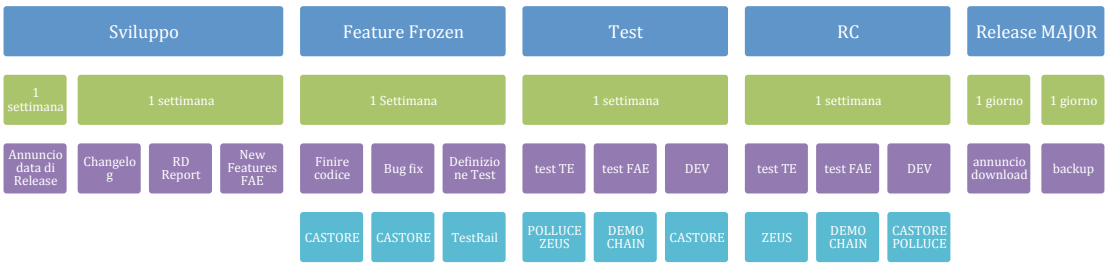
La data di RELEASE, nei casi non di crisi, viene annunciata dal CTO con un anticipo di mese nel caso di Major, con due settimane nel caso di Minor. Cadrà, possibilmente, di lunedì. Durante questo annuncio viene prenotata la presenza dei Fae, per le fasi di Test e RC. Si definisce il numero di versione da mettere in version. In questa email saranno definiti i ruoli dei vari server di test. Inoltre vengono indicato quali componenti del sistema sono coinvolti nella release.

1.1 Major Release

Le major release raccolgono interventi significativi al codice, aggiunta di funzionalità o modifiche che richiedono un intervento al manuale. I test sono estesi e profondi, si testano anche installazioni su server di lingua diversa dall'inglese. Le modifiche al manuale dovranno essere concordate con il CTO entro un mese dalla release.

Il CTO concorda una riunione RD Report con i commerciali e una riunione New Features con i FAE.

La fase di SVILUPPO termina il venerdì antecedente alle tre settimane prima della data di RELEASE, dopo la quale comincia la fase di Feature Frozen. A seguire una settimana di Test e una di RC.



1.2 Minor Release

Le release minori sono bug fix o estensioni di funzionalità che non richiedono l'intervento al manuale. I test riguardano le funzionalità di base e le parti modificate.

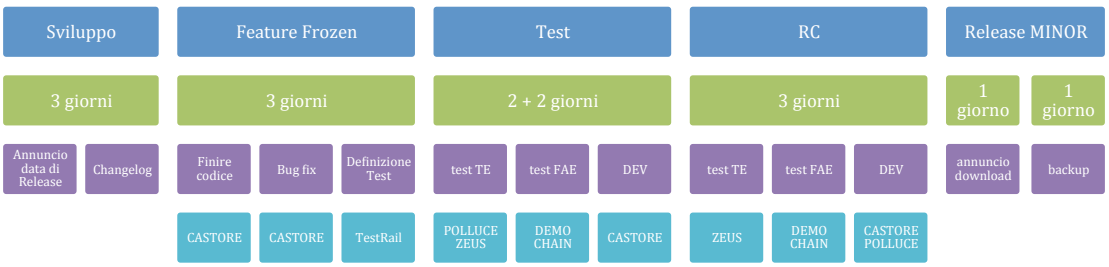
E' importante chiarire **che le Minor Release non possono contenere modifiche significative al codice**, come aggiornamento driver, aggiornamento toolchain, cambiamenti strutturali. Nel caso ci siano grandi modifiche da includere, è necessario tenerle in un branch git separato e proporle alla successiva release Major.

Le release minor partono dal branch git dell'ultima release disponibile, e non dal branch devel, che invece rappresenta la linea di sviluppo della prossima major. Una convenzione adottata è quella di avere un branch, chiamato 9.2.x, generato dal tag 9.2, che raccolga tutte le minor release 9.2. Ogni modifica a questo branch deve essere copiato nel ramo devel (merge).

Ai FAE si richiede la presenza per tre giorni la settimana precedente alla Release Minor, qualora disponibili.

