Costituzione del Repository regionale del codice sorgente e delle buone pratiche

Modello Repository

Data: **Compilato:** Francesco Azzola

Data: **Rivisto:**

Data: **Approvato:** _____

Destinatari:

1 GENERALITÀ

Oggetto:	La Community Network Umbria per le buone pratiche e il repository centrale del riuso
Riferimenti a documenti aziendali:	
Riferimenti esterni:	*Delibera n. 1527/2015*
Moduli utilizzati	
History	Versione 1
Glossario, abbreviazioni e acronimi:	**CCOS**: Centro di Competenza *Open Source;* **CNU**: Community Network Umbra; **CC-BY**: Creative Commons License; **LOD**: Linking Open Data; **OD**: Open Data; **PA**: Pubblica Amministrazione o Pubbliche Amministrazioni ; **SIR**: Sistema Informativo Regionale; **U.D.:** Punto Zero.

2 INTRODUZIONE

Il documento descrive il modello di organizzazione del repository regionale del codice sorgente e delle buone pratiche.

Tale repository conterrà tutti i progetti che verranno messi a riuso dalla Regione Umbria.

I progetti sono composti da sorgenti, documenti tecnici e documenti di progetto secondo le modalità previste dal modello PRINCE 2. L'obiettivo è quello di avere un unico punto dove memorizzare e condividere tutti progetti.

2.1 LA SITUAZIONE ATTUALE

La delibera regionale n° 1572 del 29/12/2015 affida, tra gli altri mandati, a Punto Zero il progetto e il disegno di un Repository centralizzato che dovrà contenere il codice sorgente e le buone pratiche dei progetti che vengono messi a riuso dalla Regione stessa.

Il repository è stato pensato secondo due direttrici primarie:

- Strumento di riferimento per la gestione del patrimonio software di proprietà della Regione Umbria, per la gestione ed il versionamento degli items in esso contenuti. A riguardo per items si intendono sia i sorgenti dei componenti software prodotti durante la fase di sviluppo del progetto, sia i documenti tecnici e di progetto, secondo regole e caratteristiche rappresentate da concetti di modellazione comuni in letteratura per questa tipologia di strumenti:
    - Artifact/item: rappresenta l'oggetto atomico base che deve essere memorizzato nel repository;
    - Branch: è una deviazione dalla linea principale di "sviluppo";
    - Baseline: è il consolidamento delle versioni dei singoli items che vengono raggruppati in una versione di riferimento.
- Ambiente digitale in grado di rendere disponibile ed accessibile una informazione strutturata e chiara che sia da guida per le altre Amministrazioni pubbliche. Per questo, la strada del repository consente la messa a disposizione di un contesto in cui sarà possibile accedere alle risorse documentali e software licenziate dalla Community stessa in EUCL. A tal fine il repository presenterà funzionalità in grado di:
    1. accedere alla documentazione (manuali utente, manuali di installazione, specifiche tecniche);
    2. accedere ai sorgenti e compilati software;
    3. interagire con le amministrazioni per evoluzioni software;

2.2 DESCRIZIONE DELLA PROBLEMATICA E FINALITÀ

La finalità del lavoro proposto, con il presente documento, non è solo quella di definire e disegnare un Repository in grado di contenere e gestire i sorgenti e i documenti tecnici e di progetto dei prodotti

software di proprietà o di interesse della Regione Umbria, ma anche quella di rappresentare il contenitore delle "Buone Pratiche". Questo allo scopo di disporre, in un ambiente unico, di tutto il materiale di interesse necessario all'utilizzo concreto dei prodotti.

Per questo il repository oltre ai contenuti che lo caratterizzano come contenitore specialistico delle soluzioni tecnologiche, dovrà prevedere anche un contesto documentale statico per ospitare i seguenti contenuti tematici per ogni soluzione:

- Documentazione descrittiva della "buona pratica" con indicazione dei software a supporto, ma anche dei Soggetti che la utilizzano, dei tipi di utilizzo attuati, delle evoluzioni funzionali nel tempo, delle informazioni di promozione e di conoscenza della buona pratica;
- Documentazione tecnico – amministrativa per il riuso della soluzione e per l'iscrizione dei servizi messi a disposizione; cataloghi dei servizi e referenze;
- Progetti di attuazione della buona pratica con tutta la documentazione prodotta e certificata;

Stante il contesto di utilizzo del Repository oggetto di analisi, è chiaro che la problematica relativa al progetto è quella di predisporre un ambiente integrato che risolva la situazione di dispersione del software e dei documenti in archivi, uffici e competenze diverse, senza che vi sia un controllo centralizzato, sia sugli accessi, sia sulle modalità di interazione e gestione dei dati di tali repository.

Questa, peraltro, è la situazione riscontrata già in fase di censimento delle soluzioni attualmente in uso, che ha reso difficile pervenire ad un quadro complessivo del patrimonio informatico della Regione e che, si ritiene renda ancor più difficoltosa la gestione di tale patrimonio. La dispersione delle informazioni su diversi repository/uffici rende problematica l'individuazione dei documenti, delle prassi e dei software, soprattutto relativamente alle versioni, evidenziando per queste ultime, esigenze di regole chiare e certe di gestione.

Già limitatamente al solo software, da quanto detto, risulta difficoltoso poter estrarre un certa versione dal codice immagazzinato in uno di tali repository e distribuire a terzi il sorgente con le relativa documentazione. Tale scenario è quello tipico del riuso del software dove è richiesto di distribuire ad altri enti una versione del software che sia congruente non solo con i diversi componenti contenuti in tale versione ma anche con la relativa documentazione.

La necessità di definire un modello chiaro di gestione di un repository centralizzato che contenga, come già detto, non solo i sorgenti ma anche tutti i documenti della "buona pratica", così come descritta, diventa la chiave del processo di organizzazione di uno strumento che sia funzionale agli obiettivi della Regione.

E' necessario, altresì, definire un flusso operativo che stabilisca quali deliverables vanno inseriti nel repository e le responsabilità di tali operazioni.

L'obiettivo è quello di:

- Identificare e memorizzare gli item (o artifacts) che vanno gestiti nel repository;
- Controllare e monitorare i cambiamenti di ciascun artifact;
- Definire ed organizzare il versionamento degli artifacts;
- Definire ed organizzare le baseline;
- Tracciare le richieste di cambiamento;

- Definire la struttura del repository;
- Assicurare la riproducibilità del software sorgente.

2.3 DESCRIZIONE DELLA SITUAZIONE ATTUALE

Come sopra anticipato l'assenza attuale di un repository centralizzato dove far confluire tutti i prodotti software e documentali, che nel corso del tempo sono stati progettati, ha portato alla presenza di diversi repository tra loro scollegati con un proliferare di diverse modalità di gestione e memorizzazione delle informazioni.

L'assenza di linee guida precise e di un sistema centralizzato dove memorizzare gli items prodotti ha generato ridondanza delle informazioni e un disallineamento fra le stesse.

Tale situazione si ripercuote non solo sul reperimento del materiale alla bisogna, ma anche sulla distribuzione dei software prodotti, infatti in molti casi diventa problematico stabilire quale repository è quello di riferimento.

E' essenziale allineare ed unificare tali repository, centralizzando le informazioni ed evitando, al tempo stesso, la duplicazione in modo da individuare un repository autorevole per i progetti messi a riuso.

