Guida allo sviluppo e gestione di software libero

Release stabile

italia

31 mag 2022

Indice dei contenuti

Buone pratiche, suggerimenti e tecniche utili allo sviluppo di software libero per chi progetta, realizza e gestisce i servizi pubblici digitali italiani.

Questo documento si pone l'obiettivo di fornire agli attori della trasformazione digitale delle buone pratiche di gestione di progetti e indicare strumenti che possono essere utilizzati nello sviluppo di software libero.

CAPITOLO 1

Introduzione

Developers Italia[2] è il punto di riferimento per lo sviluppo dei servizi pubblici italiani. Vogliamo aiutare a costruire una Pubblica Amministrazione digitale collaborativa e indipendente, che aiuti il Paese a generare innovazione e che possa concentrarsi sul fornire servizi alle persone grazie a soluzioni software sicure, riutilizzabili e tecnologicamente eccellenti. Mettiamo a disposizione strumenti tecnici, software, librerie, occasioni di incontro, forum di discussione, eventi di formazione e molto altro ancora per aiutare chi «fa» la trasformazione digitale della Pubblica Amministrazione a fare di più, meglio e con meno costi. Molti di questi strumenti sono offerti in collaborazione con Designers Italia[3], che mira ad essere l'anello di congiunzione tra tutti gli attori coinvolti nella trasformazione digitale di servizi delle Pubbliche Amministrazioni italiane e diffondere buone pratiche di progettazione e un approccio virtuoso per la realizzazione di servizi pubblici a misura di cittadino.

1.1 Cos'è la guida allo sviluppo e alla gestione di software libero

Questa guida è una raccolta di buone pratiche utili a **supportare le organizzazioni pubbliche nello sviluppo e nella gestione del software** e delle politiche ad esso connesse. Chiunque sviluppi o gestisca software per scopi pubblici può usare questo documento per realizzare **servizi pubblici di alta qualità** a un costo minore, il cui sviluppo presenti meno rischi sia in termini operativi che di lock-in verso un particolare fornitore, rispettando i principi imposti dalla legge di *economicità e di efficienza, tutela degli investimenti, riuso e neutralità tecnologica* (art. 68 del CAD - Codice dell'amministrazione digitale).

1.2 A cosa serve questa guida

Imparare a conoscere il software libero è oggi fondamentale per lavorare in un ente pubblico. Ricordiamo infatti che il CAD impone alle pubbliche amministrazioni di adottare software libero ogni qual volta sia possibile, e l'eventuale non adozione deve essere adeguatamente motivata da un atto formale depositato e protocollato all'interno della «valutazione comparativa». Alle disposizioni del CAD si sono aggiunte nel 2019 le «Linee guida su acquisizione e riuso di software per le pubbliche amministrazioni» che hanno introdotto un nuovo modello allineato alle migliori pratiche internazionali per acquisire, modificare e condividere soluzioni software.

[2] https://developers.italia.it
[3] https://designers.italia.it/

Lo scopo di questa guida **è quello di supportare concretamente le Pubbliche Amministrazioni italiane** in un contesto nuovo e potenzialmente complesso, offrendo **soluzioni pratiche apprese sul campo**. Il fine è di massimizzare la semplicità e l'efficacia delle iniziative volte a utilizzare software libero già esistente, creare (o co-creare) nuovi strumenti per le esigenze pubbliche, condividere esperienze, risorse e attrarre nuovi sviluppatori grazie all'utilizzo di sistemi di sviluppo e collaborazione pubblici.

Abbiamo provato a fondere l'expertise e la conoscenza del *software* libero delle esperte e degli esperti del Dipartimento per la trasformazione digitale (DTD) con l'importante esperienza e i bisogni di coloro che operano a vario titolo nelle pubbliche amministrazioni.

1.3 Una breve panoramica

In questa guida vogliamo fornire una panoramica complessiva di tutti gli elementi in gioco, e ci soffermeremo in particolare sui seguenti aspetti:

- dove trovare i migliori strumenti di software libero e in base a quali criteri valutarli
- i modelli di business del software libero e il loro valore per i fornitori e per la Pubblica Amministrazione;
- le migliori tecniche per sviluppare software libero e per interagire con altri soggetti interessati allo sviluppo;
- quali aspetti cambiano e a quali aspetti fare maggiore attenzione per migliorare le relazioni tra Pubblica Amministrazione e i fornitori;
- gli strumenti di gestione del software utili per migliorare i servizi pubblici;
- le basi di codice che possono essere riutilizzate in contesti diversi e gestite in modo collaborativo;
- quali tecniche utilizzare per aiutare a ridurre il debito tecnico e il rischio di fallimento del progetto;
- come avere un maggior controllo sui propri sistemi IT.

1.4 A chi si rivolge la guida

Protagoniste di questo documento sono le **Pubbliche Amministrazioni** e le **buone pratiche** che abbiamo imparato insieme a loro durante le nostre attività di supporto. La guida è suddivisa in tre capitoli, ognuno dei quali è dedicato a uno specifico profilo professionale che può beneficiare dell'uso di buone pratiche nella gestione del software pubblico.

I tre principali capitoli di questa guida si rivolgono quindi a: **chi ha responsabilità politiche**, **chi amministra nel pubblico** e **chi sviluppa servizi pubblici.**

Definiamo così i ruoli:

- **chi ha responsabilità politiche:** persone coinvolte nella definizione delle politiche e delle regole che vengono applicate nello sviluppo di servizi e soluzioni software per la pubblica amministrazione, indirizzandone l'attuazione. Sono coloro che definiscono le priorità e gli obiettivi dei progetti e di norma sono meno interessati agli aspetti tecnologici delle soluzioni realizzate;
- **chi amministra nel pubblico:** persone responsabili delle amministrazioni pubbliche che gestiscono i progetti e garantiscono l'esecuzione nel rispetto di tempi e costi previsti, si relazionano con gli stakeholder e sono responsabili del servizio una volta rilasciato;
- **chi sviluppa:** persone che si occupano di creare il software libero che implementa i servizi. Hanno solitamente un profilo molto tecnico e il loro lavoro ha un impatto diretto sulla qualità dei servizi.

1.4.1 Introduzione al software libero nella Pubblica Amministrazione

Che cos'è il software libero

Il software si definisce libero se il suo codice sorgente è distribuito agli utenti ed è corredato da una licenza libera, ovvero una licenza che ne permette il riutilizzo e la modifica per qualsiasi fine da parte di qualunque soggetto (pubblico o privato), richiedendo al più condizioni minimali (come la citazione degli autori originali).

In Italia, questa definizione è presente nella normativa grazie al concetto di «licenza aperta» introdotta nell'articolo 69 del Codice dell'Amministrazione Digitale. Per maggiori informazioni si può fare riferimento alle "Linee guida su acquisizione e riuso di software per le pubbliche amministrazioni", Allegato C: "Guida alle licenze Open Source[4]".

Il software libero e la Pubblica Amministrazione

Lo sviluppo aperto aiuta a realizzare software innovativi e sicuri, ragione per la quale soluzioni di software libero sono diventate punti di riferimento per risolvere problemi in quasi tutti i rami dell'informatica. Il software a licenza libera entra nella Pubblica Amministrazione già nel 2005, anno in cui il legislatore, con la prima versione del Codice dell'Amministrazione Digitale (o CAD), istituisce il "catalogo del riuso" delle soluzioni software e impone protocolli di messa a riuso del software tra le varie amministrazioni.

Questo primo modello, seppur ambizioso, limitava il riuso alla condivisione tra le sole Pubbliche Amministrazioni, relegando la partecipazione delle aziende software a un modello di sviluppo tradizionale. Dalla revisione del 2017, e in particolar modo dalla pubblicazione delle Linee guida su acquisizione e riuso di software per le pubbliche amministrazioni[5] , viene invece assegnato al *software* libero un'importanza strategica per lo sviluppo sostenibile dei servizi pubblici e in generale per l'industria digitale del nostro paese, sia in fase di acquisizione (creando una corsia preferenziale per approvvigionarsi di soluzioni basate su questo tipo di software) che di manutenzione, obbligando le Pubbliche Amministrazioni a rendere disponibile al pubblico qualunque miglioria venga sviluppata per loro conto.

Il nuovo modello di riuso, introdotto dalla riforma del CAD appena citata, semplifica ulteriormente i processi, consentendo di eliminare eventuali accordi bilaterali e utilizzando in via esclusiva le cosiddette «licenze aperte», **rendendo di fatto il software pubblico un bene comune di tutto il Paese**. Grazie a questo modello, una pubblica amministrazione (ma anche un cittadino o un ente del terzo settore) può beneficiare delle soluzioni software già sviluppate da altre amministrazioni senza affrontare nuovamente l'investimento da zero.

È ora compito delle realtà industriali supportare la Pubblica Amministrazione nella transizione digitale, ed è nostro compito aiutare le realtà industriali nella creazione di ecosistemi aperti che le facciano diventare gli attori principali di questa trasformazione.

Questo nuovo modello "a sviluppo aperto" è stato creato dopo l'analisi delle migliori pratiche industriali e con la consapevolezza dell'importanza che ricopre la creazione di un ecosistema di know-how che sappia operare in un contesto di *software* libero e che coinvolga tutti i soggetti interessati, al fine di effettuare una transizione digitale sostenibile e che generi benefici a tutto il Paese.

Il modello non è appannaggio esclusivo del nostro Paese - con questi presupposti anche molti altri (ad esempio qui le iniziative di Francia[6], Regno Unito[7], USA[8]) hanno avviato politiche nazionali per favorire l'uso del *software* libero all'interno della pubblica amministrazione.

Questa è una guida pratica, per cui non ci dilungheremo ulteriormente in spiegazioni sui benefici strutturali del *software* libero (che possono essere reperiti tra i documenti della Commissione Europea[9] e del MITD[10])

[4] https://docs.italia.it/italia/developers-italia/lg-acquisizione-e-riuso-software-per-pa-docs/it/bozza/attachments/allegato-d-guida-alle-licenze-open-source.html
[5] https://docs.italia.it/italia/developers-italia/lg-acquisizione-e-riuso-software-per-pa-docs/it/stabile/index.html
[6] https://www.etalab.gouv.fr/
[7] https://gds.blog.gov.uk/
[8] http://code.gov/
[9] https://digital-strategy.ec.europa.eu/en/library/study-about-impact-open-source-software-and-hardware-technological-independence-competitiveness-and
[10] https://innovazione.gov.it/notizie/articoli/il-valore-dell-open-source-per-un-europa-digitale-indipendente-e-competitiva/

A livello internazionale, l'utilizzo di *software* libero si è molto diffuso e in numerosi casi ha sostituito il ricorso a soluzioni proprietarie chiuse. I vantaggi di questo approccio sono sia di natura tecnica che di natura economica e strategica. Anche a livello europeo, l'attenzione al *software* libero è molto cresciuta e ha contribuito alla nascita di numerose iniziative. Ad esempio, la Commissione Europea ha promosso diversi programmi finalizzati a favorire il riuso di soluzioni software e la loro interoperabilità (si veda l'iniziativa ISA[2] o l'"osservatorio OSOR[11]) e incentivato l'uso e la contribuzione a progetti di *software* libero al proprio interno, come testimonia la strategia 2020-23 della CE[12]. Queste iniziative puntano a migliorare la qualità dei servizi pubblici, aiutare le Pubbliche Amministrazioni europee a ridurre i costi, sviluppare l'economia del territorio e incentivare una maggiore indipendenza geo-politica dai grandi fornitori di tecnologia extra europei.

[11] https://joinup.ec.europa.eu/collection/open-source-observatory-osor
[12] https://ec.europa.eu/info/departments/informatics/open-source-software-strategy_en

CAPITOLO 2

Per chi ha responsabilità politiche

2.1 A chi si rivolge questo capitolo

Questo capitolo si rivolge a chi si occupa di fornire indirizzi politici alla pubblica amministrazione per spiegare i vantaggi dell'utilizzo di software libero per le persone, le imprese e i propri enti.

2.2 I vantaggi del software libero

Il software oggi è al centro delle istituzioni pubbliche, modella il lavoro dei dipendenti pubblici e influenza la vita di quasi tutte le persone. Non è più solo uno strumento a supporto del lavoro degli esseri umani, ma in alcuni casi li sostituisce, sia applicando in modo automatico le previsioni normative (quando ad esempio calcola una riduzione di tariffa in base all'ISEE) che implementando algoritmi di intelligenza artificiale (che ad esempio suggeriscono quali distretti necessitino di servizi sociali o di rinforzi dei servizi di pubblica sicurezza).

Per questi motivi il codice applicativo dovrebbe essere soggetto ai principi di controllo e governance democratici allo stesso modo del codice legale. Il software libero, grazie alla disponibilità pubblica del codice, è particolarmente indicato a soddisfare queste relazioni. Per il principio della trasparenza amministrativa[1], deve essere sempre possibile verificare l'applicazione delle norme da parte di chi amministra nel pubblico, e così dev'essere per il software che sostituisce o collabora al loro lavoro.

Nel 21° secolo, il software può essere considerato un'infrastruttura pubblica strategica. Basare i propri servizi su codice con licenza libera, mantiene in capo alla pubblica amministrazione (o a un suo fornitore scelto di volta in volta) la proprietà del codice e la facoltà di effettuare correzioni tempestive qualora emergessero criticità.

Negli ultimi anni i governi europei hanno posto una particolare attenzione nel garantire la sovranità tecnologica - che consente loro di impostare e controllare il funzionamento del software pubblico proprio come sono in grado di impostare e controllare la politica, pubblica. Il software deve essere gestito in modo responsabile, trasparente e nel rispetto dei diritti delle persone e degli attori della società civile. Chi progetta il software deve pertanto tenere conto che si tratta di un'infrastruttura civica essenziale.

[1] L . 241/1990 modificata da L 15/2005

2.2.1 Gli svantaggi del software proprietario

Al contrario, per gli enti pubblici, l'uso di software proprietario **può comportare una mancanza di trasparenza,** soprattutto nei processi di erogazione dei servizi pubblici. La mancata trasparenza causa degli errori, anche nell'applicazione delle norme, che soltanto i fornitori possono correggere.

Nell'ultimo decennio, le organizzazioni pubbliche che hanno acquisito soluzioni software proprietarie sono state talvolta sorprese nello scoprire che:

- non sono libere di modificare (autonomamente, o tramite terzi) il loro software per riflettere modifiche nelle politiche e nei regolamenti o per sfruttare nuove tecnologie;
- non hanno libero accesso ai dati che amministrano poiché tali dati sono bloccati nei sistemi proprietari;
- sono tenute a pagare canoni di licenza in costante aumento.

Questo ha in alcuni casi comportato situazioni di lock-in[13] particolarmente oneroso, il cui vero costo è emerso solo in seguito.

Riteniamo che il software "pubblico" - quello che interviene nell'applicazione delle politiche pubbliche/nell'amministrazione della nostra società e nella gestione dei dati pubblici - non debba essere una scatola nera, acquisita da società esterne che tengono nascosta la logica con cui il loro software opera. Al contrario, il software pubblico dovrebbe permettere alle amministrazioni di operare sempre in modo libero, trasparente e indipendente.

2.2.2 I requisiti del software pubblico

Il software pubblico deve quindi essere trasparente; chi ha responsabilità amministrative, nonché la collettività, devono poter verificare facilmente il rispetto delle norme.