Le fasi si articolano su due settimane, comprimendo le fasi secondo questi principi:

1) la data di annuncio e la richiesta di Changelog avvengono al termine della settimana che precede di quindici giorni il lunedì, data di Release.
2) La fase Feature Frozen è di tre giorni. Si ritiene che sia un tempo sufficiente per chiudere il codice aperto e perché gli sviluppatori testino il proprio codice. Anche i test da inserire e da concordare non saranno tantissimi.
3) La fase di Test dura complessivamente 4 giorni.
4) La fase RC dura tre giorni.



1.3 Release di Crisi

Nel caso di Release di Crisi il tempo e la precisione sono cruciali, l'impegno da parte di tutti è minimizzare l'intervento al codice, modificando solo lo stretto indispensabile per risolvere il problema specifico. L'indicazione è di generare un branch dalla release precedente, recuperando l'ambiente salvato (Es: branch 9.2.x), così come avviene con le release Minor.
Nel caso in cui la crisi avvenga durante la fase RC, si utilizzeranno i pacchetti RC per apportare eventuali modifiche aggiuntive.

1.4 Release di Hotfix

Se la contingenza richiede che venga rilasciato, in via eccezionale, un hotfix critico, per uno o per più clienti, entro un giorno, ad insindacabile giudizio del CTO si può procedere con la Release di Hotfix.
I test saranno svolti su Polluce e poi su Zeus, avendo avuto cura di riportare lo stato dello snapshot all'ultima release prima dell'applicazione della patch.
La release Hotfix può innescare una release minor, che consenta i test più approfonditi.
L'installer è minimale, applicabile come patch all'ultima release.

2 Fasi delle Release

2.1 Sviluppo

Fase durante la quale ogni sviluppatore, sulla base delle linee guida determinate dal CTO, aggiunge caratteristiche (features) ai componenti del sistema, operando in autonomia tutti i test che ritiene opportuni per verificare la consistenza interna del codice. In questa fase, usando CASTORE, può effettuare i test di integrazione, per verificare la corretta interazione tra il proprio componente e il resto del sistema. In autonomia potrà effettuare l'upgrade del core con le versioni prodotte.
Lo sviluppatore svolge inoltre sia i test atti a verificare la conformità delle caratteristiche introdotte, sia i test di regressione, che verificano che le nuove modifiche non abbiano rovinato il funzionamento di qualche altra caratteristica del sistema. Prova oltresì ogni configurazione ragionevole, in termini di versioni e di condizioni.

La fase di sviluppo termina tre settimane prima della data di RELEASE MAJOR oppure due settimane prima della RELEASE MINOR.
Entro la fine della fase occorre che ogni sviluppatore, indipendentemente da fatto che abbia effettuato cambiamenti al proprio codice, svolga le seguenti operazioni all'inizio della settimana:

1) Per ogni componente di cui è responsabile mandi una **mail a qa@hackingteam.com** contenente:
    a. **Changelog ad alto livello**, ad uso dei clienti, che raccoglie Alberto Ornaghi come di consueto, per redigere il changelog di release. Se non sono stati apportati cambiamenti, si mandi una mail che lo espliciti.
    b. **Changelog a basso livello**, in altre parole l'elenco dei cambiamenti nel codice effettuati dall'ultima release, utilizzabile dal gruppo QA per organizzare i test e gli incontri con gli sviluppatori per accordarsi sui test. Occorre elencare le sotto componenti che sono state intaccate dalle

modifiche, così da determinare la profondità dei test che si ritiene necessaria per avallare la versione.

c. Devono essere segnalati eventuali cambiamenti da apportare alla **compatibility list** e al **manuale**. Nel caso di modifica al manuale, si contatti il responsabile esterno della documentazione. Solo nelle release MAJOR si ammette la modifica al manuale. Qualora sia necessario, si descrivano le best practices per l'uso delle nuove funzionalità, specificando se siano informazioni da mettere nel manuale o da comunicare ai FAE.
d. Nel caso possa essere stato introdotto un cambiamento che possa modificare **l'invisibility list**, occorre dichiararlo nella mail, così che si possa verificare con test diurni sul server preposto (MINOTAURO), in stretta collaborazione con il gruppo QA, eventuali anomalie.
e. Devono essere esplicitati quali **test** sono stati effettuati nella fase di Sviluppo, quali si intende svolgere nella fase successiva e quali si ritiene necessario far svolgere al gruppo QA. Si spiega il contesto e le modalita' d'uso dei moduli o delle features aggiunte o modificate, per facilitare la progettazione e l'esecuzione dei test.