3 IL REPOSITORY REGIONALE DEL RIUSO

3.1 DEFINIZIONE DEGLI OBIETTIVI DEL REPOSITORY

Stante il quadro fornito, è necessario individuare i riferimenti progettuali di cui si terrà conto per elaborare la soluzione che riguarda il modello del Repository unico regionale, predisposto per supportare le seguenti caratteristiche funzionali ed organizzative tese a risolvere i problemi:

- Essere il concentratore di tutti i prodotti realizzati o acquisiti come software pubblico dalla PA locale regionale, come previsto ai punti 5.10 e 6.8 del "Disciplinare per l'attuazione della legge regionale n. 9/2014 " di cui alla DGR n.1778 del 22/12/2014, analogamente a quanto già accade per gli Open Data della piattaforma PA Umbria che costituisce oggi il contenitore di raccolta dei dati OD e LOD messi a disposizione dalle PA umbre. Con analogo principio sarà caratterizzato il Repository. In questo modo sarà possibile avere due riferimenti di prodotto aperti e pubblici: Dati (Portale regionale dell'Open Data) e funzioni (soluzioni informative ed organizzative per la produzione e gestione dei dati: Repository Regione degli output di progetto);
- Acquisire all'interno i prodotti informatici e di buona pratica presenti sul territorio regionale e messi a disposizione dagli Enti, sia per i prodotti realizzati, in conformità alla pratica internazionale legata ad open source ed open content sotto licenze di tipo "copyleft " come EUPL e CCBY, o similari, compatibili con il dettato della legge regionale 25 luglio 2006, n.11, "Norme in materia di pluralismo informatico, sulla adozione e la diffusione del software a sorgente aperto e sulla portabilità dei documenti informatici nell'amministrazione regionale " così come aggiornata dalla citata l.r. n.9/2014, che per quelli adottati attraverso il riuso passivo da altre amministrazioni;

Le finalità del repository sono molteplici e di diversa natura:

- costituire, per le Pubbliche Amministrazioni, un modello di accesso, utilizzo e partecipazione alle soluzioni indicate come buone pratiche dall'Amministrazione regionale;
- acquisire, razionalizzare, pubblicare e distribuire dati e informazioni da qualsiasi fonte informativa di interesse appartenente alla Pubblica Amministrazione o a soggetti Privati, coerentemente con i dettami normativi in materia di trasparenza e semplificazione;
- favorire la realizzazione e la diffusione di istanze digitali per la gestione dei processi e l'erogazione di servizi, attraverso forme di sussidiarietà tecnologica e organizzativa che minimizzino gli oneri per gli Enti, soprattutto per i Comuni notoriamente caratterizzati da minore capacità di investimento;
- progettare un ambiente in grado di creare e promuovere un catalogo di servizi predisposti dal realizzatore della soluzione, in cui per ogni buona pratica concessa a riuso, siano indicate possibili soluzioni offerte per la gestione e mantenimento della soluzione;

3.2 BREVE INTRODUZIONE A GIT E GITHUB

Git è un sistema di controllo di versione moderno che offre le funzionalità familiari di CVS o

Subversion, ma fornisce, al contempo, rispetto agli altri prodotti, una nuova visione delle modalità di gestione del software. Git estende la nozione stessa di sistemi di controllo versione (VCS) per la sua capacità di offrire quasi tutte le sue caratteristiche per l'utilizzo offline e senza un server centrale.

Oggi, gli sviluppatori di tutto il mondo stanno migrando in massa verso questa piattaforma. Gli utenti apprezzano le sue prestazioni, la flessibilità di utilizzo, le funzionalità offline, e le funzioni di collaborazione.

GitHub è la versione cloud di Git nel senso che offre le stesse funzionalità di Git ma con la possibilità di creare repository in cloud.

GitHub è in grado di condividere un repository e un changeset direttamente con un altri utenti semplicemente accedendo via web ed effettuando un check-in/ check-out degli items,

3.3 SOFTWARE CONFIGURATION MANAGEMENT

La gestione del codice sorgente e dei documenti, è una pratica più comunemente indicata come *SCM Software Configuration Management*. Con *SCM* si intende la gestione delle modifiche apportate ai documenti, a prodotti software e più in generale ad un insieme di informazioni. Il Configuration Management, in altri termini, ha lo scopo di permettere la gestione ed il controllo degli oggetti (siano questi prodotti software o documenti).

Nell'ambito del *Configuration Management* vengono gestiti gli input/output direttamente o indirettamente legati alla costruzione di un prodotto software. Si tratta, quindi, non solo di archiviare in modo controllato le varie versioni del codice sorgente sviluppato, ma anche di gestire le altre entità create nel corso delle diverse fasi dello sviluppo. Ogni elemento oggetto delle attività di gestione della configurazione viene normalmente chiamato **configuration item**. Perché tutti i configuration items possano essere adeguatamente gestiti è necessario che siano definite le possibili tipologie e le operazioni che su di essi possono essere compiute, compresi i ruoli dei vari attori coinvolti. Tutto ciò viene tipicamente definito all'inizio del processo di sviluppo del software (ovvero di un progetto) ed è dipendente in una certa misura dal progetto. La gestione delle configurazione può essere applicata, in generale, a diverse categorie di oggetti (comunemente noti come "*artefatti*"). Qui di seguito viene riportata una breve lista di tali categorie di oggetti:

- *Documenti di progetto:*
  - Prodotti di fase (vedi Prince 2);
  - Documenti tecnici;
- *Codice sorgente;*
- *Manuale utente;*
- *Test Case;*
- *Manuali di installazione e gestione del prodotto;*
- *Prodotti finali dello sviluppo (compilati, eseguibili, ecc.);*
- *Schemi dei database;*
- *Documenti relativi alla manutenzione del prodotto;*

- *Standard e procedure adottate;*
- *Documenti per la formazione tecnica e funzionale degli utilizzatori;*
- *Documenti descrittivi della buona pratica gestita attraverso lo strumento, presentazioni e seminari;*
- *Progetti in corso derivati dall'uso della buona pratica/software, istanze di partecipazione se bando aperto, istanze di risultato raggiunto se bando chiuso;*
- *Documenti descrittivi delle Amministrazioni cedenti e riusatrici, documenti per il riuso della soluzione.*

I cambiamenti apportati, a ciascuno di tali oggetti, vengono etichettati tramite l'utilizzo di numeri e/o lettere e vengono indicati come ***revisioni***. Nell'ingegneria del software la gestione delle revisioni è l'insieme delle pratiche adottate per garantire il controllo, la storicizzazione e la rintracciabilità delle modifiche apportate.

Una revisione può essere vista, secondo la teoria dei grafi, come una linea di sviluppo (denominata *trunk*), da cui si possono diramare uno o più rami (indicati come *branch*), che corrispondono a linee di sviluppo parallele che confluiscono in prodotti con differenti configurazione e/o funzionalità. Le revisioni si susseguono lungo una linea (linea del tempo) e l'ultima revisione viene indicata come *head*. La lista delle revisioni dal punto di partenza fino all'ultima revisione (head) viene indicata come *mainline*. L'operazione di merge, infine, è la convergenza di un ramo che viene integrato.

E' chiaro quindi che la gestione delle revisione e degli oggetti software è una disciplina molto complessa e che è necessario avere uno strumento di supporto che consenta di eseguire le operazioni prima descritte in maniera rapida, efficiente e sicura.

Il gestore del repository è uno strumento potente a supporto operativo e documentativo nelle fasi di sviluppo di un progetto ed in grado di gestire tutti gli aspetti che sono stati descritti.

Rimanendo confinati negli aspetti di gestione della documentazione del software e del sorgente (per gli altri temi di interesse del repository i criteri di modellazione sono gli stessi), GitHub è lo strumento attraverso il quale applicare e gestire le tecniche di configuration management. GitHub è, in altre parole, il repository finale dei prodotti software che si intendono mettere a riuso.