Il software deve riflettere i valori insiti nelle norme della società, ad esempio deve essere inclusivo, **non discriminatorio e rispettoso della privacy**. Per soddisfare questi requisiti, i sistemi software proprietari attualmente utilizzati da organizzazioni pubbliche dovrebbero essere **validati e certificati in modo indipendente a ogni modifica**. Un *software* libero, invece, è **trasparente per definizione** e, se usato insieme a disposizioni per la messa in produzione ugualmente aperte, darà luogo a infrastrutture pubbliche automaticamente trasparenti e verificabili.

2.2.3 I vantaggi di riutilizzare un software libero

Riutilizzare il software libero comporta diversi vantaggi. I principali sono:

- il miglioramento incrementale della qualità possibile grazie a una costante evoluzione, anche collaborativa, del software libero;
- la possibilità di basare le proprie soluzioni su altri software aperti che spesso rappresentano lo stato dell'arte nel loro ambito;
- la trasparenza, intrinseca nel software libero, di cui tutti possono vedere e leggere il codice sorgente;
- l'accountability del fornitore, il cui lavoro è pubblicamente visibile;
- la formazione e l'accesso facilitato alla conoscenza;
- la sicurezza del software, che può venire corretto nei suoi problemi da un ampio numero di contributori;
- le opportunità per PMI, software house e anche sviluppatori indipendenti, che possono far conoscere le proprie capacità e competenze, collaborando o evolvendo software aperti.

[13] https://it.wikipedia.org/wiki/Vendor_lock-in

2.2.4 Quali sono gli aspetti da conoscere della gestione del ciclo di vita di un software?

Accettare segnalazioni e contributi da terzi

Un progetto di *software* libero è per sua natura a disposizione di tutti. Il suo codice sorgente può essere liberamente consultato anche da persone e e imprese, che a loro volta possono verificarlo, inviare segnalazioni di malfunzionamenti o proporre miglioramenti anche in forma di codice pronto all'uso. Questo modello cambia il tradizionale rapporto tra amministrazione committente e fornitore perché **introduce anche la possibilità di contributi esterni** o in generale la formazione di un ecosistema di più soggetti che, collaborativamente, migliorano i servizi pubblici a beneficio di tutti.

Il codice libero per la pubblica amministrazione deve essere rilasciato secondo lo schema di riuso descritto nelle "Linee guida su acquisizione e riuso di software per le pubbliche amministrazioni[14]". Secondo tale schema, una PA titolare di un software, lo deve rilasciare come *software* libero su un repository pubblico. Dopo il rilascio è probabile, e anzi auspicabile, che altre sviluppatrici e sviluppatori contribuiscano allo sviluppo software risolvendo problemi o aggiungendo nuove funzionalità.

È bene che questo tipo di contributi, una volta esaminati e valutati da un punto di vista architetturale rispetto alla struttura della soluzione e alla loro qualità, vengano accettati e uniti al codice del progetto rilasciato come *software* libero, nonché alla versione in uso, per migliorare la qualità del proprio software e al contempo evitare la proliferazione di versioni alternative.

Questo consente la possibilità di creare partnership tra enti e istituzioni usando soltanto la forza dei termini di licenza libera e senza bisogno di accordi bilaterali o altre soluzioni ad hoc, mettendo in comune risorse e idee a tutto vantaggio delle persone.

2.2.5 Contribuire a software libero sviluppato da altri

Relazioni con i maintainer del software

Secondo il nuovo modello di riuso, le interazioni tra amministrazione cedente e amministrazione acquirente in una situazione di riutilizzo di una soluzione software sono limitate e non obbligatorie. Le uniche possibilità di utilizzo, modifica e rilascio del codice sono quelle descritte nella licenza applicata dal titolare del software al momento del suo rilascio come *software* libero.

Può essere necessario confrontarsi con il titolare quando si realizzano modifiche o sviluppi del software, nel momento del nuovo rilascio come *software* libero, per inserirsi nella roadmap di sviluppo della soluzione da lui definita e governata.

2.2.6 Cosa può fare in concreto chi ha responsabilità politiche

Per sostenere l'innovazione e incrementare l'uso di soluzioni libere, una/un responsabile politico può:

a. includere espressamente il *software* libero nella strategia tecnologica dei documenti di programmazione del suo ente. Il modello di riuso deve essere visto non solo come un adempimento legislativo, ma come il punto di partenza per generare delle opportunità di innovazione per il singolo ente e per l'intero sistema Paese;

b. adottare le norme del CAD con una delibera, secondo la quale lo sviluppo di software nel suo ente sarà incentrato sul *software* libero, al fine di ottenere i benefici descritti in questa guida;

c. incentivare i tecnici a collaborare apertamente a progetti di *software* libero di interesse per il suo ente;

d. incentivare l'acquisizione di personale tecnico con competenze legate al *software* libero;

[14] https://www.agid.gov.it/it/design-servizi/riuso-open-source/linee-guida-acquisizione-riuso-software-pa

e. incentivare i funzionari a utilizzare apertamente forum.italia.it[15] (un Forum di discussione per gli attori della trasformazione digitale del Paese) come strumento di confronto e condivisione.

[15] https://forum.italia.it/

CAPITOLO 3

Per chi amministra nel pubblico

3.1 A chi si rivolge questo capitolo?

Questo capitolo si rivolge ai referenti delle amministrazioni pubbliche che si occupano a diverso livello dei progetti software del proprio ente.

3.2 I vantaggi del software libero

Il software libero offre un modello di sviluppo alternativo e spesso migliore rispetto al software proprietario, sia per le organizzazioni pubbliche che per le società private. L'uso del *software* libero aumenta il controllo sulla propria possibilità di operare, la qualità delle soluzioni, e può essere un volano per aumentare le possibilità di business.

Progettato sin dall'inizio per essere aperto, adattabile e per garantire la portabilità dei dati, il software può essere sviluppato da personale interno all'amministrazione o da fornitori esterni. Il modello del *software* libero permette alla pubblica amministrazione di cambiare fornitore, se necessario, senza dover ricominciare il processo di sviluppo dal principio. **Aumenta quindi le opportunità di apprendimento e controllo pubblici**, consentendo la collaborazione su moduli funzionali più piccoli e, in tal modo, facilitando l'offerta pubblica per le piccole e medie imprese del territorio. Le pubbliche amministrazioni possono utilizzare i propri processi di acquisto del software per **stimolare l'innovazione e la concorrenza** nella loro economia locale.

L'adozione di *software* libero può quindi essere vista come **un investimento che porta alla crescita economica del sistema Paese**: nel tempo saranno sempre più richiesti nuovi fornitori a causa della crescente domanda di tecnologia.

Il *software* libero può essere utilizzato e sviluppato da team di sviluppo interni permanenti, appaltatori o fornitori in outsourcing. Inoltre, i fornitori delle organizzazioni pubbliche possono includere *software* libero nelle loro offerte.

Nella quasi totalità delle situazioni quando si progetta un nuovo servizio digitale si possono usare almeno in parte codice o componenti liberi già pronti, o ci si può basare su di essi in modo importante.

È dunque utile specificare in fase di definizione di progetti digitali non solo che i nuovi sviluppi dovranno essere rilasciati con licenza libera, così come richiesto dal CAD, ma anche che la soluzione dovrà basarsi unicamente o quanto più possibile su *software* libero.

3.3 Norme per l'acquisizione di software per la Pubblica Amministrazione

L'acquisizione di soluzioni software per la pubblica amministrazione segue il processo descritto nell'articolo 68[16] del Codice dell'Amministrazione Digitale.

In sintesi, una PA per acquisire software deve sempre effettuare una valutazione comparativa tra le soluzioni disponibili. Le alternative possono essere:

- riuso di una soluzione già disponibile, sviluppata e messa a disposizione da un'altra PA;
- riuso di una soluzione di *software* libero sviluppata da terzi;
- sviluppo di una nuova soluzione utilizzando sviluppatori interni alla PA;
- sviluppo di una nuova soluzione su commissione a un fornitore esterno.
- sviluppo ex-novo di una soluzione.

Nel caso in cui la PA decida di acquisire software proprietario o di svilupparne uno nuovo (vale a dire nel caso in cui non sfrutti il riuso di una soluzione già disponibile o di un *software* libero di terze parti) deve motivare tale scelta attraverso la redazione di una **valutazione comparativa scritta**. Il software commissionato o sviluppato deve quindi obbligatoriamente essere rilasciato come *software* libero per favorire il suo riuso da parte di altre amministrazioni.

Il catalogo di Developers Italia[17] è centrale in questo processo di pubblicazione e riuso di software: facilita le operazioni di inserimento di nuove soluzioni da parte delle pubbliche amministrazioni e la consultazione di soluzioni già messe a disposizione.

3.3.1 Le fasi di sviluppo e riuso di software

Nel seguito sono descritte nel dettaglio la **fase di sviluppo** e la **fase di riuso** di software evidenziando in particolare le interazioni che avvengono con il catalogo.

Diagramma esemplificativo della **fase di sviluppo**:

1. La PA "A" decide di sviluppare un ipotetico software "Imago" da zero e lo commissiona a uno sviluppatore (interno o tramite appalto).
2. La PA "A" acquisisce la completa titolarità di quanto sviluppato.
3. La PA "A" incarica lo sviluppatore di pubblicare il codice sorgente completo della soluzione con licenza libera, durante o al termine della lavorazione, su uno strumento di code hosting.
4. Il software viene "registrato" su Developers Italia[18], per acquisire visibilità nazionale e internazionale.

[16] https://docs.italia.it/italia/piano-triennale-ict/codice-amministrazione-digitale-docs/it/v2017-12-13/_rst/capo6_art68.html
[17] https://developers.italia.it/it/software
[18] https://developers.italia.it/it/riuso/pubblicazione

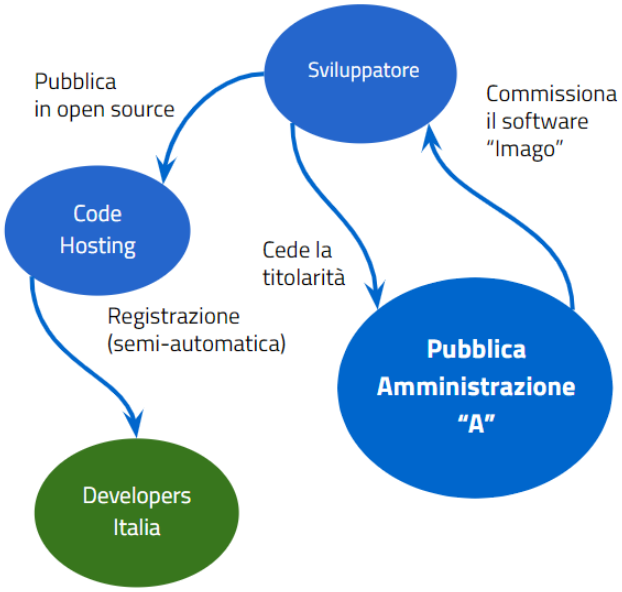
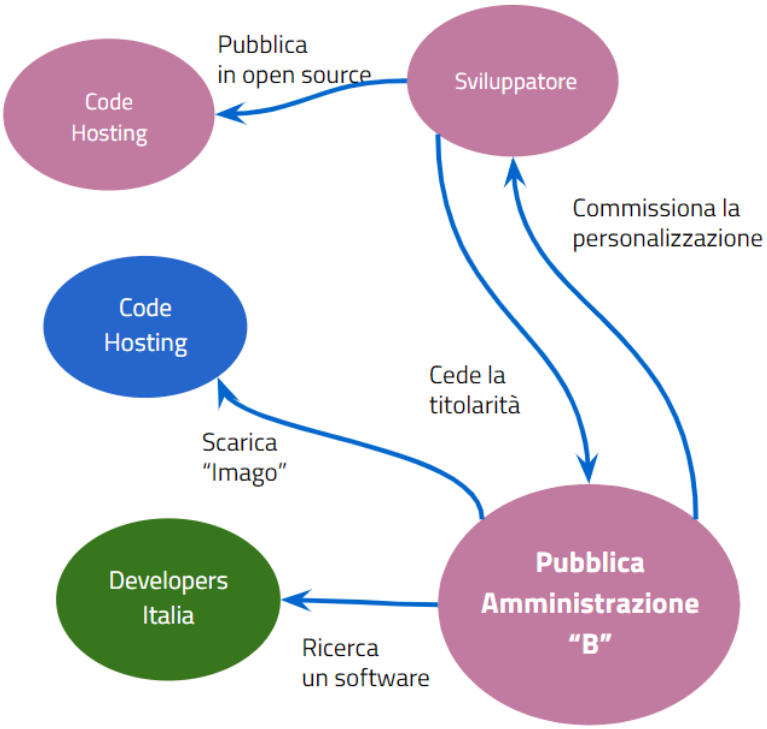


Diagramma esplicativo della **fase di riuso**:

1. La PA "B", durante la valutazione comparativa, cerca[19] su Developers Italia un software a riuso adatto alle proprie esigenze e trova "Imago".
2. La PA "B" scarica liberamente il software dallo strumento di code hosting della PA "A" e lo valuta in autonomia.
3. La PA "B" incarica uno sviluppatore di personalizzare il software, installarlo, mantenerlo, e formare il proprio personale. Questo sviluppatore può essere lo stesso o un altro, ma deve proporre ogni miglioria per integrazione agli autori originali. Il software non deve essere necessariamente ripubblicato a patto che tutte le personalizzazioni siano integrate dal team di sviluppo originale.
4. Il software, se personalizzato, dovrà essere pubblicato sulla propria piattaforma di code hosting e nuovamente registrato all'interno di Developers Italia[20].

[19] https://developers.italia.it/it/search?type=software_reuse&sort_by=relevance&page=0
[20] https://onboarding.developers.italia.it/



In generale, rispetto a questo modello di riuso possiamo sottolineare che le Pubbliche Amministrazioni "A" e "B" **non devono neppure contattarsi.** La licenza libera regola le condizioni di utilizzo del software. Il software può essere scritto da uno sviluppatore e personalizzato da un altro, questo elimina i vincoli e i "lock-in". La parte tecnica di pubblicazione è sempre demandata agli sviluppatori poiché è parte integrante del processo di sviluppo. Questo, quindi, è un sistema efficiente e a basso costo.

3.4 Rilasciare nuovo software libero

Dopo aver presentato i benefici del *software* libero, in questo capitolo viene descritta una possibile modalità di pubblicazione del proprio software come *software* libero.

3.4.1 Scegliere una piattaforma di code hosting

Le pubbliche amministrazioni dovrebbero scegliere un sistema di controllo delle versioni basato sul sistema git, prendendo in considerazione l'utilizzo di una delle seguenti piattaforme di code hosting, in base alle loro funzionalità e all'adozione significativa da parte della comunità di *software* libero di riferimento.

Esistono piattaforme di code hosting sia on-premises che SaaS (anche a utilizzo gratuito). Tra queste ultime citiamo le più scelte tra i progetti liberi moderni:

- GitLab - https://gitlab.com[21]
- GitHub - https://github.com[22]
- Bitbucket - https://bitbucket.org[23]

[21] https://gitlab.com/
[22] https://github.com/
[23] https://bitbucket.org/

- Codeberg - https://codeberg.org/

All'interno di uno strumento di code hosting è possibile organizzare i propri progetti in modo gerarchico, creando un'**organizzazione** che conterrà all'interno tutti i singoli **repository** di software. Tale organizzazione può assumere un nome diverso in base alla piattaforma utilizzata: "organizzazione" nel caso di GitHub, "Group" per Gitlab e "Team" per Bitbucket.

Si può immaginare l'organizzazione come un raccoglitore dei singoli repository che a loro volta conterranno il codice sorgente.