2.2 Feature Freeze

La fase Feature Freeze è caratterizzata dal fatto che al codice non sono più apportate modifiche che non siano necessarie all'eliminazione di qualche bug o alla chiusura del codice aperto. Durante questa fase sono definiti test che saranno svolti nelle fasi TEST e RC.

Ogni sviluppatore che abbia effettuato modifiche da integrare nella release viene **convocato dal Responsabile QA**, ed insieme valuteranno nel dettaglio tutti i test che sono stati svolti e che verranno svolti. I test sono codificati e inseriti in *TestRAIL*, piattaforma che consente di definire nel dettaglio tutte le casistiche di *Test Case*, in classi chiamate *Test Suite*. Queste sono istanziate in specifici *Test Run*, ognuno dei quali assegnato a singole release. Ogni *Test Case* presente nelle Test Suite consente di specificare i requisiti, l'applicabilità, i passi (*step*) da eseguire e i risultati da verificare. Ogni istanza di test consente di specificare i risultati per ogni passo. I test sono organizzati ad albero, secondo una tassonomia definita e mantenuta dal responsabile QA.

Un sottoinsieme significativo di questi *Test Case* è istanziato in uno o più *Test Run* per la release in oggetto, così da poter definire un livello di qualità che si vuole raggiungere. Poi si assegna ai Test Engineer (TE) disponibili (in parte del gruppo QA, in parte volontari del gruppo di sviluppo, in parte Fae), un gruppo di test da eseguire durante la fase di TEST e un gruppo durante la fase RC. Per ogni test eseguito deve essere riportato l'esito ed eventuali *screenshot* e log utili per il *debug*. Ogni TE riporta i risultati tramite piattaforma *TestRAIL*, il gruppo QA trasmetterà i dettagli del bug agli sviluppatori di competenza.

I test appartengono a una di queste categorie:

1) **Functional Regression**: test di base, installazione, upgrade, disinstallazione, sync, presenza di evidence, persistenza. Da eseguire ad ogni step di release (TEST e RC). Ogni architettura dovrebbe completare il ciclo in meno di un'ora. Il test del server può richiedere un paio d'ore.

2) **Release Specific**: test specifici di release, utili a testare le nuove funzioni. Da eseguire ad ogni step di release (TEST e RC). E' la parte più estesa dei test da eseguire alle release.
3) **Continous Testing**: test di verifica che le funzionalità delle release precedenti siano ancora presenti, verifica che le nuovi versioni degli OS e delle applicazioni continuino a garantire il corretto funzionamento, verifica dell'invisibilità dei vari OS. Da eseguire quotidianamente su un ciclo di un paio di mesi, ci sarà una risorsa che si occuperà di svolgerli e di automatizzarli.
4) **Demo**: test atti a verificare che le demo svolte dai Fae siano ancora funzionanti ed efficaci. Da eseguire ad ogni step di release (TEST e RC).
5) **Acceptance**: elenco dei test tipicamente richiesti dai clienti in fase di delivery. Questo set è curato da Daniele Milan e dai FAE. Da eseguire ad ogni step di release (TEST e RC).
6) **Crisis**: Da eseguire in caso di Crisi.

Al termine della fase, per ogni componente modificata **lo sviluppatore salvi su git le modifiche**, nel branch devel o minor, a seconda dei casi. In questa fase non occorre un tag.
Dove necessario si genera un core, e si copia nella directory "cores galileo" di rcs-dev.

2.3 Test

Questa fase è dedicata all'esecuzione dei test stabiliti nella fase Feature Freeze.
Si crea un pacchetto di installazione con tutti i core presenti su RCS-DEV e si aggiorna POLLUCE. I test sono svolti dai TE su questo server, pertanto l'aggiornamento dei core sarà gestito dal gruppo QA e da Alberto Ornaghi.

Si crea un'attività "x.y.z TEST", e un target specifico per ogni core. I test devono essere fatti in modo da poterne tenere traccia, sia per i casi positivi sia per quelli negativi, in modo da facilitare la comunicazione tra sviluppatori e tester.