Il gestore del repository è un sistema staccato dalla fase di produzione vera e propria e si configura come un "archivio" dove depositare e tracciare i prodotti finiti.

Per prodotti finiti, si intendono gli output delle diverse fasi di un progetto. In tale ottica, i documenti finali, i prodotti software testati e pronti al rilascio vengono memorizzati nel repository centrale. Utilizzando tale scenario, è possibile continuare a sfruttare le potenzialità del prodotto per tutte quelle tematiche riguardanti la gestione delle versioni, il tracciamento delle modifiche che vengono fatte alle versioni finali, come ad esempio patches, integrazioni di documenti ecc.

3.4 IMPLEMENTAZIONE DEL REPOSITORY

Sempre restando nel contesto del repository utilizzato come archivio della documentazione del software oggetto di riuso, di seguito verrà illustrato il modello di implementazione del repository.

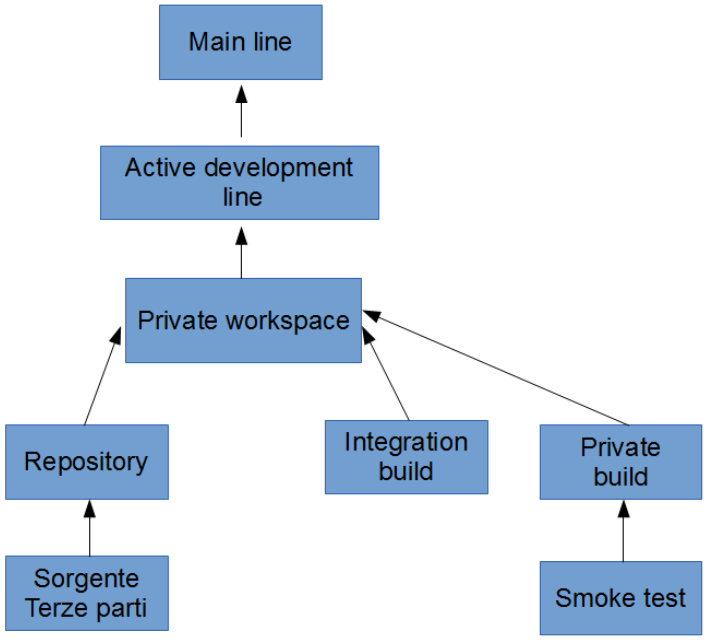
A supporto della strutturazione del lavoro e dell'utilizzo del repository vi sono alcuni patterns consolidati nel tempo che permettono di ottimizzare sia le attività svolte dal team di lavoro sia la produttività.

In generale si possono individuare tre categorie di pattern:

- ***Core pattern;***
- ***Workspace pattern;***
- ***Code line pattern;***

Ciascuno di questi tre patterns incide su un particolare aspetto dello sviluppo, della gestione del repository e dell'allineamento del codice sorgente e della documentazione.

Qui di seguito viene riportato uno schema di massima dei pattern e della struttura del repository e di come dovrebbe essere utilizzato il repository al fine di massimizzare l'efficienza. Il modello è indipendente dal tipo di strumento utilizzato.



Nome pattern	Descrizione
Mainline	Minimizza l'attività di merge, mantenendo il numero di

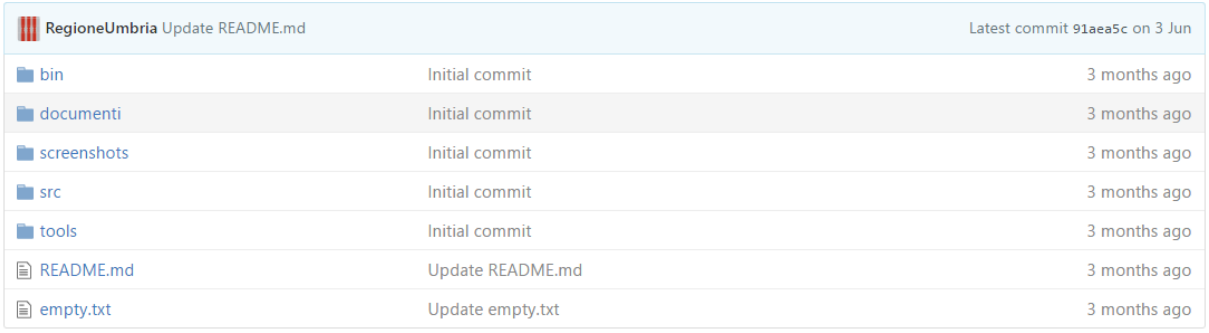
	linee di codici attive sotto controllo
Active Development Line	Mantiene stabile una linea di sviluppo che si evolve rapidamente.
Private Workspace	Evita che I problem di integrazione influiscano sulla linea di sviluppo di ciascun sviluppatore.
Repository	Inizializza un nuovo workspace popolandola con i dati provenienti dal repository.
Integration Build	Assicura che la baseline sia sempre affidabile e compilabile attraverso una attività periodica di integrazione e successive build.
Third Party Codeline	Gestisce il codice di terze parti
Smoke Test	Assicura che dopo le modifiche il Sistema continui a funzionare eseguendo uno Smoke Test

Più nello specifico, il sistema Github permette la creazione di branch e di organizzare tali branch in modo gerarchico. Questo risulta particolarmente utile per implementare i pattern prima descritti. Una tipica struttura a branch di un repository GIT è qui di seguito rappresentata:

- ***refs/heads/master*** (main line for future releases);
- ***refs/heads/releases/stable-x.y.z*** (set of branches for integrating changes and stabilizing code for release);
- ***refs/heads/user-xyz/mybranch*** (optional but very useful: namespace for individual users).

3.5 STRUTTURA DI RIFERIMENTO PER UN PROGETTO

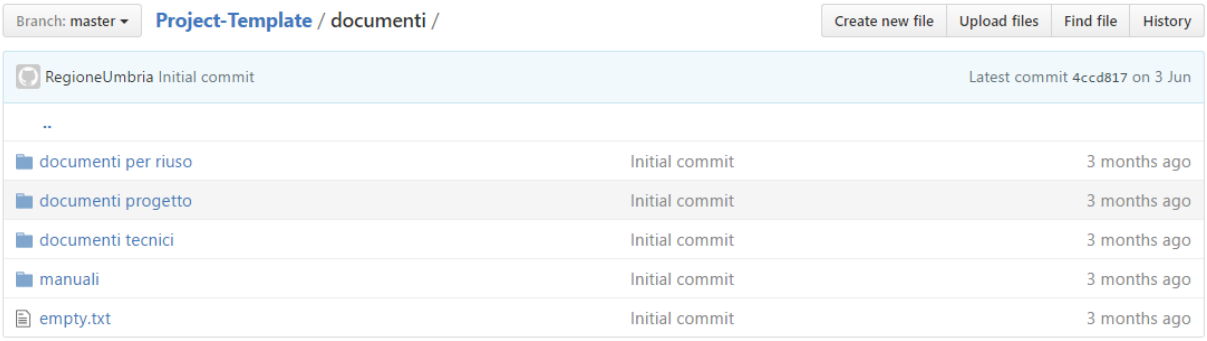
Al fine di facilitare la creazione di un progetto sul repository GitHub è stato creato un repository, denominato **Project-template**, che funge da modello avente una struttura qui di seguito riportata:



- Bin
  - Eseguibili;
  - Build.