È buona norma definire il nome dell'organizzazione in modo coerente rispetto all'ente che la sta creando. Ad esempio, prendendo l'ente fittizio creato come esempio chiamato "Comune di Réuso", l'organizzazione potrebbe chiamarsi "comune-reuso" e contenere al suo interno tutti i repository con il software di titolarità del comune. È infatti ragionevole pensare che, ad esempio, il software per la gestione del protocollo informatico abbia uno spazio a sé stante diverso da quello del software usato per gestire la lavorazione delle pratiche ricevute via PEC.

Il seguente schema ad albero rappresenta una possibile strutturazione dei repository all'interno di una organizzazione.

```
comune-reuso
|--  software-protocollo
|      |-- LICENSE
|      |-- README
|--  software-pec
|      |-- LICENSE
|      |-- README
```

La modalità di creazione di tali organizzazioni differisce a seconda della piattaforma adottata.

Qui di seguito saranno elencate le modalità di creazione di una organizzazione all'interno di alcune tra le piattaforme elencate nelle linee guida[24].

GitLab

Un'organizzazione (in questo caso "Gruppo") in GitLab può essere creata in modo molto semplice direttamente dall'interfaccia online del servizio.

Per creare un'organizzazione, è possibile seguire i seguenti passi:

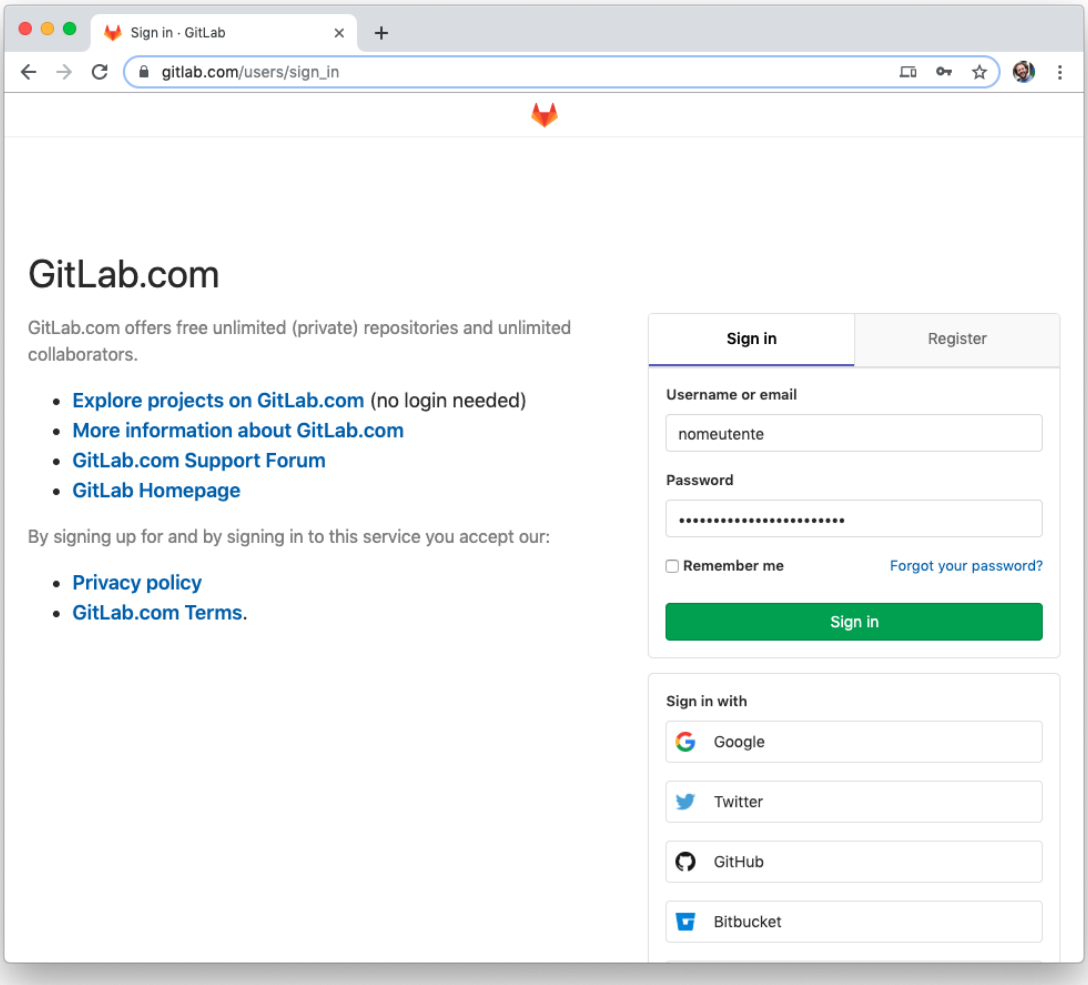
1. **Fare login su Gitlab con le proprie credenziali**

Aperto il sito https://www.gitlab.com/ è possibile fare accesso selezionando sulla barra in alto, a destra, il bottone "Sign In".

Nella schermata che compare è possibile specificare il proprio nome utente e password o scegliere una modalità di accesso differente.

Se non si dispone di un utente è possibile selezionare "Register" per creare un nuovo utente sulla piattaforma.

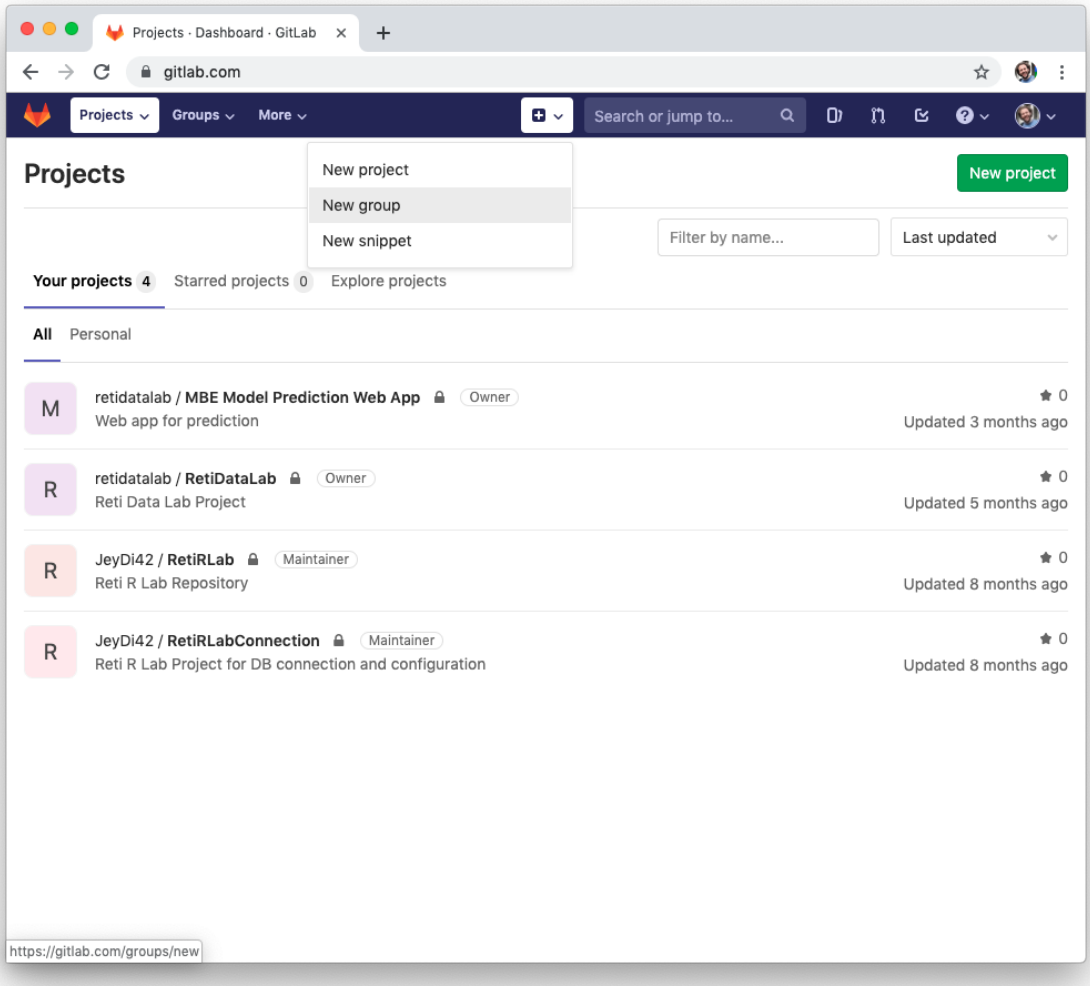
[24] https://docs.italia.it/italia/developers-italia/lg-acquisizione-e-riuso-software-per-pa-docs/it/stabile/attachments/allegato-b-guida-alla-pubblicazione-open-source-di-software-realizzato-per-la-pa.html?highlight=repository



2. **Creare un nuovo "Gruppo"**

Un gruppo può essere utilizzato anche per identificare l'organizzazione della nostra pubblica amministrazione.

Per creare un nuovo Gruppo dalla barra in alto basta portare il mouse sopra il bottone che presenta un'immagine con un + e selezionare dal menù a tendina "New group".

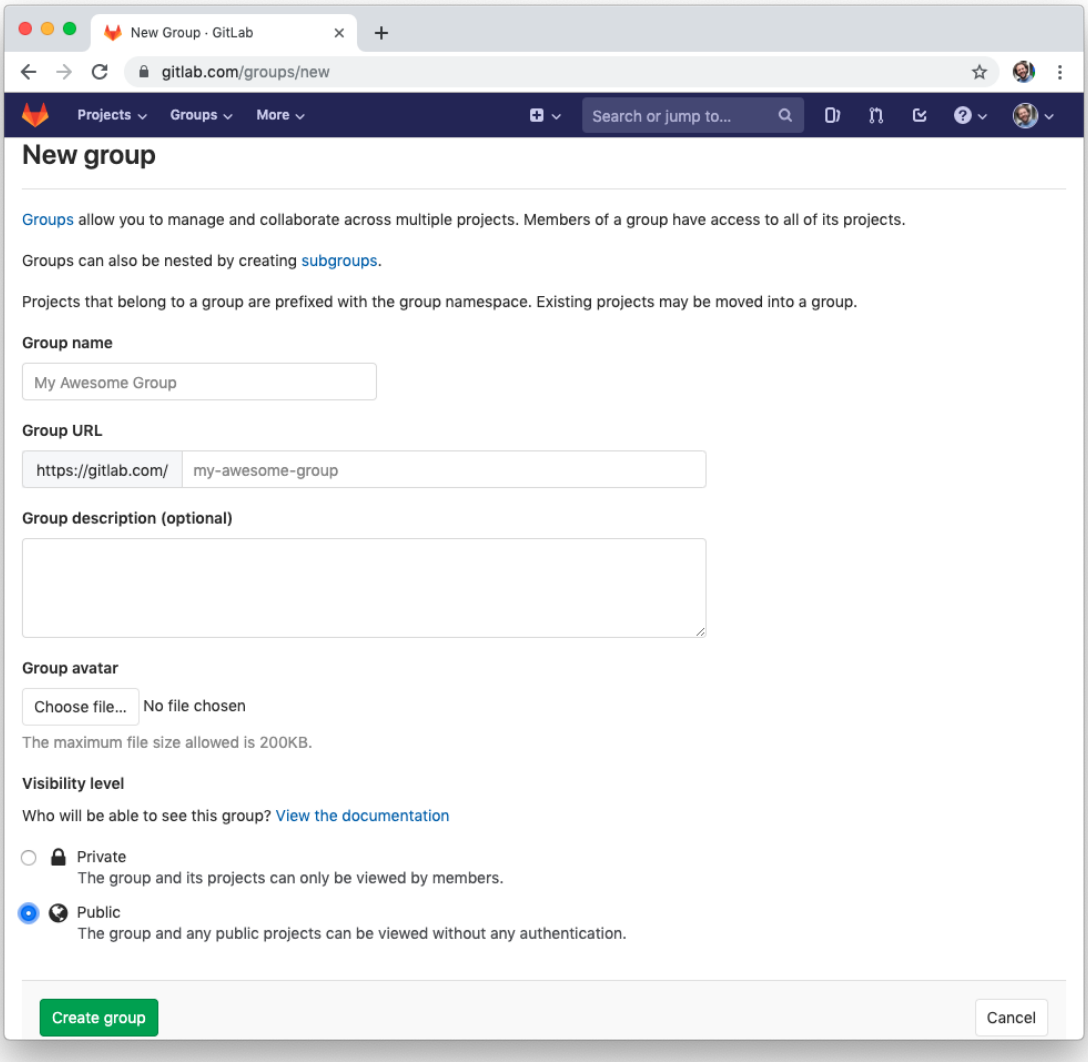


3. **Specificare le informazioni per il Gruppo**

A questo punto sarà possibile specificare le informazioni rilevanti per la creazione del gruppo. È necessario specificare:

- Group name: indicare il nome della PA (ad esempio Comune di Reuso).
- Group URL: specificare il nome breve della PA che sarà usato come parte finale della URL dell'organizzazion. Questo nome non può avere spazi o caratteri speciali, come caratteri accentati (ad esempio comune-reuso)
- Group description: opzionalmente si può specificare una descrizione della PA.
- Visibility level: attenzione! è importante specificare il livello di visibilità del gruppo **Public**.

Al termine, cliccando sul bottone "Create group" il gruppo sarà creato e disponibile per contenere i repository della PA.



L'URL del gruppo così creato (nell'esempio fatto https://gitlab.com/comune-reuso) sarà il valore da specificare nel campo "URL dell'account nello strumento di code hosting" durante la procedura di aggiunta al Catalogo di Developers Italia[25].

GitHub

Un'organizzazione su GitHub può essere creata in modo molto semplice e gratuito direttamente tramite l'interfaccia online del servizio. Per creare un'organizzazione, è possibile seguire i seguenti passi:

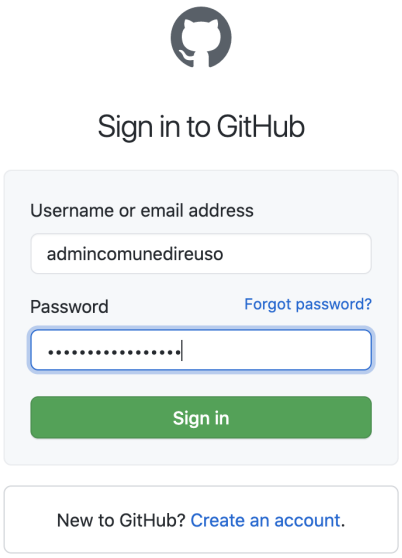
1. **Fare login su GitHub con le proprie credenziali**

Aperto il sito https://www.github.com/ è possibile effettuare l'accesso selezionando sulla barra in alto a destra, il bottone "Sign In".

Nella schermata che compare è possibile specificare il proprio nome utente e password o scegliere una modalità di accesso differente.