Gli sviluppatori potranno correggere eventuali bug su CASTORE. Per ogni componente modificata **lo sviluppatore salva su git le modifiche**, nel branch di competenza della release (devel o 9.2.x). In questa fase non occorre un tag.

Gli sviluppatori che vogliano contribuire ai test, o che sono eletti al ruolo di TE ad insindacabile giudizio del CTO, in questa fase, avranno l'opportunità di contribuire attivamente alla qualità del prodotto, facendosi carico di un insieme di test, piccolo a piacere, che siano relativi a moduli di cui non siano autori. Quest'assegnazione non deve andare in conflitto con l'impegno prioritario dello sviluppatore a correggere eventuali bug rilevati nella parte di codice di cui è responsabile, solo gli sviluppatori che abbiano corretto i propri errori o che non siano direttamente responsabili della release, sono eleggibili a questa funziona temporanea.
Il primo test da eseguire è quello di Upgrade, tutti gli altri in seguito.

Durante questa fase è richiesta la presenza di un FAE per due o tre giorni, che si occupi di installare su una catena demo la versione corrente e svolga, in autonomia, i test di Demo e Acceptance. Su POLLUCE si svolgono i test funzionali e di regressione. Il

Fae riporta i problemi trovati al gruppo QA, plausibilmente tramite TestRail o via email.
Al termine dei test il Fae produce un backup delle evidence coinvolte nel test.

I test effettuati sul server devono richiedere che siano analizzati i log prodotti.

Al termine della fase Test, quando cioè tutti gli errori sono stati corretti, si passa alla fase successiva, che consiste nella verifica ultima del funzionamento di tutto il sistema in condizioni il più possibile realistiche.

2.4 Release Candidate

La fase antecedente la release è utilizzata per validare che la RELEASE CANDIDATE corrente sia adeguata per la release.

La prima operazione svolta nella fase RC è la preparazione del pacchetto di installazione da parte di Alberto Ornaghi, che calcola un digest dei **core presenti su rcs-dev**. a quel punto **non potranno più essere sostituiti**, se non in una fase RC successiva, atta a risolvere i bug identificati nella RC precedente.

La verifica della RC corrente avviene con questi passi:

1) ZEUS è ricostruito da uno snapshot delle VM che contengano solo il SO aggiornato. ZEUS è un sistema di test il più possibile aderente a un'installazione realistica, sarà composto di un backend, uno shard, un frontend, due anonimizer, un connector, un archivio e un token fisico. Si preveda uno script per la creazione veloce degli utenti, dell'attività e dei target.
2) ZEUS è connesso tramite connector verso un server RCS TEST ARCHIVIO, in maniera da mantenere traccia dei test effettuati tra una sessione RC e l'eventuale successiva.
3) Si crea un'attività "x.y.z RC n", e un target specifico per ogni modulo. I test devono essere fatti in modo da poterne tenere traccia, sia per i casi positivi sia per quelli negativi, in modo da facilitare la comunicazione tra sviluppatori e tester. Solo i TE e gli amministratori hanno un account su ZEUS.
4) Si aggiorna MINOTAURO con i pacchetti nuovi, in modo da svolgere i test d'invisibilità.

Da questo momento, eventuali crisi si risolvono con i pacchetti RC correnti.

Durante questa fase viene richiesta la **presenza di un FAE per due o tre giorni**, che si occupi di svolgere i test di Demo, Acceptance e Poc su ZEUS e sulla catena demo.

Il primo test da svolgere è il **test di Upgrade (QA 1)**, che lascia il sistema con la versione da verificare.

Il Fae riporta i problemi trovati al gruppo QA, plausibilmente tramite TestRail.
Al termine dei test il Fae produce un backup delle *evidence* coinvolte nel test.

Qualora i bug rilevati non consentano la release, vengono risolti dagli sviluppatori e si procede ad un'altra iterazione del ciclo di test RC, che da RC1 diverrà RC2, e così via. Occorre verificare e registrare i digest dei core per essere certi che non siano stati

introdotti core diversi da quelli che ci si aspetta e per mantenere un riferimento per le iterazioni successive.