- Documenti (documenti del progetto)
- src (sorgenti del progetto):
  - Src;
  - Test case;
  - Script:
    - Database;
    - Configurazione e varie;
    - Build script;
- tools

Navigando la struttura del repository, il folder ***documenti*** a sua volta è suddiviso nelle seguenti sotto-cartelle:



Dove:

- **documenti**
  - *Documenti per riuso*
  - *Documenti progetto*:
    - Documenti Prince 2 (output di fase);
    - Stati avanzamento;
    - Documenti di collaudo;
  - *Documenti tecnici*:
    - Specifica dei requisiti;
    - Documenti di architettura;

- Use case;
- Piano dei test;
- Riferimenti a standard e metodologie adottate;
- Documenti su struttura Database;
- Documenti su specifiche di colloquio con il sistema e tra il sistema e altri componenti esterni;

- *Manuali*:
  - Manuali di installazione;
  - Manuale utente;
  - Manuale di gestione del prodotto;
  - FAQ;

Ciascun progetto deve avere nella radice iniziale un file denominato README.md. Tale file viene visualizzato da GitHub al momento in cui si accede al progetto. Qui di seguito viene riportato un esempio:

Project name

Descrizione

Questa è una breve descrizione del progetto. La descrizione deve essere chiara con lo scopo di illustrare le finalità del progetto e l'ambito di intervento.

Struttura del repository

Il repository ha la seguente struttura

Folder	Descrizione
bin	Questo folder contiene i file compilati o binari.
documenti	Questo folder contiene la parte documentale del progetto. Il folder è suddiviso in sub folders per contenere documenti tra loro omogenei.
screenshots	Questo folder contiene alcuni screenshots delle schermate principali del prodotto in modo da dare a chi legge un'idea immediata della UI del prodotto
src	Questo folder contiene la parte del codice sorgente del prodotto. In questo folder vanno inseriti non solo il codice sorgente ma anche tutti gli scripts necessari alla creazione del database
tools	Questo folder contiene tutti gli eseguibili dei prodotti necessari al corretto funzionamento dell'applicativo e alla sua compilazione (es. maven, ant, ecc.)

Ambiente di sviluppo

In questa sezione vanno inserite tutte le informazioni relative all'ambiente di sviluppo utilizzato per il prodotto. In particolare va indicato con chiarezza sia il nome sia la versione dell'IDE utilizzata.

Licenza

In questa sezione va inserita la parte relativa alla licenza con cui si intende distribuire il codice. Nel caso in cui si preferisca utilizzare una file apposito (LICENSE.md) allora è necessario inserire il link a tale file.

Riferimenti

In questa sezione vanno inseriti i riferimenti alla persona/persone ovvero alla struttura che ha in carico la gestione del prodotto e può fornire informazioni utili.

Tale file deve essere scritto utilizzando Markup language. Per maggiori informazioni si può consultare il seguente link (https://guides.github.com/features/mastering-markdown/).

In generale, ciascun progetto deve avere un proprio repository così da poter gestire gli accessi allo stesso in maniera indipendente.

3.6 ELEMENTI MINIMI PER INSERIRE UN PROGETTO

Affinché un progetto possa essere inserito all'interno del repository contente i software messi a riuso dalla Regione Umbria è necessario che questo rispetti un insieme di regole riguardanti sia il codice sorgente sia la documentazione a corredo.

In particolare è necessario che:

- Il progetto possa essere distribuito secondo le licenze EUPL (si veda più avanti)
- Il progetto contenga il codice sorgente
- Il progetto contenga il seguente set minimo di documenti:

- Scheda AgID
- Documento architetturale
- Documento di specifica dei requisiti
- Manuale di installazione
- Manuale di configurazione

In deroga a quanto prima descritto, è possibile inserire nel repository anche i progetti che non rispettano i requisiti prima descritti purché questi siano progetti già rilasciati e installati in produzione.

3.7 MODALITÀ DI ACCESSO AI PROGETTI/REPOSITORY DI GITHUB

In generale, l'accesso ai progetti presenti su GitHub avviene in due modalità:

- ***Accesso libero***
- ***Accesso tramite autenticazione***

Le due modalità di accesso sono mutuamente esclusive per uno stesso progetto e ciascun progetto/repository può essere configurato in maniera indipendente dall'altro.

Nella prima modalità di accesso al progetto/repository, ovvero accesso libero, chiunque può connettersi a GitHub e scaricare i files del progetto, tramite un'operazione che viene chiamata *clone del repository*. E' evidente che in questa modalità non vi è controllo sugli accessi e il progetto può essere scaricato ed utilizzato da chiunque. In questa modalità di accesso il repository (ovvero il progetto) è ***pubblico***.

Nella seconda modalità, ovvero accesso tramite autenticazione, il progetto/repository è privato. Gli utenti, intesi come aziende, privati o pubbliche amministrazioni, che intendono accedere al repository devono prima registrarsi su GitHub e successivamente essere abilitati all'accesso del progetto di interesse. Tale abilitazione viene effettuata dal gestore del repository o dai soggetti abilitati a tale funzione.

3.7.1 GITHUB GESTIONE RUOLI

Github prevede tre ruoli fondamentali:

- *Owner*
- *Billing manager*
- *Member*

L'**Owner** è il proprietario del repository e dovrebbe sempre corrispondere ad un utente della Regione Umbria o di Punto Zero in maniera da assicurare il pieno controllo sul repository e sulle attività che

possono essere compiute su di esso.

Il **Billing manager** è colui che gestisce, ovvero delega, la gestione economica dell'account di GitHub.

Il ruolo **Member** è il ruolo di default.

3.7.2 GITHUB GESTIONE PERMESSI

Dal punto di vista dei permessi, GitHub supporta tre tipi di permessi per gli utenti che appartengono ad una organizzazione (nel caso in esame *RegioneUmbria*):

- *Read*
- *Write*
- *Admin*

che vengono mappate

Repository action	Read permissions	Write permissions	Admin permissions	Owner permissions
Pull (read), push (write), and clone (copy) *all repositories* in the organization				**X**
Promote organization members to team maintainer				**X**
Convert organization members to outside collaborators				**X**
Create repositories (see "Creating repositories" for details)	**X**	**X**	**X**	**X**
Delete repositories (see "Deleting repositories" for details)			**X**	**X**
Change a repository's			**X**	**X**

Repository action	Read permissions	Write permissions	Admin permissions	Owner permissions
settings (see "Changing repository settings" for details)				
Change a repository's visibility			**X**	**X**
Transfer repositories into, and out of, the organization account			**X**	**X**
Add a repository to a team (see "Adding a repository to a team" for details)			**X**	**X**
Add outside collaborators to a repository			**X**	**X**
Remove outside collaborators from a repository			**X**	**X**
Pull from (read) the team's assigned repositories	**X**	**X**	**X**	**X**
Push to (write) the team's assigned repositories		**X**	**X**	**X**
Fork (copy) the team's assigned repositories	**X**	**X**	**X**	**X**
Send pull requests from forks of the team's assigned repositories	**X**	**X**	**X**	**X**

Repository action	Read permissions	Write permissions	Admin permissions	Owner permissions
Merge and close pull requests		X	X	X
Merge pull requests on protected branches, even if there are no approved reviews			X	X
Submit reviews on pull requests	X	X	X	X
Submit reviews that affect a pull request's mergeability		X	X	X
Open issues	X	X	X	X
Close, reopen, and assign issues		X	X	X
Close issues they opened themselves	X	X	X	X
Apply labels and milestones		X	X	X
Have an issue assigned to them	X	X	X	X
Create and edit releases		X	X	X
View draft releases		X	X	X
View published releases	X	X	X	X
Edit and delete their own comments on commits, pull requests, and issues	X	X	X	X

Repository action	Read permissions	Write permissions	Admin permissions	Owner permissions
Edit and delete anyone's comments on commits, pull requests, and issues		X	X	X
Edit wikis	X	X	X	X
Create statuses		X	X	X

L'Owner (proprietario) del repository ha i permessi su tutte le possibili azione che possono essere effettuate sul repository.