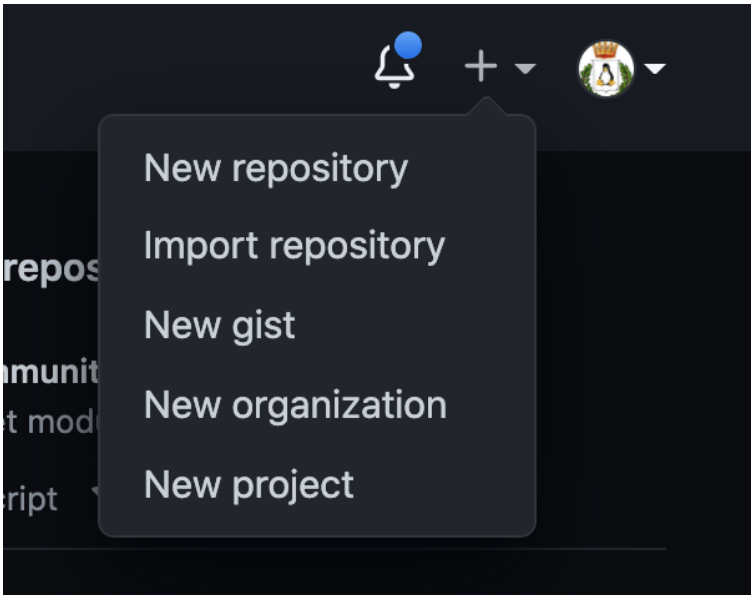
[25] http://onboarding.developers.italia.it

Se non si dispone di un utente è possibile selezionare “Create an account” per creare un nuovo utente sulla piattaforma.



2. **Creare una nuova organizzazione**

Per creare una nuova organizzazione è necessario selezionare il bottone “+“ a fianco dell’avatar del vostro utente e selezionare dal menù a tendina “New organization”.

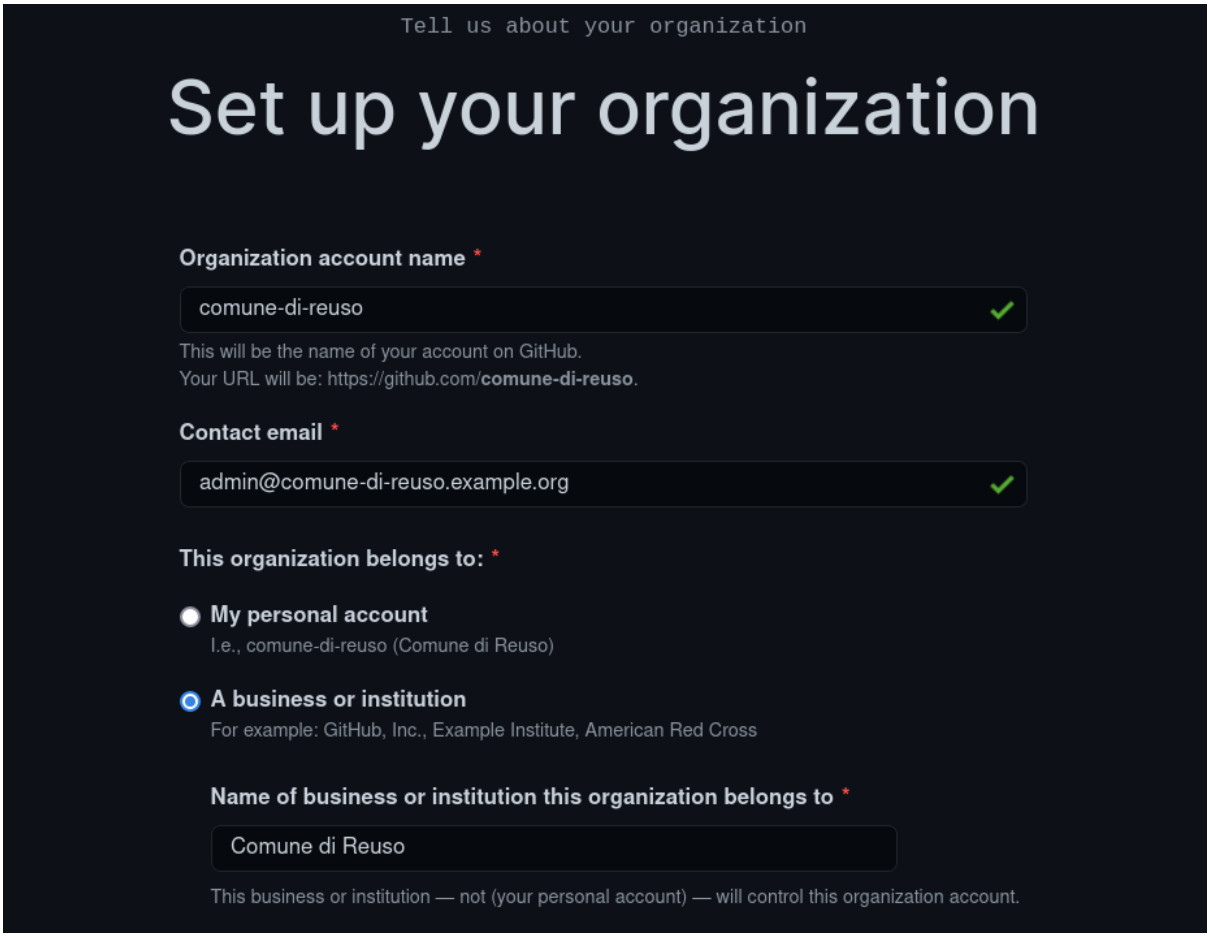


3. **Specificare le informazioni per l’organizzazione**

Dopo aver specificato di voler creare una organizzazione, sarà possibile aggiungere le informazioni rilevanti. È necessario specificare:

- Organization account name: indicare il nome dell’organizzazione della PA, ad esempio comune-di-reuso, in modo che sia raggiungibile su https://github.com/comune-di-reuso.
- Contact email: specificare la mail di contatto per qualsiasi informazione riguardante l’organizzazione.
- Belongs to: indicare il nome di un ente al quale l’organizzazione sarà legata

Dopo una serie di passaggi di verifica, premendo sul tasto "Next" verrà richiesto quali account invitare all'interno dell'organizzazione e quale visibilità impostare (nel caso specifico suggeriamo "public").



L'URL dell'organizzazione così creato sarà https://github.com/comune-di-reuso.

Bitbucket

Un'organizzazione (in questo caso "Team") in Bitbucket può essere creata in modo molto semplice direttamente dall'interfaccia online del servizio.

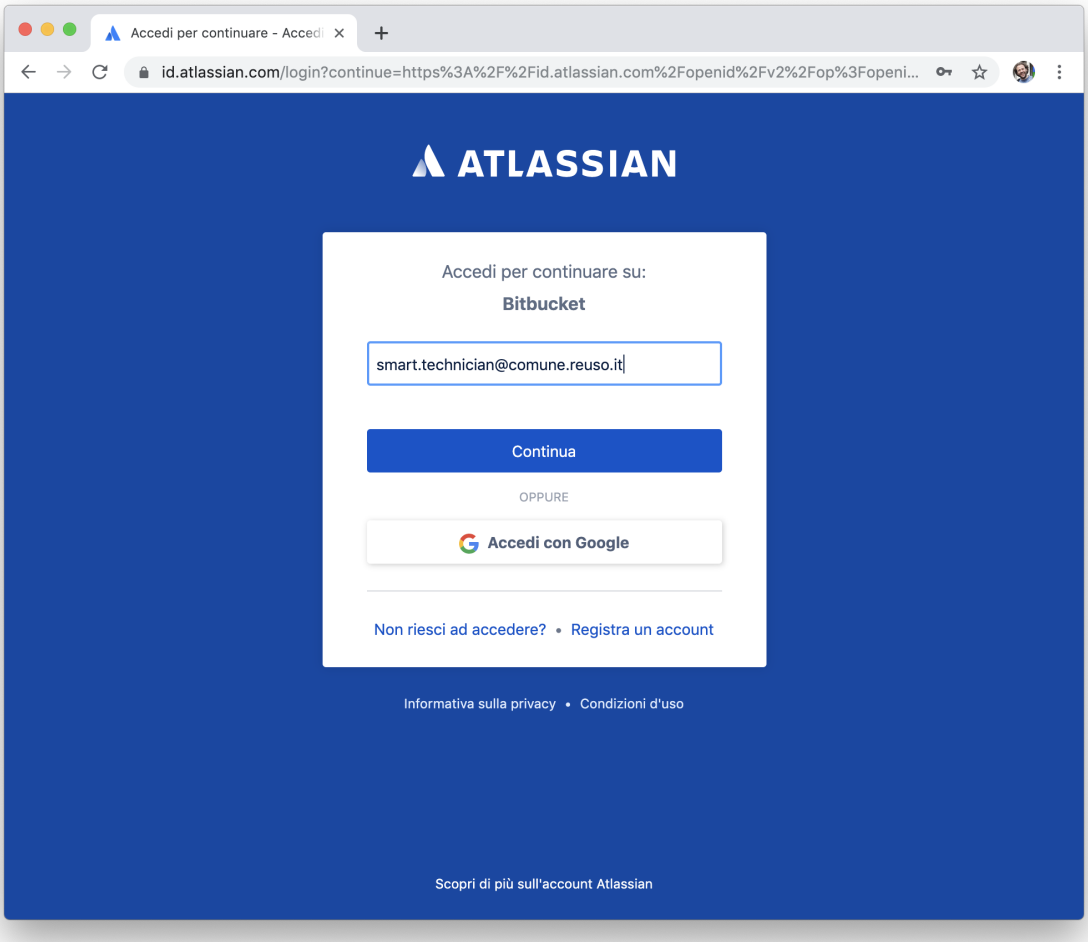
Per creare un'organizzazione, è possibile seguire i seguenti passi:

1. **Fare login su Bitbucket con le proprie credenziali**

Aperto il sito https://bitbucket.org[26] è possibile fare accesso selezionando "Log in" in alto e quindi specificando il proprio nome utente e password o scegliendo una modalità di accesso differente.

Se non si dispone di un utente è possibile selezionare "Registra un account" sotto ai bottoni di accesso per creare un nuovo utente sulla piattaforma.

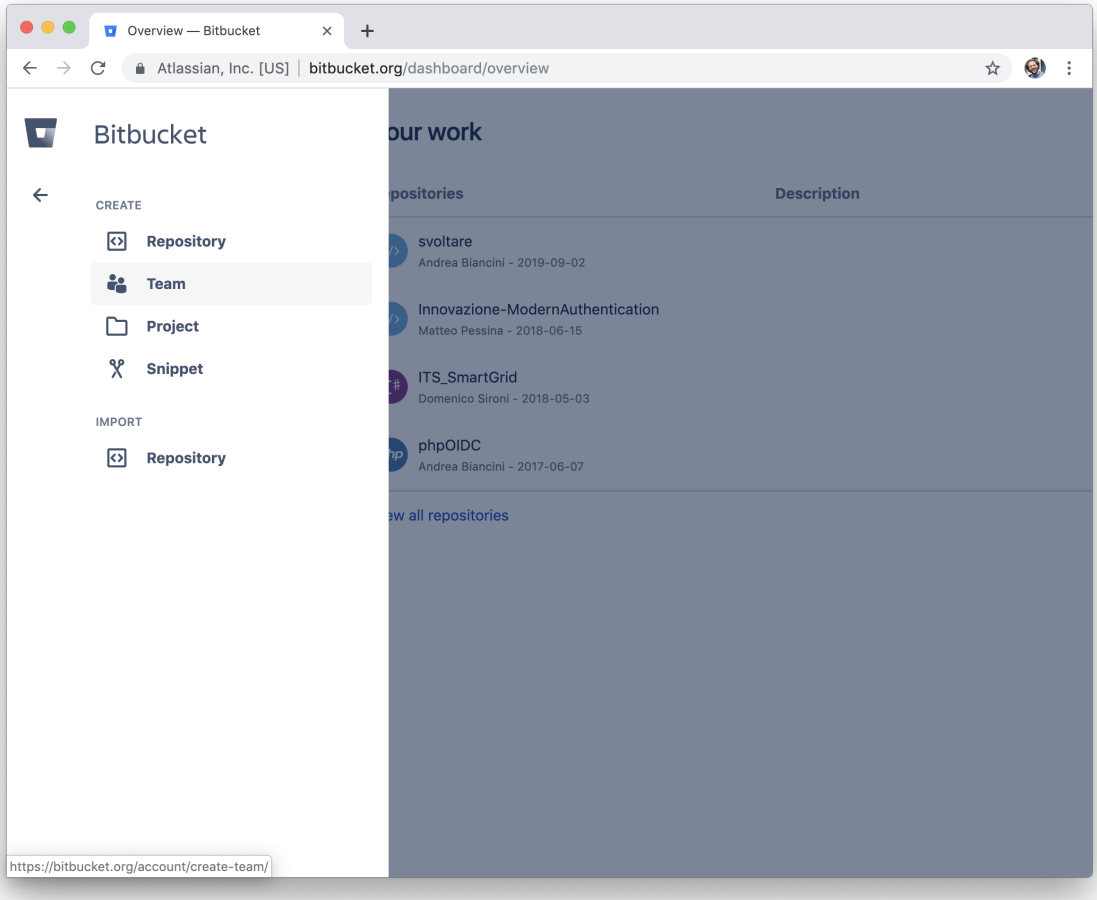
[26] https://bitbucket.org/



2. **Creare un nuovo "Team"**

Un team può essere utilizzato anche per identificare l'organizzazione della nostra pubblica amministrazione.

Per creare un nuovo Team è possibile cliccare sull'icona con il simbolo **+** nel menu a scomparsa di sinistra e quindi selezionare Team.

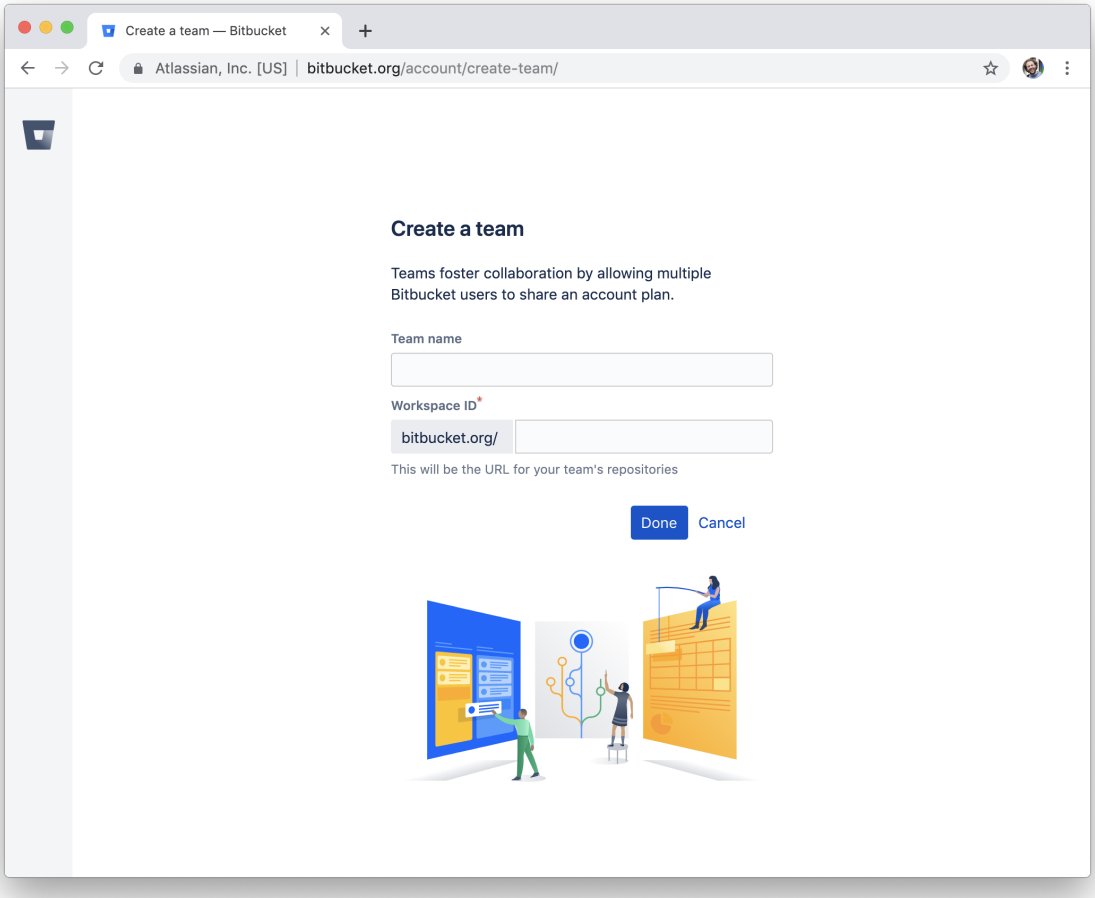


3. **Specificare le informazioni per il Team**

A questo punto sarà possibile specificare le informazioni rilevanti per la creazione del team. É necessario dunque specificare:

- Team name: indicare il nome della PA (ad esempio Comune di Reuso)
- Workspace ID: specificare il nome breve della PA che sarà usato come parte dell'URL dell'organizzazione. Questo nome non può avere spazi o caratteri speciali, come caratteri accentati (ad esempio "comune-reuso")

Al termine, premendo sul bottone "done" il team sarà creato e disponibile per contenere i repository della PA.