A valle del test, qualora non ci siano problemi, o che siano stati dichiarati accettabili per la release dal CTO, sono verificati i digest dei pacchetti e il CTO delibera la Release. **È fatto anche un backup di tutto ciò che si ritiene necessario per minimizzare i tempi di intervento in caso di Release di Crisi**.

Qualora la fase RC raggiunga il suo obiettivo prima della fine della settimana, comunque si aspetterà il lunedì successivo per la distribuzione

2.5 Release

Il giorno della release si prepara la mail con l'annuncio, si verifica il digest del pacchetto di release, lo si sposta nella directory di download, si verificano le licenze, si generano i ticket.

Tutti **i responsabili** delle componenti operano un **merge del git da devel a master e applicano un tag nella forma: x.y.x**, ad esempio 9.2.2, secondo le indicazioni di release.

2.5.1 Download

Fabio, se necessario, genera una directory sul server di supporto e copia l'albero dei download relativi alla release in oggetto. Questo albero contiene una directory galileo che contiene:

- x.y (ultima major)
  - setup
  - console
  - ocr
- x.y.z (ultima minor)
  - setup
  - console
  - ocr
- doc
- exploit
- network injector
- contrib (air)

2.5.2 Licenze e News

Daniele indica quali clienti sono attivi.
Fabio genera le nuove licenze, solo in caso di Release Major o in alcuni casi di Crisi.
Pubblica la news della nuova release, contente il link all'area segreta di download.

2.5.3 Altro

Il giorno successivo è dedicato al backup e Fabio Busatto si occupa di:

1) verifica del backup
2) sostituzione disco di backup

Eventuali installazioni particolari saranno seguite, nelle release Major, dai Fae.
Lo script di generazione degli Exploit deve essere aggiornato, per togliere gli hash della versione precedente, aggiungendo quelli della versione attuale.

3 Appendice

3.1 Linee generali dei test

Ogni test deve essere descritto nei suoi prerequisiti, negli step da compiere e nelle verifiche da controllare. A valle della sua istanziazione, ogni test verrà assegnato ad un Test Engineer, che potrebbe essere uno sviluppatore o un componente del gruppo QA. La sua esecuzione serve o a verificare che il funzionamento testato rispetti le specifiche oppure a fornire allo sviluppatore tutto ciò che lo possa aiutare a capire l'errore riscontrato, che dovrà essere supportato da descrizione ed eventuale screenshot.
Se il test prevede che ci siano delle evidence, occorre specificare il riferimento all'istanza della factory sul server, che non deve essere cancellata.
Chi esegue il test è responsabile di svolgerlo al meglio, deve capirlo bene nel suo intento ed eventualmente suggerire elementi che possano aiutare a migliorarne l'esecuzione o la superficie.

ATTENZIONE:
I test sono responsabilità del gruppo QA, nella loro formulazione, nella loro esecuzione e nella loro comunicazione. Si richiede la massima collaborazione per evidenziare ogni incongruenza e ogni problema riscontrato.

I bug sono responsabilità degli sviluppatori, non del gruppo QA. La prima e più importante difesa contro i difetti del codice è lo sviluppatore, che mentre scrive il codice, valuta e controlla ogni possibile anomalia e attua e progetta i primi test per la verifica del funzionamento della modifica, nel caso normale e nei casi speciali.

Per minimizzare gli errori si consiglia la tecnica di Pair Review, che consiste nello spiegare il codice modificato ad un collega.

3.2 Server di Test

Nella mail di presentazione della release verranno definite le associazioni tra server e ruoli dei server.

DEV (Castore): server di test durante lo sviluppo, per codice potenzialmente instabile
TEST (Polluce): server di test durante le fasi TEST e RC, per codice stabile
INVISIBILITY(Minotauro): server ad uso esclusivo dei test automatici
ACCEPTANCE(Zeus): sistema di test per i test RC, aggiornato esclusivamente dal responsabile della release. Questo server si sua anche per i test di UPGRADE, aggiornato esclusivamente dal responsabile della release. Occorre che ci siano sempre due snapshot, uno al SO aggiornato (per i test di ACCEPTANCE) e uno all'ultima release stabile (per i test di UPGRADE).