3.7.3 GITHUB GESTIONE COLLABORATORI ESTERNI

GitHub, come detto, prevede diversi profili e modalità di accesso al progetto/repository con differenti autorizzazione. In generale, il profilo di accesso consigliato da parte di un soggetto (privato o PA) non appartenente all'organizzazione *Regione Umbria* è quello di "***external collaborator***". Con questo profilo, è possibile accedere in sola lettura al progetto ed eventualmente fare il clone del repository nel caso in cui l'utente sia interessato.

Utilizzano il profilo di ***external collaborator*** è possibile organizzare e controllare gli accessi per i singoli progetti senza che questi utenti abbiano gli stessi ruoli e gli stessi privilegi degli utenti appartenenti all'organizzazione Regione Umbria.

Una volta che l'utente ha raggiunto il progetto/repository ha a disposizione diverse funzionalità ed in particolare può o semplicemente navigare all'interno della struttura del progetto oppure fare il download dell'intero progetto tramite la funzione di clonazione del repository. Una volta clonato il repository può essere gestito e modificato in maniera indipendente dal repository originale. Questo, per esempio, consente agli utenti di poter provare quanto rilasciato su GitHub.

Per consentire all'utente di avere un quadro complessivo del progetto e al fine di permettere a chi accede di orientarsi è necessario che ciascun progetto presente sul repository abbia un testo di presentazione (denominato in GitHub ***readme.md***) che contenga la descrizione del progetto, le sue finalità e quali processi supporta.

3.8 MINI-SITO PROGETTO

GitHub permettere di gestire due tipi di siti:

- Sito complessivo legato all'utente
- Sito per il repository

Nel primo caso, il sito web è legato alla username dell'utente con cui è stata fatta la registrazione iniziale (RegioneUmbria). Tale sito web serve per permettere di informare gli utenti che accedono ai vari repository gestiti dalla Regione Umbria. Questo sito è pubblico e dovrebbe essere utilizzato per informazioni di carattere generale e per mostrare la lista dei progetti gestiti. In particolare è utile per quei progetti memorizzati su repository privati che non possono essere acceduti senza previa autenticazione. Tali repository risultano nascosti agli utenti non loggati, quindi tramite tale sito web è possibile mostrare una breve descrizione dei progetti privati così che se l'utente è interessato può richiedere l'accesso al repository contente il progetto.

A tale scopo è stata creata una semplice pagina web, scritta in HTML, denominata ***index.html,*** che contiene una lista dei progetti attualmente gestiti tramite GitHub, raggiungibile all'indirizzo:

https://regioneumbria.github.io/

GitHub permette di gestire, inoltre, per ciascun progetto delle pagine Web creando così un sito di supporto al progetto stesso.

La costruzione di un "mini-sito" illustrativo del progetto, delle finalità e dell'ambito di applicazione del progetto stesso è utile al fine di permettere una più rapida comprensione dei prodotti presenti nel repository.

Tale mini-sito deve contenere non solo la descrizione del progetto ma anche tutte quelle informazioni che possono essere utili a far comprendere il funzionamento del progetto e i casi d'uso tipici, così che eventuali PA che vogliono riusare il prodotto presente nel repository possano valutare se questo corrisponde alle proprie esigenze, prima di procedere con il calcolo dell'indice di riusabilità come previsto dalle "Linee guida per l'inserimento ed il riuso di programmi informatici o parti di essi pubblicati nella 'Banca dati dei programmi informatici riutilizzabili' di DigitPA" .

In tale mini-sito si potrà fare riferimento direttamente agli items presenti nel repository così che l'accesso alle informazioni memorizzate sia più semplice ed immediato.

L'utilizzo di strumenti quali Jekyll favoriscono inoltre la gestione dei template delle pagine così che i diversi progetti abbiano una navigazione simile favorendo l'accesso da parte degli altri utenti/aziende.

In aggiunta a quanto indicato è opportuno prevedere un apposito canale sul sito istituzionale della Regione Umbria che presenti il repository del codice sorgente, le sue finalità e le modalità di utilizzo dei prodotti software presenti nel repository stesso. Il canale rimanderà il download dei progetti ai singoli repository presenti in Github.

3.9 DISTRIBUZIONE IMMAGINE DEL PRODOTTO TRAMITE DOCKER

Al fine di permettere all'utente di provare l'applicativo in maniera semplice e rapida è possibile aggiungere al repository GitHub un'immagine compatibile con Docker.

Docker è l'applicazione leader per la gestione di "container" applicativi che eseguono processi in ambienti isolati.

Docker permette di pacchettizzare un applicativo utilizzando un proprio filesystem che contiene tutto

quello che serve all'applicativo stesso per funzionare, come per esempio librerie, tools ecc. Tutto questo può essere installato, come una singola unità software, direttamente su un server/pc.

Tale sistema permette di semplificare notevolmente il processo di installazione di un nuovo prodotto al fine di valutarne le caratteristiche. Un ulteriore vantaggio è data dal fatto che l'utente, che intende provare l'applicativo, non ha bisogno di conoscere i diversi passi di installazione e di configurazione del prodotto stesso.

Docker, inoltre, è una piattaforma open-source e gratuita e perfettamente integrabile con Github.

4 GESTIONE DEL REPOSITORY

4.1 GESTIONE DEI BRANCH PER DIVERSE "VERSIONI"

Può accadere, specialmente per tutti gli applicativi che hanno una diffusione sui comuni, che vi siano versioni differenti di uno stesso prodotto adattate alle necessità di un singolo comune.

In questo caso lo sviluppo/rilascio non può essere portato avanti sulla "Active Development line" in quanto la funzionalità o le funzionalità che vengono implementate non verranno rilasciate nell' applicativo nel suo complesso, ma sono customizzazioni specifiche per ogni singolo Comune/Ente. Anche se questa pratica è sconsigliata, tuttavia tale necessità può sorgere. In tal caso sarà necessario utilizzare un branch specifico applicando il pattern (task branch).

In generale tale pattern risulta particolarmente utile quando si verifica almeno una di queste condizioni:

- Un gruppo di lavoro deve lavorare/ha lavorato su un task che diverge dalla linea principale dello sviluppo;
- Quando vi è la necessità di iniziare rilasciare una nuova release, ovvero nuove funzionalità, prima che la release corrente venga consolidata.

Tenendo conto che l'utilizzo di branch è una pratica che può portare a numerosi inconvenienti è necessario utilizzare tale approccio solo quando i benefici attesi siano superiori all'aumento di lavoro per gestire il branch ed il successivo riallineamento del codice.

In tale ottica, un continuo allineamento del codice con la main line di sviluppo è auspicabile in modo da ridurre al minimo i problemi nella successiva fase di merge.

Un branch di tale natura, alla fine della fase di sviluppo, avrà due possibili alternative:

- Viene abbandonato;
- Viene integrato nella mainline aggiungendo le nuove funzionalità implementate nella versione principale del prodotto.

A queste due vie canoniche si aggiunge anche una terza via che prevede che il branch non venga né abbandonato né integrato con la main line ma dia origine ad una versione modificata del prodotto (versione divergente). E' evidente che tale strada non può essere perseguita senza notevoli difficoltà nelle gestione del prodotto stesso e quindi è necessario utilizzarla solo quando le funzionalità implementate non possono essere integrate.