L'URL del team così creato sarà https://bitbucket.org/comune-reuso/.

Convertire un utente in organizzazione

Nel caso in cui si fosse inavvertitamente effettuato il processo di onboarding inserendo l'indirizzo (URL) di un utente invece che di un'organizzazione è possibile apportare una correzione senza dover rifare l'operazione di onboarding.

GitHub

La piattaforma GitHub permette di convertire un account utente in un'organizzazione ma, nel farlo, si perderà l'accesso al primo.

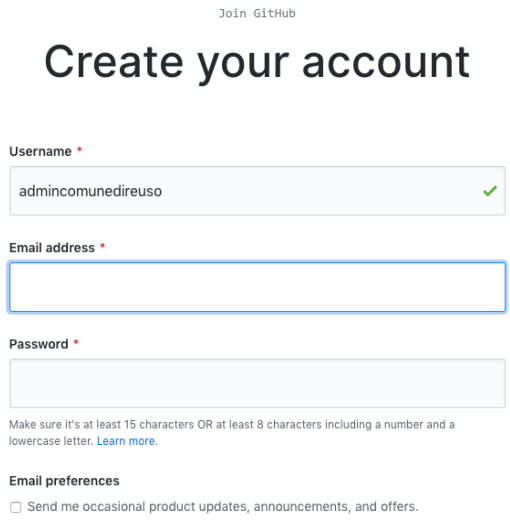
Ipotizziamo di avere un utente denominato "comune-reuso" da convertire in un'organizzazione. Per farlo si possono seguire i seguenti passi:

1. creare un nuovo utente che coprirà il ruolo di amministratore, ad esempio "*admincomunedireuso*"
2. Effettuare il login con il vecchio utente nel nostro esempio denominato "comune-reuso".
3. Convertire l'utente denominato "*comune-reuso*" in una vera e propria organizzazione. Durante questa operazione è importante indicare il nome dell'utente che avrà la gestione dell'amministrazione, nel nostro esempio sarà "*admincomunedireuso*".

4. A questo punto avremo una URL del tipo "*github.com/comune-reuso*" che sarà un'organizzazione gestita dall'utente "*admincomunedireuso*"

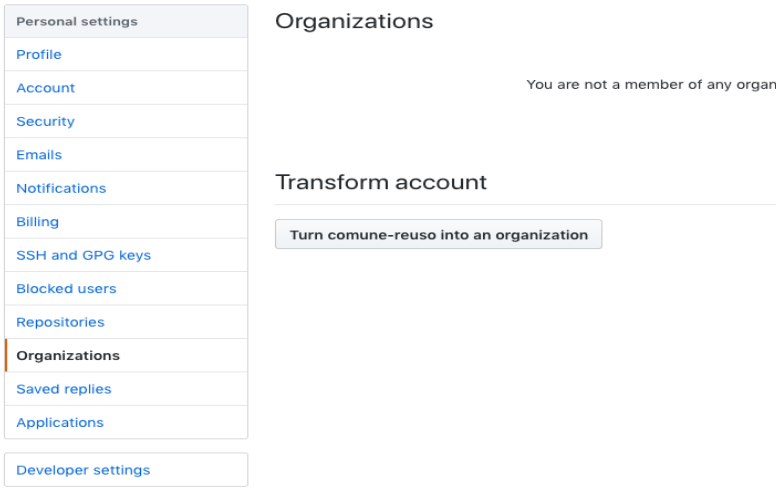
Più nel dettaglio:

1. **Creare un utente di amministrazione** (https://github.com/join)

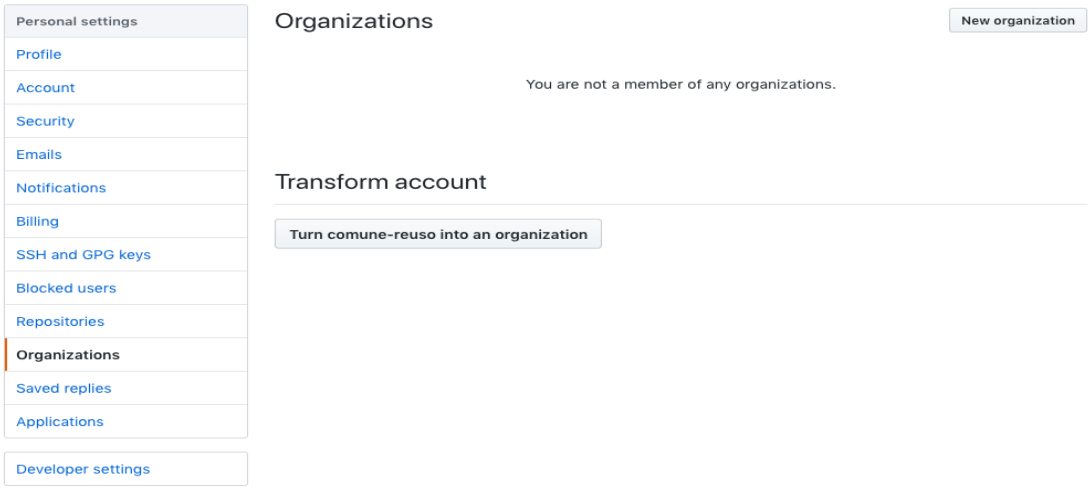


2. **Convertire l'account in un'organizzazione**.

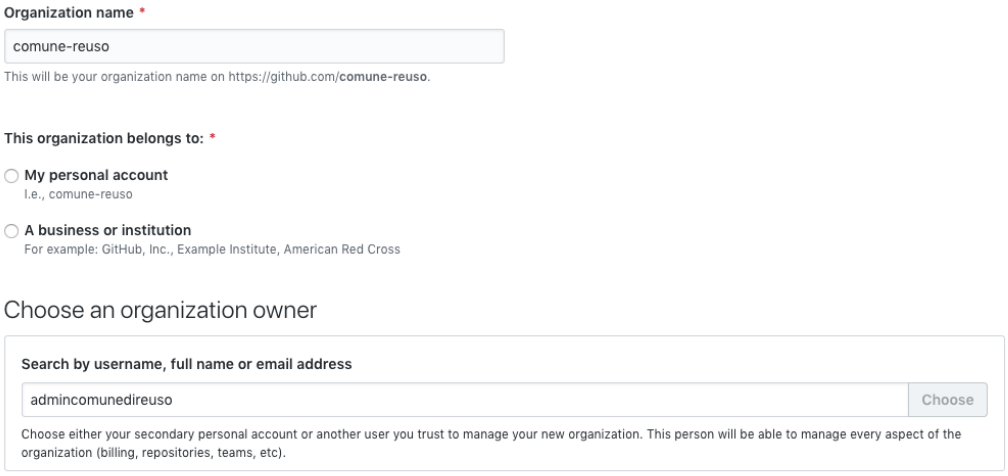
Per farlo è necessario effettuare il login come "comune-reuso", selezionare il menu **Settings** e successivamente selezionare il menu sulla sinistra chiamato **Organizations.**



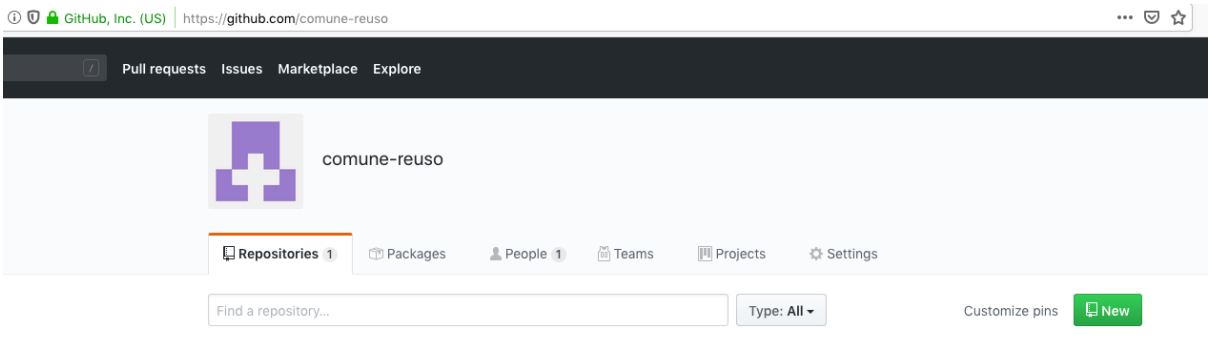
3. Selezionare il bottone **Turn comune-reuso into an organization.**



Proseguire con la procedura, **facendo molta attenzione ad assegnare l'utente creato al punto 1, ovvero "admincomunedireuso", come titolare della nuova organizzazione .** Se questo non dovesse essere effettuato correttamente **si perderebbe l'accesso a tale organizzazione.**



A questo punto la procedura è terminata, sarà dunque possibile visualizzare la propria organizzazione su https://github.com/comune-reuso



3.4.2 Aggiungere l'organizzazione a Developers Italia

Una volta ottenuta la propria organizzazione dallo strumento di code hosting la si può registrare nel Catalogo di software pubblico[27] di Developers Italia, attraverso il sito https://onboarding.developers.italia.it. All'interno di questo portale si dovranno inserire le informazioni relative al referente e la URL dell'organizzazione dell'ente.

Questa operazione permetterà a Developers Italia di indicizzare automaticamente tutti i software dell'organizzazione in modo da renderli facilmente trovabili all'interno del Catalogo del software a riuso.

3.4.3 Scegliere il nome del progetto

La denominazione del progetto (e del repository associato) è una parte importante del rilascio.

Si suggerisce di:

- usare un nome descrittivo che chiarisca le finalità del progetto.
- non utilizzare marchi di terze parti se non quando necessario; ad esempio possono essere utilizzati come descrittori (ad esempio «Librerie di test per Java» anziché «Librerie di test Java»).
- non scegliere come nome di progetto un marchio registrato di proprietà altrui.
- per i nomi dei repository, separare le parole con trattini invece di concatenarle (ad esempio invece di "*succes-sortoserverless"* utilizzare "*successor-to-serverless"*). Questo aumenta la leggibilità da parte di chi dovrà usare il software.

3.4.4 Scegliere e dichiarare la licenza

È fondamentale operare la scelta della licenza nel momento della nascita del progetto. Oltre al fatto che un progetto senza licenza non può essere considerato *software* libero (a prescindere dalla leggibilità del suo codice sorgente) possono emergere problemi nel caso dovessero sopravvenire modifiche o suggerimenti di miglioramento. In questo caso la licenza dei contributi non sarebbe chiara e questo potrebbe comportare controversie legali.

Inoltre, è importante evitare di usare la dicitura «Tutti i diritti riservati» o «All rights reserved», in quanto in contraddizione con l'apposizione di una licenza libera.

Per questo motivo ogni repository deve obbligatoriamente avere una licenza riportata nel file dedicato (chiamato normalmente LICENSE o LICENSE.md). In caso di conferimento iniziale, il file LICENSE può essere incluso direttamente nella prima *pull request* (come viene chiamato su molte piattaforme il meccanismo di proposta di modifiche) purché il commit sia effettuato dal soggetto titolare del codice.

Per indicazioni circa le licenze, si può fare riferimento alle "Linee guida su acquisizione e riuso di software per le pubbliche amministrazioni", Allegato C: Guida alle licenze Open Source[28].

3.4.5 Accettare dei contributi dopo il rilascio

Dopo aver rilasciato un *software* libero per il riuso, è molto probabile che qualche altro soggetto o amministrazione la voglia utilizzare per i propri scopi. In questo riutilizzo, il codice potrebbe ricevere contributi di miglioramento, correzioni di errori o sviluppo di nuove funzionalità.

È bene che questi contributi siano accettati e integrati nel codice sorgente del progetto in modo da rappresentare un miglioramento per tutti coloro che sono interessati al suo riuso. Per accettare i contributi, tuttavia, occorre verificare alcuni aspetti:

[27] https://developers.italia.it/it/search

[28] https://docs.italia.it/italia/developers-italia/lg-acquisizione-e-riuso-software-per-pa-docs/it/bozza/attachments/allegato-d-guida-alle-licenze-open-source.html

- i contributi devono essere revisionati in termini di potenziali rischi per la sicurezza della soluzione;
- i contributi non devono riguardare personalizzazioni del software in questione non compatibili con un utilizzo generico da parte di terzi;
- è consigliabile che il titolare e mantenga il controllo dell'architettura e della qualità del software da lui prodotto e verifichi quindi che i contributi non violino regole di struttura o di organizzazione del progetto.

3.5 Gestire un software nel suo ciclo di vita

3.5.1 Gestione delle segnalazioni

Quando un utilizzatore trova una funzionalità mancante o un problema sul codice, può aprire una issue (segnalazione) per notificare un problema in modo che esso venga preso in considerazione e corretto.

L'Allegato B[29] alle Linee guida su acquisizione e riuso di software per le pubbliche amministrazioni descrive proprio questi aspetti legati alla manutenzione di *software* libero.

I sistemi di versionamento del codice, solitamente, dispongono di meccanismi per registrare issue e per seguire la loro risoluzione. Quando si usano tali sistemi, è buona pratica che chi gestisce le issue **risponda abbastanza tempestivamente alle segnalazioni indicando** la presa in carico e specificando le modalità e le tempistiche con cui saranno risolte.

L'allegato alla linee guida descrive alcune interazioni tipiche che possono verificarsi sul sistema di code hosting di *software* libero:

- risoluzione di bug;
- richieste di nuove funzionalità;
- richieste di informazioni o supporto;
- contributi di codice.

L'Ente che ha rilasciato il software deve tendenzialmente occuparsi di garantirne il mantenimento, impostando il flusso di gestione e intervento sulle segnalazioni che provengono dall'esterno. Le segnalazioni, quindi, devono essere monitorate e prioritizzate. Si interverrà poi su alcune di esse, per risolverle e chiuderle.

Diverse issue possono essere prese in carico in un unico periodo e portare al rilascio di una nuova versione del software che le risolve tutte contemporaneamente. In questo caso il progetto lavora per *milestone*, e ogni milestone risolve un numero variabile di *issue* aperte nel periodo precedente.

3.5.2 Gestione delle richieste di modifica

Quando un contributore invia una richiesta di modifica al codice tramite pull/merge request il maintainer è tenuto a dare un riscontro tempestivo, anche solo per ringraziare il contributore in attesa della revisione del codice. Fornire un feedback immediato ai contributori aiuta a comunicare che il progetto è vivo e mantenuto e incoraggia altri contributi esterni. In questo modo, inoltre, si evita che, non vedendo riscontri, gli altri sviluppatori proseguano lo sviluppo in un proprio fork.

[29] https://docs.italia.it/italia/developers-italia/lg-acquisizione-e-riuso-software-per-pa-docs/it/stabile/attachments/allegato-b-guida-alla-manutenzione-di-software-open-source.html

3.5.3 Dichiarazione di obsolescenza di un progetto

Può capitare che un repository non venga più mantenuto o esaminato e quindi non sia più di interesse, è tuttavia utile conservarlo per scopi di archiviazione e visualizzazione pubblica. In questo caso è importante denominare il repository come *deprecated* (obsoleto) in modo da segnalare che il progetto non è più attivo e mantenuto ma è conservato per scopi di archiviazione.

Rendere obsoleto ("deprecate") un progetto che nessuno usa

Di seguito i passaggi da seguire per deprecare un repository inattivo:

- Cambiare la descrizione sul repository e all'interno del file *README.md*. La descrizione è la prima cosa che viene letta da un utente che accede al repository e si trova all'inizio del file README. Risulta quindi importante aggiungere una frase del tipo:

  ```
  Attenzione! Questo progetto non è più manutenuto dai suoi autori.
  ```

  Si suggerisce anche di aggiungere nel README.md anche i riferimenti di contatto se qualche utente volesse prendersi carico del progetto come maintainer. Se c'è un motivo particolare per cui il software è stato deprecato è bene specificarlo in questa sede.

- Aggiungere un topic al repo: per esempio **deprecated**, **obsolete**, e **archived**.
- Aggiungere il badge "no-maintenance-intended" (il codice si trova su https://unmaintained.tech/). Questo è un altro modo per segnalare in maniera molto visibile che il progetto non sarà più mantenuto.

CAPITOLO 4

Per chi sviluppa il software

4.1 A chi si rivolge questo capitolo?

Questo capitolo è rivolto alle sviluppatrici e agli sviluppatori che lavorano con il software pubblico. Si presuppone pertanto una conoscenza di base degli strumenti di lavoro come git, linguaggi di programmazione e strumenti di integrazione, ma non di metodologie avanzate o collaborative, come il cosiddetto "GitFlow" o come saper gestire progetti software pubblici.

In questa sezione, particolare attenzione sarà dedicata a come scrivere e gestire codice in maniera sicura, considerando gli accorgimenti necessari a separare lo strato di autenticazione ai servizi (ad es. le credenziali di un database) dalla logica applicativa, nonché tutte le buone pratiche. A differenza di quanto si possa credere, infatti, la pubblicazione del codice sorgente, se eseguita in maniera corretta, aiuta a ridurre la superficie di attacco di un possibile malintenzionato.

4.2 Quando devo rilasciare il mio software?

Sviluppare in modo aperto, o utilizzando i paradigmi del *software* libero non significa solo condividere il codice ma anche potenzialmente aprire ad un pubblico più ampio l'intero processo di sviluppo. Aprendo il processo si consente a soggetti terzi, anche della stessa organizzazione, ma esterni alla squadra originale di sviluppo, di partecipare e comprendere l'evoluzione tecnica del prodotto. Anche le prime bozze di codice sono spesso utili a capire le scelte di design effettuate, e poter ricostruire la storia di ogni singola riga aiuta a trovare e correggere più facilmente bug. A questo scopo occorre utilizzare un sistema di controllo di versione che consenta di registrare separatamente i contributi dei diversi sviluppatori che giornalmente collaborano al progetto. Il sistema di controllo di versione distribuito git[30], sviluppato dalla comunità di sviluppatori del kernel Linux, rappresenta la soluzione maggiormente utilizzata nell'ambito dello sviluppo di software libero.

Ai fini di ricostruire correttamente la storia del codice, la pubblicazione di un nuovo progetto non deve pertanto essere rimandata al momento del rilascio della prima versione completa del software, ma dovrebbe essere effettuata già dalla prima settimana di sviluppo, specificando ovviamente nel README e nelle altre documentazioni utente che si tratta di un lavoro in corso. Questa pubblicazione, nel caso di software pubblico deve essere fatta su di una piattaforma

[30] https://git-scm.com/book/it/v2/Per-Iniziare-Una-Breve-Storia-di-Git

pubblica di code hosting che consenta la proposta di contributi e modifiche da parte di altri sviluppatori, come ad esempio GitHub o GitLab - servizi web costruiti intorno al sistema git.

Lo sviluppo deve essere fatto per piccoli incrementi, da dividere in commit separati, ed è buona prassi adottare un workflow esplicio per lo sviluppo sulla piattaforma di code hosting, come ad esempio il GitHub Flow[31], ovvero una metodologia che consiste nel separare il lavoro su rami di sviluppo separati e proporre le modifiche verso il ramo principale (solitamente chiamato *master o main*), in modo da poter chiedere la revisione agli altri maintainer o ai Project Leader. Questo processo aiuta a scrivere codice più robusto, a documentarlo in modo più trasparente e a controllare che modifiche frettolose non ne modifichino il funzionamento.

In ultimo, per consentire ai potenziali contributori o agli interessati di capire la direzione del progetto è importante condividere non solo il software, ma anche le decisioni, le priorità e le roadmap.

4.3 Rilasciare software libero

Nel seguito sono descritte delle buone pratiche per rilasciare software in modo efficace seguendo il paradigma del *software* libero.

4.3.1 Preparare il progetto

Prima di rilasciare del codice come *software* libero è fondamentale prestare molta attenzione a ogni singolo contenuto che dovrà essere caricato sul repository e che di conseguenza diventerà pubblicamente fruibile. È quindi importante effettuare dei controlli preventivi a tutti i file (anche a quelli nascosti che a volte potrebbero sfuggire ad un'analisi superficiale) per assicurarsi che siano adatti alla visualizzazione pubblica. In tal senso, è importante porre attenzione anche ai commenti del codice: spesso, infatti, accade che nei commenti gli sviluppatori utilizzino un linguaggio inopportuno alla pubblicazione.

Inoltre, è bene accertarsi di alcuni principi elementari di sicurezza per verificare che il software sia ragionevolmente sicuro per essere rilasciato pubblicamente. Per esempio, è fondamentale porre particolare attenzione nel verificare che nel codice non siano state dimenticate delle password (nemmeno di ambienti di test o sviluppo).

Problema simile a quello delle password riguarda i token di autenticazione usati per alcuni servizi acceduti tramite rete.

Per ovviare a ciò è possibile usare dei rapidi stratagemmi. Ad esempio, per individuare nomi host, indirizzi di posta elettronica e indirizzi IP, è possibile eseguire questo comando in console:

```
egrep -r '\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,6}\b"

\|([0-9]+\.){3}[0-9]+' ${SOURCE_PATH:=.}
```

Oppure, per trovare i commenti nei sorgenti in linguaggi Java/C/C++/Go/Objective-C/Objective-C++/Swift/Kotlin è possibile usare il seguente comando:

```
find ${SOURCE_PATH:=.} -type f \| egrep

'\.(c|cc|h|cpp|go|java|kt|m|mm|swift)' \| while read f; do echo

"------------ $f ------------------"; sed -n -e '/\/\*.*\*\// {p;

b}' -e '/\/\*/,/\*\//p' -e '/\/\//p' "$f"; done
```

Infine, per vedere i commenti in Python/Bash:

[31] https://guides.github.com/introduction/flow/

```
find ${SOURCE_PATH:=.} -type f \| egrep '\.(py|sh)' \| while read f; do
echo "------------ $f ------------------"; grep -o "#.*" "$f"; done
```

Questi sono solo alcuni esempi utili per trovare rapidi riscontri ma deve essere cura del maintainer effettuare dei controlli approfonditi in questa fase preparatoria alla pubblicazione.

Se è stato utilizzato il sistema di versionamento git fin dall'inizio dello sviluppo è fondamentale prestare attenzione al fatto che tutte le modifiche apportate ai vari file siano state tracciate e quindi visibili nella *history*. In tal senso, se vengono rimosse delle informazioni in fase di preparazione è fondamentale che le stesse informazioni siano rimosse anche dalla history. Per effettuare questo tipo di operazione esistono diversi strumenti. Il documento Removing sensitive data from a repository[32] (scritto dalla piattaforma GitHub ma applicabile in generale a tutti i repository git) rappresenta una guida utile per rimuovere in sicurezza le informazioni sensibili dal proprio codice sorgente e, di conseguenza, dalla history del progetto.

Si raccomanda di eseguire alcuni controlli di base sulla qualità e sulla sicurezza del codice prodotto, utilizzando strumenti disponibili gratuitamente. Ad esempio, molti IDE moderni includono già strumenti di linting per controllare l'assenza sia di errori che di warning, per controllare l'esistenza della documentazione, eventuale codice non utilizzato, variabili non inizializzate e così via. È fortemente consigliato l'utilizzo di questo tipo di strumenti già in fase di sviluppo per trovare e rimuovere i problemi in fase embrionale. Inoltre, si suggerisce di inserire questi controlli (ad es., linting) anche nella propria toolchain di build. La ridondanza, in questo caso, può essere d'aiuto.

Numerosi sono gli strumenti di audit gratuiti che effettuano controlli di sicurezza e di qualità ulteriori per progetti di *software* libero. Come supporto per la scelta è possibile controllare le comparazioni degli strumenti sia FLOSS che di mercato in queste liste:

- Source Code Analysis Tools[33]
- Source Code Security Analyzers[34]

Come vedremo nel capitolo dedicato, alcuni di questi strumenti sono facilmente integrabili all'interno di diverse piattaforme di code hosting o di Continuous Integration: questo permette di eseguire una serie di controlli automatici per ogni operazione che viene eseguita sul repository (ad esempio, ad ogni commit) ed è quindi considerata una buona pratica.

4.3.2 Gestione delle dipendenze

Spesso, nello sviluppo di software, gli sviluppatori usano librerie scaricate localmente nella propria directory di sviluppo. Questa modalità, sebbene possa sembrare utile in fase di testing o prototipazione nell'ambiente di sviluppo locale, è considerata una cattiva pratica nello sviluppo di applicativi moderni. Essenzialmente un approccio di questo tipo comporta una forte difficoltà nella gestione sul lungo periodo, diversi problemi nell'aggiornamento di ogni singola dipendenza e l'impossibilità di usare strumenti automatizzati di rilevazione di vulnerabilità.

È invece buona pratica gestire le dipendenze esterne del software da rilasciare usando il gestore di pacchetti del linguaggio usato (ad es. npm, composer, ecc.). In questo modo le dipendenze e le loro versioni saranno dichiarate all'interno di un file tracciato da git e presente nel repository (ad es., package.json, ecc.) il che rende la loro gestione e l'aggiornamento più facile e immediato. Per un approfondimento è possibile consultare la sezione dedicata della 12 factor app (https://12factor.net/it/dependencies).

Inoltre, anche in questo caso esistono degli strumenti automatizzati che facilitano non solo l'aggiornamento delle singole dipendenze, ma anche la segnalazione di potenziali vulnerabilità in una particolare versione utilizzata. Questi strumenti sono spesso disponibili direttamente all'interno delle piattaforme di code-hosting o possono essere trovati sotto forma di plugin facilmente installabili. Le singole dipendenze vengono analizzate e, nel caso vi siano delle potenziali vulnerabilità, il maintainer verrà notificato e spesso la soluzione sarà suggerita in automatico.

[32] https://help.github.com/en/articles/removing-sensitive-data-from-a-repository
[33] https://www.owasp.org/index.php/Source_Code_Analysis_Tools
[34] https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html

Tra gli strumenti disponibili è possibile citare:

- npm-audit[35], strumento FOSS per la gestione delle dipendenze e notifica delle vulnerabilità per Javascript/Node.js
- Dependency Scanning GitLab[36], scanner integrato nella piattaforma (solo alcune versioni)
- Dependabot[37] - ora integrato in GitHub
- Snyk[38], sistema proprietario in SaaS disponibile gratuitamente per progetti di *software* libero.

Molto *software* libero moderno è composto da numerosissime dipendenze, come ad esempio librerie di terze parti, e pensare di aggiornarle indipendentemente senza utilizzare degli strumenti di supporto è un'operazione molto esosa e incline ad errori. È perciò fortemente consigliato adottare le buone pratiche qui descritte sia per la gestione delle dipendenze tramite il gestore dei pacchetti che per quanto riguarda gli strumenti di supporto messi a disposizione dalle varie piattaforme.

4.3.3 Responsabilità di terzi

Le licenze chiariscono che gli autori del software non hanno nessuna responsabilità in merito alla completezza e funzionalità, e che è sempre l'utente a doversi assumere la responsabilità per l'adozione di tale codice.

Ad esempio, questo è quanto prevede la BSD-3[39]:

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

È dunque compito di chi sviluppa e mantiene il software rilasciato in come *software* libero assicurarsi che sia sicuro, privo di bug o di vulnerabilità, e quindi adottare strumenti finalizzati alla qualità del codice, come ad esempio: unit tests, CI, auditing, checklist per feature parity. Problemi o limitazioni note devono essere accuratamente documentati come issue, e nei casi più importanti anche esplicitati nel README.

4.3.4 I file da inserire nel repository

Il repository di un progetto libero contiene solitamente numerosi file. In particolare, è buona pratica inserirne alcuni come il README, il file LICENSE o il file AUTHORS, per permettere al visitatore di capire più nel dettaglio la natura di tale progetto senza dover leggere fin da subito il codice sorgente. Vediamo ora questi file più in dettaglio

README

Ogni repository deve avere nella root un file README, solitamente in formato Markdown (.md). È buona norma che questo file contenga:

[35] https://docs.npmjs.com/cli/v6/commands/npm-audit
[36] https://docs.gitlab.com/ce/user/application_security/dependency_scanning/
[37] https://dependabot.com/
[38] https://snyk.io/
[39] https://opensource.org/licenses/BSD-3-Clause

- il **titolo** del repository ed un **sottotitolo descrittivo**;
- **descrizione estesa** del repository in un linguaggio comprensibile anche dai non addetti ai lavori (evitare acronimi e gergo tecnico), in particolare:
  - **contesto** (mai darlo per scontato);
  - **finalità**;
  - **beneficiari e casi d'uso**;
- eventuali **screenshot**;
- link alla **pagina del progetto** su Developers Italia e ad eventuali altre pagine istituzionali relative al progetto;
- link ad eventuale **documentazione**. Nel caso di documentazione multilingua, link alla versione generica (ad es. https://docs.italia.it/italia/anpr/anpr, non https://docs.italia.it/italia/anpr/anpr/it/stabile/index.html ;
- spiegazione della **struttura del repository** anche a beneficio dei potenziali contributori (struttura delle directory e dei branch);
- **istruzioni per l'installazione**:
  - requisiti e dipendenze;
  - build system necessario;
  - comandi per la compilazione o il deployment, possibilmente automatizzati da uno script/Makefile;
  - se il software prevede rilasci periodici, link per il download (anche sotto forma di badge);
- eventuali indicazioni sullo **stato del progetto**:
  - stato di alpha/beta/stable eccetera;
  - importanti **limitazioni** o **known issues**;
  - eventuale assenza di maintainer attivi, con l'invito a candidarsi;
  - eventuale stato di abbandono del software, o riferimento a versioni successive;
- badge:
  - **canale di comunicazione** relativo al repository (come in questo esempio[40]) e link al sito per l'invito (laddove possibile):

```
[![Join the #spid-perlchannel](https://img.shields.io/badge/Slack%20channel-
↪%23spid--perl-blue.svg?logo=slack)](https://developersitalia.slack.com/
↪messages/C7ESTMQDQ)

[![Getinvited](https://slack.developers.italia.it/badge.svg)](https://slack.
↪developers.italia.it/)

[![SPID onforum.italia.it](https://img.shields.io/badge/Forum-SPID-blue.
↪svg)](https://forum.italia.it/c/spid)
```

- eventuali sistemi di Continuous Integration (ad es., CircleCI), code coverage ed altre metriche;
- eventuale immagine Docker;
- **nomi** degli autori originali del software, dei maintainer (anche passati) e degli eventuali contributori principali.