4.2 RILASCIO DI UNA NUOVA VERSIONE

Il rilascio di una nuova versione è un'attività molto comune nello sviluppo di un nuovo prodotto.

Il rilascio viene gestito attraverso il congelamento del codice ad una data prefissata. Tale operazione prende il nome di **baseline**, attività attraverso la quale le diverse versioni degli items vengono congelate e raggruppate simbolicamente. La release è la promozione della baseline cioè la baseline viene resa visibile all'esterno dell'organizzazione che l'ha gestita. Tale release sarà visibile nel

repository di Github.

La release viene etichettata con un numero la cui struttura riflette la tipologia di interventi che sono stati applicati al prodotto. La numerazione è del tutto arbitraria e ciascuna organizzazione (ovvero fornitori esterni) può adottare una qualsiasi tipologia di numerazione purché venga normata e definita e sia univoca per ciascuna release.

Il rilascio di una nuova versione deve essere notificata al catalogo AGID in modo che il codice sia sempre aggiornato.

Eventuali bug fixing su una determinata versione devono essere gestiti tramite la creazione di un branch di sviluppo. Al termine della fase di sviluppo le modifiche apportate devono essere rilasciate nella versione corrente e verrà richiesta una nuova baseline a una successiva release la cui numerazione rispecchierà l'intervento o gli interventi effettuati.

4.3 GESTIONE DEGLI ISSUE

Durante l'utilizzo di un prodotto possono manifestarsi delle anomalie di funzionamento.

Il sistema di gestione del repository permette la gestione e la tracciatura degli issue. Tramite la gestione degli issue è possibile tracciare i task, nuove funzionalità ed i bug. A ciascun issue, il sistema assegna un numero univoco.

Il sistema consente l'apertura di un issue tramite un'apposita interfaccia. Le anomalie segnalate dagli utenti del prodotto o da altri sviluppatori devono essere gestite tramite tale sistema. Il software di gestione del repository deve permettere di catalogare e organizzare gli issues attraverso l'uso di Milestone, labels a assegnatari. La milestone stabilisce in quale release futura il bug verrà risolto o la nuova funzionalità verrà implementata. L'utilizzo dei labels permette di categorizzare gli issue ed è un altro modo per organizzare le nuove features o bugs.

Infine ciascun issue può essere assegnato ad un membro del team di lavoro per il successivo sviluppo ed implementazione. La gestione degli issues tramite GitHub risulta particolarmente utile al fine di evidenziare eventuali anomalie presenti nel software così che l'utente finale possa subito avere un riscontro su eventuali problemi(***known issue***).

Anche se, come detto, GitHub è il repository "statico" delle soluzioni software, è importante tracciare gli issue su tale sistema per permettere all'utente, che utilizza il prodotto/progetto, di avere un quadro d'insieme del prodotto, sul grado di attività della "comunità" che lavora la progetto stesso.

4.4 APPORTO ESTERNO AL REPOSITORY

Nell'ottica di condivisione della conoscenza sia dal punto di vista del software che da quello documentale, organizzazioni esterne ("community") possono contribuire a vario titolo allo sviluppo di quanto già realizzato, sia dal punto di vista software, di conoscenza o best practice.

Il sistema di gestione del repository favorisce questo approccio collaborativo ad un progetto tramite l'utilizzo di *pull-request*. Ciò consente di informare i gestori del progetto che alcune modifiche sono state apportate o al codice sorgente o alla documentazione o più in generale ad uno o più items del progetto stesso. Tale modifiche rimangono in attesa di accettazione fino a quando il gestore del

progetto provvederà ad accettarle o a rifiutarle.

In questo caso, è opportuno verificare che le modifiche apportate siano corrette e che non abbiano alterato il codice. A tale scopo esistono, dei tool automatici che permettono di effettuare la compilazione in background del progetto e di verificare se le modifiche sono compatibili con il progetto stesso.

4.5 CONTROLLO DELLA QUALITÀ DEI PRODOTTI RIUSABILI

Al fine di automatizzare e garantire che le release rilasciate siano in linea con i standard qualitativi attesi per un prodotto riusabile è necessario provvedere a evolvere gli attuali standard di sviluppo oramai obsoleti e non più in linea con le attuali tecnologie.

Oltre al mero rilascio del codice sorgente è necessario fornire tutti gli strumenti che permettano di valutare la qualità del prodotto e del codice. Tale valutazione può essere fatta a diversi livelli e con diversi strumenti open-source. Tutto questo dà valore aggiunto al progetto e garantisce il soggetto che prende in riuso il prodotto della qualità di quanto rilasciato e delle modalità seguite nello sviluppo.

Esistono diversi livelli di intervento:

- Unit Test
- Integration test
- Front-end Test
- Performante/Stress test

Oltre a questi livelli di test è necessario tracciare la qualità del sorgente e dell'intero processo alla base del progetto.

4.5.1 QUALITÀ DI BASE (UNIT TEST)

Gli unit test sono i test a più basso livello delle singole unità atomiche alla base del prodotto. Per unità atomiche si intendono le singole funzioni (nel caso di linguaggi procedurali), singole classi (nei linguaggi ad oggetti). Per questi tipi di test si possono utilizzare prodotti open-source/free diventati oramai dei punti di riferimento: JUnit, Mockito.

4.5.2 QUALITÀ DEI LAYER (INTEGRATION TEST)

La valutazione della qualità dei diversi layer architetturali di un prodotto è fondamentale per comprendere anche la qualità dell'intero prodotto. La sua valutazione è importante anche per capire le possibilità di integrazione che un prodotto può offrire. I test di integrazione valutano la qualità di quanto realizzato a livello di integrazione fra i diversi layer applicativi garantendo da un lato la separazione logica dall'altro che i diversi livelli siano tra loro correttamente integrati. A tale scopo uno tra i possibili prodotti è Arquillian

4.5.3 TEST FRONT-END

La qualità del front end riveste una particolare importanza in quanto è il punto di contatto tra il prodotto e l'utente. Al fine di garantire standard qualitativi adeguati è necessario effettuare test massivi su questo layer architetturale. A tale scopo può essere utilizzato Selenium.

4.5.4 PERFORMANCE

Un aspetto fondamentale per la valutazione di un prodotto è la valutazione del carico che è in grado di gestire e l'eventuale possibilità di poter scalare sia verticalmente che orizzontalmente. Tali informazioni possono essere dedotte utilizzando i test di performance. Questo aspetto è di solito molto trascurato ma all'atto pratico è quello che incide maggiormente sulla qualità percepita dall'utilizzatore finale.

Al fine di poter valutare la qualità prestazionale di un prodotto esistono tool in grado di "stressare" un applicativo sotto diversi punti di vista (es: richieste concorrenti, sessioni, memoria sotto stress, ecc.). Uno tra questi prodotti è JMeter.

4.5.5 QUALITÀ DEL PROCESSO E AFFIDABILITÀ DELLA RELEASE

Al fine di poter valutare, la qualità dell'intero processo dal progettazione al rilascio della release è necessario che i prodotti rilasciati prevedano l'integrazione con sistemi automatici di compilazione e test. Tali strumenti consentono anche di gestire con più semplicità i rilasci in quanto garantiscono l'affidabilità di quanto rilasciato.

Un esempio è Travis-ci (https://travis-ci.org/) che permette di automatizzare il processo di build del progetto (e dei test case) e può essere facilmente integrato con GitHub. Questo sistema è gratuito per i repository pubblici, cioè per tutti i repository per cui non è previsto nessun controllo di accesso mentre ha un costo di utilizzo per i repository privati.