Un README ben curato è fondamentale per incentivare l'adozione e lo sviluppo collaborativo di un progetto. Per ispirazione si può consultare questa lista di esempi[41].

[40] https://img.shields.io/badge/Slack%20channel-%23spid--wordpress-blue.svg
[41] https://github.com/matiassingers/awesome-readme

Un dettaglio delle sezioni da inserire nel file README è presente a questo indirizzo[42] e, per nuovi progetti, è possibile utilizzare questo template[43]. Infine, qui[44] si trova una checklist che guida alla verifica della completezza e dell'efficacia del README creato.

AUTHORS

Il file AUTHORS permette di individuare chi ha lavorato a un determinato progetto e potenzialmente contiene anche un riferimento temporale. Questo è fondamentale per la gestione dei copyright. Esistono diverse modalità di gestione del copyright e questo file aiuta a chiarire anche questo aspetto. Developers Italia, ad esempio, adotta un modello a copyright diffuso: ai contributori non è richiesto di devolvere tutti i diritti, pertanto ai fini dell'attribuzione fa fede lo storico dei commit di git e quanto specificato nel file AUTHORS. **Nota bene**: in presenza di contributi esterni nessuno detiene la totalità del copyright e quindi non è generalmente consentito il relicensing sotto altre licenze non compatibili con quella attuale.

- Esempio di AUTHORS file di developers.italia.it: https://github.com/italia/developers.italia.it/blob/master/AUTHORS

publiccode.yml

Tutti i repository di *software* libero sviluppati da una Pubblica Amministrazione italiana devono contenere un file YAML[45] denominato publiccode.yml che contiene le informazioni utili al popolamento del catalogo di Developers Italia[46]. publiccode.yml è un formato di metadatazione del software nato in Italia, ma in corso di adozione internazionale. Questo tipo di file può in realtà essere adottato da chiunque e dà la possibilità a tutti gli applicativi liberi di essere inseriti nel catalogo di Developers Italia, dunque proponendo il proprio lavoro alla considerazione delle Pubbliche Amministrazioni italiane. Tale file contiene diverse chiavi che possono essere facilmente compilate sia a mano che grazie ad un editor online[47] messo a disposizione dal Dipartimento per la trasformazione digitale.

LICENSE

Al software deve essere applicata una delle licenze approvate da Open Source Initiative[48] (le Linee Guida sull'acquisizione e il riuso di software per le Pubbliche Amministrazioni ne suggeriscono alcune in particolare[49], per consentire la massima riusabilità).

Al fine di applicare la licenza scelta al materiale da rilasciare è necessario creare nella root (cartella radice) del repository un file denominato LICENSE, contenente il testo integrale della licenza scelta, senza alcuna modifica. I testi originali sono disponibili su SPDX License List[50]. Sempre le Linee Guida specificano che è obbligatorio indicare la licenza applicata tramite espressione (o codice) SPDX all'inizio di ogni file sorgente, in modo che sia possibile effettuare una metadatazione automatica delle licenze usate.

Prima di scegliere una licenza per il proprio progetto è fondamentale effettuare dei controlli sulle dipendenze del proprio software. Infatti, seppur raramente, è possibile che alcune dipendenze o alcune parti di esse siano coperte da licenze più o meno restrittive che possano quindi risultare incompatibili con altre. È dunque necessario effettuare dei controlli prima di pubblicare il proprio software con una data licenza. Esistono degli strumenti che facilitano queste operazioni come ad esempio:

[42] https://github.com/italia/readme-starterkit
[43] https://github.com/italia/readme-starterkit/blob/master/README.template.md
[44] https://github.com/ddbeck/readme-checklist/blob/master/checklist.md
[45] https://yaml.org/
[46] https://developers.italia.it/it/software
[47] https://publiccode-editor.developers.italia.it/
[48] https://opensource.org/licenses
[49] https://docs.italia.it/italia/developers-italia/lg-acquisizione-e-riuso-software-per-pa-docs/it/stabile/attachments/allegato-d-guida-alle-licenze-open-source.html
[50] https://spdx.org/licenses/

- License Checker[51] per npm
- Pivotal License Checker[52]
- FOSSA[53] (servizio SaaS)

Nota bene: il risultato di queste analisi automatiche non è comunque utilizzabile al pari del al parere di un profilo legale specializzato in materia. Laddove vi fossero dei dubbi o delle incomprensioni è buona norma richiedere una perizia e un parere legale onde evitare di infrangere le norme sul diritto d'autore o violare i termini di una licenza.

Esistono diverse modalità di applicazione delle licenze ai singoli file. Per conoscere la specifica REUSE, si consiglia la lettura della guida dedicata[54].

.gitignore

Questo file permette di configurare la propria istanza di git in locale in modo tale da ignorare alcuni file e non "tracciarli". In questo modo sarà ad esempio possibile separare i file sorgente dai file oggetto frutto di una compilazione in locale oppure i file di swap o file temporanei sui quali si sta lavorando in locale. L'utilizzo corretto di questo file rappresenta una buona pratica perché evita che alcuni file che non dovrebbero essere pubblici vengano inseriti per sbaglio nel repository remoto. Un altro esempio rappresenta i file di configurazione del software contenenti informazioni private (ad es., i file .env): questi file non devono essere pubblicati e dunque non devono essere inseriti nel .gitignore.

Esempio di .gitignore file: https://github.com/italia/developers.italia.it/blob/master/.gitignore

File di progetto

Perché un progetto di *software* libero sia davvero in grado di generare l'impatto previsto è importante che all'interno del suo (o suoi) repository non vi siano solo i file contenenti il codice sorgente, ma siano esposti -e documentati- tutti i file di "contorno" che permettono al codice di essere effettivamente compilato (laddove necessario) ed eseguito. Capita spesso, purtroppo, di incappare in progetti che potrebbero essere interessanti e avere le potenzialità per raccogliere numerosi contributori esterni, per poi scoprire che risulta estremamente complesso o impossibile eseguire tale software all'interno del proprio ambiente di sviluppo. Ecco dunque che diventa fondamentale inserire all'interno del repository pubblico tutti i file (ad es., Makefile) che permettono di effettuare queste operazioni di compilazione ed esecuzione corredati da documentazione puntuale.

Esistono delle iniziative internazionali che si prodigano per illustrare le modalità di gestione di un progetto con la finalità di garantire sistemi di build riproducibili, come ad esempio https://reproducible-builds.org/.

Nota bene: un progetto di *software* libero il cui funzionamento non possa essere correttamente riprodotto su altri sistemi al di fuori di quelli controllati dallo sviluppatore, non solo diminuisce drasticamente il suo impatto, ma viola anche parti di alcune licenze OSI compliant che prevedono che l'utente finale debba avere la possibilità di eseguire il software. In tal senso, è importante anche dichiarare l'eventuale dipendenza da sistemi proprietari sia nella documentazione che nel file publiccode.yml, in modo da notificare l'utente finale e semplificare il suo processo di acquisizione.

4.3.5 Rilascio

Una volta creato il repository pubblico è necessario pubblicare tutto il codice sorgente contenente i file già elencati più sopra. Se il software da pubblicare è già stato scritto in modalità "chiuso", è importante riportare tutti i commit passati in modo da facilitare l'interazione con i contributori terzi. Effettuare un singolo commit in modalità "bulk" è

[51] https://www.npmjs.com/package/license-checker
[52] https://github.com/pivotal/LicenseFinder
[53] https://fossa.com/
[54] https://reuse.software/practices/2.0/

considerata una cattiva pratica, proprio perché non rende trasparenti le singole modifiche effettuate nel tempo dagli sviluppatori.

I messaggi di commit sono importanti in quanto comunicano in breve quali operazioni sono state effettuate dallo sviluppatore. Esistono anche in questo caso diversi approcci e buone pratiche. Una di queste è Conventional Commits[55] che presenta il seguente formato:

```
<type>[optional scope]: <description>

[optional body]

[optional footer(s)]
```

Lo scopo è quello di trasmettere molte informazioni in modo semplice, che siano leggibili e interpretabili, sia da essere umani che da sistemi automatici. Anche in questo caso sarà dunque possibile sfruttare degli automatismi integrabili nelle piattaforme di code hosting o di Continuous Integration.

Dopo il rilascio, tutte le operazioni successive che verranno effettuate saranno pubbliche. Per prevenire la pubblicazione di altre informazioni sensibili oppure evitare di commettere errori prima del push sul repository è considerata una buona pratica l'utilizzo di sistemi di controllo pre-commit. Tali sistemi, come ad esempio https://pre-commit.com/, si possono installare negli ambienti locali di sviluppo e permettono di eseguire una serie di controlli prima di effettuare il commit o il push: in questo modo la pubblicazione viene bloccata fino a che il problema non è risolto.

4.4 Gestire un software nel suo ciclo di vita

4.4.1 L'importanza del ruolo del maintainer

Il *maintainer* di un progetto di *software* libero è un ruolo con molteplici funzioni organizzative e pratiche. Essenzialmente, il maintainer ha il completo controllo di un repository, potendo effettuare le operazioni di *clone*, *push*, *merge*, creazione di nuovi *branch* e così via. È la persona responsabile della manutenzione ordinaria del contenuto del repository, ovvero colui che si impegna a seguire l'evoluzione del software, effettuando le operazioni tipiche di *housekeeping*: si assicura che i diversi approcci vengano seguiti, assicura che il gitflow scelto venga rispettato, che il versionamento sia coerente con le scelte e così via. Inoltre il maintainer di un progetto libero nella maggioranza dei casi ha a che fare con la community, ovvero l'insieme degli utenti o dei contributori che interagiscono con il repository. Questa figura, infatti, segue l'evoluzione delle issue, dirige correttamente i contributori facendo seguire quanto presente nel file `CONTRIBUTING`, e opera i merge delle Pull Request. A volte il maintainer fa parte del team di sviluppo e può quindi effettuare direttamente le review dei contributi mentre altre volte si avvale del supporto degli sviluppatori per effettuare revisioni e consolidare i contributi. Il maintainer è dunque un ruolo fondamentale con numerose responsabilità sulla vitalità del progetto.

Non sempre il maintainer è unico, spesso infatti questo ruolo è assunto da un team di persone che hanno lavorato sul progetto per un determinato lasso di tempo e che dunque sono considerati affidabili.

4.4.2 Versionamento e rilasci

Esistono diversi approcci al versionamento del software e alla gestione dei rilasci.

Un possibile approccio è quello di non effettuare una vera e propria pubblicazione di rilasci ufficiali e numerati in quanto git è già di per sé un sistema di versionamento ed in molti casi è sufficiente. Questa scelta è percorribile nei seguenti casi:

- progetti di piccole dimensioni;

[55] https://www.conventionalcommits.org/en/v1.0.0/

- componenti destinati ad altri sviluppatori (asset grafici, toolkit, template…), unicamente nel caso di progetti interni;
- progetti che adottano strategie di Continuous Delivery[56] per i quali vale quindi sempre e solo l'ultima versione.

Al contrario, i rilasci ufficiali tradizionali sono tipicamente necessari in queste situazioni:

- software destinato ad un'ampia platea di utenti finali;
- software che richiede compilazione e quindi distribuzione in forma binaria;
- software che necessita di testing non automatizzabile (audit ecc.);
- software da rilasciare anche su altre piattaforme di distribuzione (npm, Maven Central, ecc.);
- software allineato a schemi di versionamento esterni (ad esempio per aggiornamenti della normativa);
- software per il quale viene mantenuta documentazione separata (in modo da sapere quale versione consultare per una determinata versione del software);
- software che può prevedere modifiche non retrocompatibili, con una base di utenti che può rimanere potenzialmente legata a precedenti versioni.
- software da utilizzare in architetture complesse (ad es. i componenti di un sistema a microservizi).

Se si sceglie di procedere con una strategia di rilascio, è necessario essere molto rigorosi nell'eseguire i rilasci con regolarità e tempestività (*release early, release often*): una volta adottati, non sono facoltativi.

Ad esempio, nella piattaforma GitHub è possibile usare la funzione delle *milestone* per associare le issue ai futuri rilasci, in modo da documentare cosa manca per procedere con un rilascio. Si deve infatti evitare che vi siano delle novità nella base di codice non ancora rilasciate ufficialmente, senza che vi sia alcuna indicazione di cosa stia aspettando il *maintainer* a rilasciarle.

A seconda che si adotti uno dei due modelli o l'altro, il significato del branch *master (o main)* è diverso. In assenza di rilasci, è solitamente sottinteso il principio dello *stable master branch*, ovvero: il branch master deve sempre contenere codice stabile ed utilizzabile, pertanto non può essere usato per sviluppo di funzionalità non testate. È necessario separare lo sviluppo in branch separate da portare su master attraverso *Pull Request* separate. Eventualmente, nei casi più delicati si può anche pensare ad un branch *develop* verso il quale indirizzare le *Pull Request*, e poi avere un processo di testing più approfondito per portare le modifiche da *develop* a *master*.

Al contrario, in presenza di rilasci non è strettamente richiesto che il branch master sia stabile (a quello scopo servono appunto i rilasci, spesso accompagnati dalla creazione di un branch stabile), ma è comunque bene adottare la pratica di lavorare in branch separati quando si eseguono refactoring o comunque grandi modifiche.

Nel caso in cui si scelga di associare una numerazione ai singoli rilasci, il versionamento semantico (https://semver.org/[57]) è un buon esempio da seguire quando possibile.

Il versionamento semantico definisce un numero di versione composto da tre numeri: `MAJOR.MINOR.PATCH`. I corrispondenti numeri di versione devono essere incrementati in questo modo:

- `MAJOR`: quando si apportano modifiche API/ABI incompatibili con le precedenti;
- `MINOR`: quando si aggiunge funzionalità in modo compatibile con le versioni precedenti;
- `PATCH`: quando si apportano correzioni di bug compatibili con le versioni precedenti.

Ulteriori prefissi o postfissi al numero di versione (come pre-release e build) possono essere utilizzati per segnalare versioni particolari in momenti chiave del progetto.

[56] https://en.wikipedia.org/wiki/Continuous_delivery
[57] http://semver.org/

4.4.3 Issue

L'issue tracker è uno degli strumenti messi a disposizione da molte piattaforme di code hosting per permettere agli interessati di comunicare con il team di sviluppo. Esempi di issue possono essere: segnalazioni di problemi, richieste di aggiunta di nuove funzionalità o discussioni relative al progetto. A questo proposito, è molto importante che lo strumento sia disponibile e aperto a contributi. Se questa condizione non dovesse essere soddisfatta è possibile indicare nel README il percorso per raggiungere lo strumento di issue tracking da usare come riferimento.