Il vantaggio di utilizzo di tale sistema è evidente, infatti questo consente di avere la sicurezza che il codice rilasciato sul repository sia corretto e che sia possibile produrre una versione compilata.

5 PROCESSI A SUPPORTO DELLA QUALITÀ DEL PRODOTTO: CONTINOUS DELIVERY – CONTINUOUS INTEGRATION

Uno degli aspetti che maggiormente interessa nel rilascio di un prodotto è tenere consistenti le versioni del prodotto e ridurre il più possibile il tempo tra il momento dell'idea di un prodotto e il successivo rilascio.

A tale proposito alcune tecniche e best practice hanno introdotto nuove metodologie e nuovi paradigmi:

- ***Continuous Delivery (CD)***
- ***Continuous Integration (CI)***

Tramite l'utilizzo di questi due paradigmi, i team di sviluppo riducono il tempo per la produzione del componente software e vi è certezza che il software rilasciato sia funzionante e rispetti gli standard prima descritti.

Tramite l'automazione dell'intero processo (build, deployment, test, e release) è possibile ottenere un elevato grado di affidabilità dei prodotti..

Quanto fin qui esposto risulta indipendente dal sistema utilizzato per il versionamento dei sorgenti. L'utilizzo di altri tools e risorse porta ad una migliore implementazione dei pattern prima descritti; qui di seguito viene riportata una breve lista di applicativi che potrebbero risultare utili nel contesto fin qui descritto:

Continuous Integration Tools

• *Jenkins (https://jenkins.io)*

• *Hudson*: http://hudson-ci.org

Build Tools

• *Maven*: http://maven.apache.org

• *Ant*: http://ant.apache.org

• *Buildr*: http://buildr.apache.org

• *Gradle*: http://www.gradle.org

E' importante che i software rilasciati soggetti a riuso implementino un sistema di build automatico utilizzando uno dei prodotti sopra riportati in modo da ridurre il più possibile l'intervento manuale.

6 LICENZE PROGETTI

Al fine di permettere ad altri soggetti l'utilizzo di un progetto sia questo di tipo software o un insieme di buone pratiche, è necessario che il progetto sia distribuito utilizzando una licenza di riferimento che delimiti con chiarezza gli ambiti di utilizzo.

Prima di descrivere le possibili licenze da utilizzare è opportuno definire brevemente il quadro normativo che regola l'utilizzo e la distribuzione del software. Questo è importante per verificare se il software che si intende mettere a riuso è libero da licenze di copyright e può essere distribuito:

- La legge del 22 aprile 1941, n.633, relativa al diritto d'autore, che in alcuni articoli, delinea la disciplina giuridica relativa alla tutela del software
- Il D.Lgs. 10 febbraio 2005, n. 30 - Codice della proprietà industriale; - gli articoli da 32 a 43 del Decreto del Presidente del Consiglio dei Ministri 6 agosto 1997, n.452,
- Regolamento relativo alla locazione e all'acquisto di apparecchiature informatiche, nonché alla licenza d'uso dei programmi.

In ambito europeo, la tutela del software è oggetto delle direttive 91/250/CEE, 92/100/CEE, 93/98/CEE e 2009/24/CE.

Per quanto riguarda le licenze di utilizzo, invece, queste possono essere classificate in due macro categorie:

- Licenze d'uso per software proprietari
- Licenze d'uso per software liberi

Tali licenze regolamentano le modalità di distribuzione ed uso del software.

6.1 LICENZA D'USO PER SOFTWARE PROPRIETARI

Rientrano in questa categoria, tutti quei prodotti software per i quali i titolari impongono limitazione e condizioni nell'utilizzo del software stesso, tra cui ricordiamo:

- La licenza concessa in modo non esclusivo
- La licenza può limitare l'uso ad una singola copia o ad numero predeterminato di copie
- L'impossibilità di modificare, accedere e visionare il codice sorgente
- L'impossibilità di distribuire il prodotto software

A mero titolo esemplificativo, ma non esaustivo, rientrano in questa categoria i prodotti Microsoft Office, i sistemi operativi Windows (nelle sue versioni), Mac OS X, ecc. Vi sono, inoltre, alcuni software che, pur essendo dichiarati liberi, hanno alcune limitazioni. In questo caso, tali prodotti vengono distribuiti utilizzando una doppia licenza (es: MySQL, Alfresco,ecc.)

6.2 LICENZA D'USO PER SOFTWARE LIBERI

In generale, vi sono due definizioni di software libero, spesso usate in modo intercambiabile: "software libero" e "a sorgente aperto". Sebbene, tali due definizioni si riferiscono a concetti diversi, dal punto di vista operativo possono essere usate in maniera intercambiabile. La Free Software Foundation (FSF) è alla base della definizione del concetto di software libero. Tale definizione ha come fulcro la libertà degli utenti e della comunità. In altre parole, gli utenti hanno la libertà di eseguire, copiare, distribuire, studiare, cambiare e migliorare il software. Più in generale un software è libero quanto ha gli utenti hanno le seguenti libertà:

Grado	Definizione
libertà 0	Libertà di eseguire il programma, per qualsiasi scopo
libertà 1	Libertà di studiare come funziona il programma e di modificarlo in modo da adattarlo alle proprie necessità. L'accesso al codice sorgente ne è un prerequisito.
libertà 2	Libertà di ridistribuire copie in modo da aiutare il prossimo.
libertà 3	Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti da voi apportati (e le vostre versioni modificate in genere), in modo tale che tutta la comunità ne tragga beneficio. L'accesso al codice sorgente ne è un prerequisito.

6.3 APPLAZIONE DELLE LICENZE

Un aspetto importante da considerare prima di valutare se il progetto può essere messo a riuso è se tale progetto utilizza moduli coperti da copyright o soggetti a limiti di utilizzo, come per esempio uno o più moduli soggetti a licenze d'uso con limitazioni (già descritte nei paragrafi precedenti). In tal caso, il progetto non può essere messo liberamente sul repository. E' indispensabile, quindi, che la Regione (ovvero la società che ha realizzato il progetto) alleghi al progetto una dichiarazione di conformità dove venga specificatamente dichiarato che il progetto può essere distribuito liberamente e che questo non fa uso di librerie di terze parti coperte da copyright.

Una volta accertato che il progetto può essere distribuito liberamente è necessario valutare quale licenza utilizzare.

In generale, il software, e più in generale il progetto, devono essere distribuiti utilizzando una licenza open source. La particolarità delle licenze open source, come descritto precedentemente, è quello che gli autori del progetto/software permettono la modifica del progetto stesso allo scopo di permettere l'evoluzione del progetto e una sua più ampia diffusione.

Vi sono attualmente diverse licenze che declinano in diverso modo i principi alla base dell'open source, ciascuna di queste licenze stabilisce dei limiti di utilizzo e di distribuzione del software o della relativa documentazione.

La commissione europea ha redatto una nuova licenza open source che ben si adatta alle pubbliche amministrazione che intendono rendere disponibile il proprio software/documentazione.

La licenza si chiama **EUPL** (*European Union Public Licence*). Tale licenza concede al licenziatario una licenza mondiale, non esclusiva, gratuita con la possibilità di prevedere sub-licenze.

Tale licenza dà il diritto di utilizzare, modificare, distribuire un prodotto software o documentale e di

concedere in sub-licenza tali prodotti. Questo tipo di licenza è stato creato principalmente per le pubbliche amministrazioni europee al fine di uniformare i diritti d'autore nei diversi paesi europei.

Questa licenza è adatta ad essere utilizzata per i progetti rilasciati sul repository.

7 MODELLO DI GESTIONE DEL REPOSITORY

Al fine di definire il modello di gestione del repository è necessario, prima, individuare i possibili attori che in vario modo interagiscono con il repository.

Gli attori possono essere suddivisi in attori appartenenti agli Enti pubblici e quelli appartenenti alle aziende private. Per quanto riguarda gli attori "pubblici" questi possono essere divisi in:

- *Cedenti:* soggetti che hanno sviluppato una soluzione e la mettono a disposizione sul repository (generalmente la Regione Umbria);
- *Riusanti:* soggetti che accedono al repository e scelgono di adottare una delle soluzioni offerte;

Oltre agli Enti, sono presenti altri attori "privati" che offrono beni servizi e che accedono al repository sia per prelevare il prodotto sia per contribuire a vario titolo all'evoluzione del prodotto stesso.

Il ruolo di coordinare le attività di gestione e di "animazione" del repository spetterà alla Regione Umbria mentre il coordinamento tecnico Centrale alla Società Punto Zero SCARL. Quest'ultima, in sinergia con la Regione, ha il compito di animare gli utenti ed i contributori alle soluzioni software e alle buone pratiche, cercando di estendere il numero dei contributori; rifacendosi al modello delle *community open source* svolge inoltre il ruolo di riferimento delle comunità degli utenti (anche non ICT) di ognuna delle suddette soluzioni o buone pratiche.

Peraltro, in prima battuta, la Regione Umbria ha già disciplinato nella Delibera oggetto del presente Progetto un processo di conferimento a Punto Zero delle funzioni svolte dall'ex SIR nella materia dei prodotti già individuati in riuso e presi in carico dalla Regione a seguito della liquidazione ai sensi dell'art. 12 della L.R. n.9/2014 del SIR. In questo modo, viene autorizzata Punto Zero, *in house* della Regione, anche alla pubblicazione del codice sorgente, della documentazione e degli altri contenuti, sotto licenza " *copyleft* " EUPL e CC-BY, o similari, compatibili con la l.r. 25 luglio 2006 n.11, nel " repository regionale del codice sorgente e delle buone pratiche", liberamente accessibile in internet, in modo tale da favorire la diffusione e la collaborazione, ai sensi dei punti 5.10 e 6.8 del "Disciplinare per l'attuazione della legge regionale n. 9/2014" ex DGR 22 dicembre 2014, n. 1778. Allo stesso modo viene dato ad Punto Zero il mandato di individuare altre soluzioni e di procedere alla loro.

Accanto alla definizione della *governance* del repository nel suo complesso, è necessario prevedere delle linee di governo delle soluzioni sottostanti le buone pratiche. Il tema della *governance* delle Linee di Soluzione (o Progetti).

Le linee guida AgID, classificano le tipologie di riuso del software in:

- riuso in cessione semplice in cui una Amministrazione cede completamente l'applicativo a un'altra;
- riuso con gestione a carico del cedente in cui oltre a cedere l'applicativo, l'Amministrazione proprietaria del software si fa carico della manutenzione dello stesso;
- riuso in *facility management* in cui l'Amministrazione cedente si fa carico oltre che della manutenzione anche della predisposizione e gestione dell'ambiente di esercizio;

- riuso in *Application Service Providing* (ASP) in cui è un soggetto terzo a farsi carico della manutenzione e dell'esercizio del software per più Amministrazioni, che riconoscono il corrispettivo in relazione al servizio ricevuto.

In funzione dei poteri decisionali che resteranno in capo alla Pubblica Amministrazione che cede la soluzione si potranno quindi configurare degli scenari di riuso con un differente livello di centralizzazione, quali:

- software di proprietà della Regione concesso a riuso passivo;
- software di proprietà della Regione concesso a riuso con esigenze di evoluzione da parte dell'Ente riusante;
- software NON di proprietà della Regione preso dalla stessa come riuso passivo;
- software NON di proprietà della Regione preso dalla stessa come riuso in cui saranno generate evoluzioni o personalizzazioni del prodotto.

Per ognuna di queste configurazioni deve essere individuata la più opportuna formula di *governance* del prodotto. In questa sede progettuale, avendo in carico la definizione del modello, è possibile dare le seguenti direttive di funzionamento generale della *governance* dei prodotti sottostanti le buone pratiche. Sulla base di queste, verrà poi costruito il dettaglio della governance delle Linee di soluzione.

- Il *Repository* che detiene la versione aggiornata del software a riuso è univoco e in carico alla Regione o all'Amministrazione cedente.
- Il centro di competenza per la gestione del software a riuso, che ricopre la funzione di "manutentore" del prodotto stesso è univoco e definito come "*PA Leader*", identificato nell'Amministrazione cedente salvo la sua espressa decisione di non ricoprire questo ruolo. L'Amministrazione cedente può dichiarare però di non voler ricoprire il ruolo di "*PA Leader*"; in questo caso l'Amministrazione riusante può farsi carico di questo ed essere riconosciuta da parte di altri Enti come manutentrice del prodotto
- La *PA Leader* può indicare un'eventuale manutentore esterno, anche privato, e metterlo in contatto con le Amministrazioni riusanti o dare tutte le informazioni perché queste possano accedere ai suoi servizi nelle forme previste dalla legge.
- Prevedere l'opportunità per l'Amministrazione cedente che non si occupa della manutenzione diretta del software di affidarla invece ad un Soggetto terzo privato, che sia stato oggetto di selezione e si sia attestato per il prodotto su MEPA/CONSIP e abbia avuto dall'Amministrazione proprietaria (cedente) tutte le certificazioni di servizio necessarie per la proposizione della sua offerta. In questo caso tra gli accordi di servizio l'Amministrazione cedente, oltre a richiedere la pubblicazione su Consip dei costi di servizio, deve anche chiedere al Soggetto privato la dichiarazione di disponibilità a formare Soggetti indicati dalle Amministrazioni riusanti con un prospetto dei costi a catalogo.
- L'Amministrazione cedente può formulare un catalogo del modello di cessione e dei servizi previsti con il riuso da rendere disponibile anche nel Repository.

Su queste problematiche è chiaro che sarà necessario un approfondimento complessivo che riguarda gli aspetti normativi, gli indirizzi nazionale e i processi di gestione del riuso attivo (es. progetto PAOC).

Dal punto di vista più strettamente operativo, l'accesso di tali attori al repository si concretizza in una delle modalità prima previste attribuendo a ciascun soggetto interessato ad uno o più progetti presenti nel repository stesso.

8 FIGURE PROFESSIONALI COINVOLTE

Qui di seguito viene riportata la tabella delle figure professionali coinvolte nel rilascio di una nuova versione di uno dei prodotti messi a riuso:

- Responsabile del repository;
- Responsabile del progetto (da mettere a riuso);

Il rilascio di un nuovo prodotto nel momento in cui il responsabile del progetto rilascia una nuova versione del prodotto:

Passo	Responsabile del progetto	Responsabile Repository
1	Rilascia una nuova versione	
2	Avvisa il responsabile del repository e invia il pacchetto	
3		Preleva il pacchetto
4		Estrae il pacchetto
5		Esegue la build del progetto
6		Effettua il commit del pacchetto su repository GitHub
7	Fornisce la nuova descrizione del progetto	
8	Fornisce il numero della nuova versione	
9		Tagga gli items con la nuova versione
10		Informa dei nuovi puntamenti alla nuova versione (es. siti web ecc.)