I maintainer hanno il compito di monitorare la sezione delle issue e di dare un tempestivo riscontro, non necessariamente entrando nel merito della questione: avere a che fare con un repository seguito e non abbandonato incentiva gli altri membri della community ad entrare nella discussione e contribuire "investendo" il proprio tempo nel progetto.

Strumenti di tracking esterni

Se i maintainer adottano strumenti esterni per tracciare le proprie attività ed allocare le risorse (ad es. Jira, Trello) potranno riportare su questi ultimi i riferimenti alle issue su cui intendono intraprendere attività, ma tali strumenti non sono sostitutivi delle issue dove è comunque opportuno dare aggiornamenti periodici pubblici sulle attività in corso, a scopo di trasparenza e coinvolgimento. Le issue costituiscono una *knowledge base* utile anche una volta chiuse poiché vengono indicizzate dai motori di ricerca e aiutano i futuri contributori a capire perché sono state fatte certe scelte.

Ove possibile, dovrebbero essere utilizzati strumenti specifici per la segnalazione e la tracciabilità delle issue. Molti siti di hosting di progetti offrono questa funzionalità integrata. In alternativa, la funzionalità può essere fornita da progetti autonomi come:

- Trac - https://trac.edgewall.org/
- Redmine - https://www.redmine.org/

Richiesta di aiuto

All'interno di molti strumenti di "issue tracking" è possibile assegnare delle etichette (label) alle singole issue. È buona pratica utilizzare queste label sia per segnalare che un maintainer ha effettivamente controllato il contenuto della issue (*triaging*) che per comunicarne in modo rapido il contenuto. Alcune label, come ad esempio "bug" o "help wanted" sono auto-esplicative e servono per organizzare al meglio la gestione delle singole issue.

Segnalazioni di bug

La segnalazione va tenuta aperta fino alla risoluzione. Salvo correzioni semplici,è buona pratica chiedere all'utente che ha segnalato il problema di chiudere la issue dopo aver verificato l'efficacia della correzione, invece che chiuderla d'ufficio.

È buona prassi aggiornare la issue con l'avanzamento della risoluzione, incluse eventuali riflessioni ed esplorazioni o collegamenti a pagine web correlate (ad es. issue esterne). Questo incentiva l'aiuto da parte degli altri utenti e costituisce una forma di documentazione per ricostruire a posteriori le scelte fatte. Anche nel caso di interazioni via chat è bene aggiornare la issue a beneficio di altri lettori. In altre parole, è importante ricordarsi che **le issue non sono una canale di comunicazione individuale, ma una forma di documentazione a beneficio dell'intera community**.

Tutta l'interazione con gli utenti deve essere svolta pubblicamente all'interno della issue e deve essere spostata su canali privati (ad es. helpdesk) solo limitatamente ad eventuali informazioni riservate legate più alla messa in esercizio che al progetto software in sé.

È inoltre raccomandato citare il numero della issue nel messaggio di commit che la risolve, in modo da legare la discussione e le origini della issue e la sua risoluzione.

Richieste di miglioramento (Feature request)

I maintainer non sono tenuti a portare avanti tutte le richieste di miglioramento o di nuove funzionalità, ma le possono valutare insieme ai Project Leader. In ogni caso è bene lasciare aperte le feature request, purché ritenute compatibili con la roadmap del progetto, in modo che si possano raccogliere ulteriori commenti ed essere implementata da contributori.

Gestione delle issue

Il processo di gestione di una issue da parte di un maintainer si articola solitamente in più passaggi:

1. identificazione del problema, assegnazione di un'etichetta (label) e risposta alla issue (*triaging*);
2. assegnazione della issue ai componenti del team (*assign*);
3. analisi del problema e individuazione di soluzioni/raccomandazioni (*fix*);
4. aggiornamento della issue con le soluzioni/raccomandazioni (*update*);
5. implementazione e monitoraggio e controllo delle attività per il superamento della issue (*validate*);
6. chiusura della issue attraverso la verifica che le azioni pianificate siano state implementate ed il problema risolto. Nel caso non lo fosse, il ciclo viene ripetuto a partire dal punto 4 (*close*).

Una volta chiusa, la issue continua comunque ad essere visibile e contraddistinta da una URL univoca. Questo è importante per garantire innanzitutto visibilità sulle azioni che hanno portato alla sua chiusura, ma è anche fondamentale per costruire una *knowledge base* pubblica, indicizzata dai motori di ricerca. È inoltre fondamentale in fase di triage di nuove issue in quanto è sempre possibile far riferimento ad un'azione compiuta in passato che ha risolto un problema simile.

4.4.4 Accettare i contributi dopo il rilascio

Una volta rilasciato il codice sorgente in modo pubblico è possibile che dei contributori di terze parti, ovvero non facenti parte del contingente originale che ha sviluppato il software, apportino delle modifiche al codice.

Tali contributi possono avvenire tramite un meccanismo chiamato *Pull Request (PR)* o *Merge Request* (la nomenclatura varia al variare della piattaforma utilizzata per la pubblicazione). La *Pull Request* è una richiesta, fatta all'autore originale di un software, di includere modifiche al suo progetto.

Quando una nuova Pull Request viene aperta, la piattaforma notifica al maintainer che è necessario affrontare le operazioni di revisione.

4.4.5 Integrazione Continua (Continuous Integration)

I sistemi di integrazione continua (Continuous Integration, CI) sono utili per ridurre i tempi di ricerca di bug, consentendo di effettuare test automatizzati dell'intera code base. Ciò è particolarmente utile per i progetti che coinvolgono una grande comunità di sviluppatori. Tuttavia è buona norma dotarsi di tali strumenti fin dall'inizio dello sviluppo.

Inoltre, ogni modifica proposta tramite il meccanismo delle Pull/Merge Request deve "passare" una serie di test automatici prima di essere anche solo presa in considerazione dai maintainer. I processi di CI rappresentano un supporto di fondamentale importanza sia in fase di sviluppo, per identificare eventuali problemi o migliorare la qualità generale del codice, che in fase di analisi dei contributi esterni, per validare le proposte e evitare di effettuare numerose interazioni con i contributori su codice che non è stato accuratamente allineato con le esigenze del progetto.

Esistono molti esempi di questi sistemi di integrazione continua. Tra i più utilizzati citiamo:

- Jenkins CI[58], particolarmente adatto a deployment locali (installazione on-premises)
- Gitlab CI[59], integrato con la piattaforma di code-hosting GitLab
- Circle CI[60], servizio proprietario, disponibile come SaaS gratuitamente per i progetti di *software* libero
- GitHub Actions[61]

La particolarità di questi sistemi è che si integrano perfettamente con le piattaforme di code hosting più comuni e permettono quindi di eseguire una serie di test e/o controlli automatici per ogni singola azione che viene eseguita sulla codebase (ad es., commit, merge etc.).

Normalmente la configurazione di questi sistemi richiede la presenza di un semplice file di configurazione da posizionare nella cartella radice del repository. In tal senso, il Team di Developers Italia ha realizzato alcuni template pronti all'uso, per testare ad esempio il file publiccode.yml, ma che si possono facilmente estendere per essere utilizzati in altri contesti.

- Qui puoi trovare il publiccode-parser-orb per CircleCI: https://github.com/italia/publiccode-parser-orb
- Qui puoi trovare la action da integrare nel tuo repository github: https://github.com/italia/publiccode-parser-action

Elenchiamo di seguito le principali possibilità che un sistema di Continuous Integration offre:

- esecuzione di test automatici (sia unitari che End-to-End);
- audit di sicurezza con sistemi di SAST;
- analisi della qualità del codice;
- analisi della quantità di codice coperto da test unitari (code coverage), importantissima funzionalità per garantire un codice di qualità;
- analisi dei messaggi di commit (commit-lint).

Oltre a queste analisi, che possono offrire una panoramica sullo stato di salute del codice in ogni momento utile,c'è anche la possibilità di automatizzare numerose operazioni come ad esempio:

- effettuare dei rilasci automatici (tag e release);
- effettuare la compilazione dei sistemi con le informazioni di produzione (build);
- effettuare delle operazioni di interazione con altri sistemi web tramite API, ad esempio le immagini docker possono essere inviate a dei registri pubblici.

Infine, l'ultimo passaggio interessante, anche denominato Continuous Deployment (CD), consiste nell'utilizzare questi strumenti anche per effettuare il deploy, quindi la messa in produzione, del sistema. Il concetto fondamentale in questo caso è quello di utilizzare la stessa codebase per tutti i deployment che verranno effettuati (ad esempio, development, staging, pre-prod, live) dove a variare saranno solo le risorse connesse all'applicazione (ad es., i DB) tramite opportuni file di configurazione (maggiori approfondimenti sono disponibili su 12factor app[62]).

Abbiamo visto come i sistemi di CI possono facilitare la vita dello sviluppatore, aumentare la qualità del codice, supportare le analisi di sicurezza e, infine, effettuare la messa in opera dell'intero sistema in modo completamente automatico: possono sicuramente essere considerati il coltellino svizzero dello sviluppo di *software* libero.

[58] https://www.jenkins.io/
[59] https://about.gitlab.com/product/continuous-integration/
[60] https://circleci.com/
[61] https://github.com/features/actions
[62] https://12factor.net/it/codebase

CAPITOLO 5

Bibliografia / Sitografia

Di seguito alcuni riferimenti utili sui quali approfondire queste tematiche

- Standard for Public Code: https://standard.publiccode.net/
- Google Open Source: https://opensource.google/docs/
- GitHub Guides: https://guides.github.com/
- GitLab Guides: https://docs.gitlab.com/ce/gitlab-basics/[63]
- 12 Factor App: https://12factor.net/it/

[63] https://docs.gitlab.com/ee/gitlab-basics/

CAPITOLO 6

Glossario

CAD Il Codice dell'Amministrazione Digitale, ovvero il Decreto Legislativo 7 marzo 2005, n. 82[64], è il testo unico che riunisce e organizza le norme riguardanti l'informatizzazione della Pubblica Amministrazione italiana

Code Hosting (strumento di) Una piattaforma che consente la pubblicazione di codice sorgente, organizzato in più repository. Gli strumenti di code hosting offrono spesso anche funzionalità legate all'evoluzione di un *software* quali sistemi di ticketing (ovvero sistemi per tenere traccia di problemi o proposte di modifica), processi per la contribuzione di codice da parte di terzi, area per il download dei rilasci, ecc.

Ad esempio, GitLab, GitHub e Bitbucket sono piattaforme di code hosting popolari.

Codice sorgente Il codice sorgente (spesso detto semplicemente «sorgente») è il testo di un programma scritto in un linguaggio di programmazione (es. C o Visual Basic) dal quale si deriva il programma finale che l'utente usa. L'accesso al codice sorgente è fondamentale per poter modificare un programma.

Community Aggregazione di persone, fisiche e giuridiche, e risorse (ad esempio forum, chat e tecnologie per riunirsi e interagire in una località virtuale), dotata di regole e di una struttura, finalizzata alla realizzazione e/o gestione di un progetto comune.

Formato aperto (di dato) È un formato di rappresentazione dei dati, versionato, documentato esaustivamente e senza vincoli all'implementazione. Un formato aperto è riconosciuto da un ente di standardizzazione e mantenuto in modo condiviso tra più enti che forniscono implementazioni concorrenti, con un processo trasparente. Il formato deve rimanere consistente con la versione dichiarata.

JPEG, PNG e ODF sono alcuni esempi di formati aperti.

Interoperabilità In ambito informatico, la capacità di sistemi differenti e autonomi di cooperare e di scambiare informazioni in maniera automatica e strutturata, sulla base di regole condivise.

Licenza Il testo legale che permette al titolare dei diritti d'autore di un'opera (software, contenuti, dati, . . .) di concedere determinati diritti anche agli utilizzatori finali. In assenza di una licenza esplicita, si assume generalmente «tutti i diritti riservati» e gli utilizzatori finali non possono usare, studiare, modificare o diffondere l'opera senza un permesso scritto fornito dal titolare dei diritti.

[64] https://docs.italia.it/italia/piano-triennale-ict/codice-amministrazione-digitale-docs/

Licenza di software libero È una licenza che concede a chiunque usi un software, i diritti di uso, copia, modifica, distribuzione di copie anche modificate; per fare ciò, è necessario anche che il codice sorgente sia liberamente disponibile.

Lista di licenze di software libero[65].

Nel Codice dell'Amministrazione Digitale è chiamata "licenza aperta".

Lock-in Fenomeno di natura tecnica ed economica in cui un generico utente non riesce a svincolarsi da una scelta tecnologica precedentemente effettuata. Tale incapacità è tipicamente causata degli elevati costi legati al cambio di tecnologia ma, in molti casi, può anche dipendere dall'adozione di soluzioni proprietarie che impediscono di effettuare migrazioni. L'utilizzo di formati aperti per il salvataggio dei dati, e l'accesso libero a questi dati (soprattutto nel caso di soluzioni SaaS) sono prerequisiti per evitare fenomeni di lock-in.

Repertorio o Repository o Deposito (di codice sorgente) All'interno di uno strumento di code-hosting, un repository è l'unità minima di contenimento del codice sorgente di un software. Il termine «repertorio» è la sua traduzione italiana (usata per esempio nel CAD Art 69, comma 1).

Riuso Nel contesto di questo documento, si intende il processo delineato dal CAD (art. 69) con il quale una amministrazione distribuisce («mettere a riuso») un *software* di cui ha titolarità come *software* libero, a favore di altre amministrazioni che possano utilizzarlo («prendere a riuso»). Tutto il *software* a riuso è *software* libero, ma non tutto il *software* libero è a riuso (poiché non tutto il *software* libero è di titolarità di una amministrazione).

SaaS *Software* as a Service. Indica una modalità di distribuzione del *software* in cui questo non viene installato sui computer o sui server del cliente finale ma che viene fruito direttamente dai server del fornitore al quale si delegano gestione dell'infrastruttura, manutenzione, aggiornamenti, etc. Tipicamente, si tratta di applicativi web.

Software libero È una modalità con cui il *software* può essere concesso in licenza (vedi Licenza di software libero)

Deve rispettare queste caratteristiche:

- Libertà di eseguire il programma come si desidera, per qualsiasi scopo
- 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 di ridistribuire copie in modo da aiutare gli altri
- Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti apportati (e le versioni modificate in genere), in modo tale che tutta la comunità ne tragga beneficio. L'accesso al codice sorgente ne è un prerequisito.

La definizione completa di software libero[66].

Nel linguaggio comune è usato a volte il termine *open source* come sinonimo. Malgrado non tutto il software *open source* sia anche software libero, lo è nella maggior parte dei casi.

Software proprietario *Software* che ha restrizioni sul suo utilizzo, sulla sua modifica, riproduzione o ridistribuzione, imposti dal titolare dei diritti di sfruttamento economico, cioè l'autore o - in caso di cessione dei diritti patrimoniali - il cessionario dei diritti in questione.

TCO Total Cost of Ownership: approccio utilizzato per valutare tutti i costi del ciclo di vita di una risorsa IT calcolato su una finestra temporale adeguata al contesto della valutazione e che include il costo di migrazione verso altra soluzione (eg., acquisto, installazione, gestione, manutenzione e smantellamento). Il calcolo del TCO è basato sulla considerazione che il costo totale di utilizzo di una risorsa IT non dipende solo dai costi di acquisto, ma anche da tutti i costi che intervengono durante l'intero ciclo di vita della risorsa, comprese le attività di dismissione della stessa.

[65] https://www.gnu.org/licenses/license-list.it.html#SoftwareLicenses
[66] https://www.gnu.org/philosophy/free-sw.it.html