RELAZIONE SUL PROGETTO OPEN SOURCE PARANCOE

di Tonon Michele e Benna Diego

Indice generale

1. Introduzione

In questa relazione analizziamo il progetto Parancoe, non dall'aspetto tecnico, ma lo studieremo dal punto di vista dell'open source.

Per la realizzazione di questa relazione abbiamo intervistato Lucio Benfante, project leader del progetto, approfondendo poi gli argomenti dal punto di vista teorico studiati a lezione durante il corso di Tecnologie Open Source.

2. Cos'è Parancoe

Parancoe è un'aggregazione di meta-framework in Java il cui scopo è dare agli sviluppatori un insieme di librerie pronte per costruire standard web application senza preoccuparsi della configurazione dei framework necessari. Parancoe è composto da un completo pattern MVC:

•Hibernate/JPA: è una piattaforma middleware open source per lo sviluppo di applicazioni Java che fornisce un servizio di Object-relational mapping (ORM), ovvero che gestisce la rappresentazione e il mantenimento su database relazionale di un sistema di oggetti Java.

•Spring 2: è un framework open source per lo sviluppo di applicazioni su piattaforma Java. è una valida alternativa al modello basato su Enterprise JavaBeans (EJB). Rispetto a quest'ultimo, il framework Spring lascia una maggiore libertà al programmatore fornendo allo stesso tempo un'ampia e ben documentata gamma di soluzioni semplici adatte alle problematiche più comuni.

La prima versione venne scritta da Rod Johnson e rilasciata con la pubblicazione del proprio libro "*Expert One-on-One Java EE Design and Development*"..

Sebbene le peculiarità basilari di Spring possano essere adottate in qualsiasi applicazione Java, esistono numerose estensioni per la costruzione di applicazioni *web-based* costruite sul modello della piattaforma Java EE. Questo ha permesso a Spring di raccogliere numerosi consensi e di essere riconosciuto anche da importanti vendor commerciali quale framework di importanza strategica.

•Spring MVC: è un framework che fornisce una completa implementazione del pattern architetturale MVC(Model View Control).

•Supporto di AJAX (Asynchronous JavaScript and XML): è una tecnica di sviluppo per la realizzazione di applicazioni web interattive (Rich Internet Application). Utilizzando la libreria DWR per utilizzarlo da Java.

L'idea era di organizzare in maniera coerente alcuni degli strumenti che alcuni membri del JUG Padova già utilizzavano con soddisfazione per lo sviluppo di applicazioni Web, integrandoli in modo da avere un ambiente di sviluppo il più possibile produttivo e comodo per lo sviluppatore, con una filosofia molto simile a quella di Ruby on Rails (con gli ovvii limiti imposti da un linguaggio compilato).

2.1. Punti forti

•Sappiamo che generalmente lo sviluppatore si annoia a redarre documentazione e a perdere tempo nella configurazione. Uno dei punti forti di Parancoe è che necessita di una configurazione minima: lo sviluppatore scrive codice, non passa il tempo a scrivere configurazione. Per esempio non deve interessarsi della connessione al database o allo stato della connessione poiché è tutto configurato in un file XML.

L'idea è quella di non dover scrivere "configurazione" durante lo sviluppo delle funzionalità, come troppo spesso avviene con quasi tutti i framework.

Ad esempio quando si usa Spring: si devono definire i bean nell'XML, quindi ci si mette 2 minuti a scrivere una classe Java e 10 per legarla alle altre classi con l'XML. è da considerare che questo XML probabilmente non cambierà quasi mai nella vita dell'applicazione, e se cambia probabilmente è perché viene cambiata l'applicazione.

Quindi non è realmente configurazione, ma sviluppo molto scomodo, noioso e costoso.

Altro esempio: Quando scriviamo un controller e dobbiamo configurare da qualche parte (al di fuori della classe controller) qual è l'URL su cui viene invocato quel controller, i parametri passati, la vista che ne deve visualizzare il risultato, ecc. Anche in questo caso sono necessari 2 minuti di scrittura di codice Java e 10 di XML. E anche in questo caso la configurazione cambierà raramente con il tempo, a meno che non cambi l'applicazione (e l'applicazione durante lo sviluppo ovviamente cambia). Quindi si perde un sacco di tempo per cambiare cose sparpagliate in più file, spesso ripetute, e senza nessun aiuto da parte dei normali strumenti di sviluppo. Sono cose che si devono scrivere semplicemente per far funzionare il tutto.

Tipicamente con Parancoe si lavora su un unico file per ogni layer dell'applicazione, e praticamente mai su file XML, senza dover scrivere due volte la stessa informazione.

Il tutto si traduce nel non spendere il proprio tempo in quei 10 minuti di "configurazione", limitandosi ai 2 di codifica. Quindi minor tempo, ma anche maggior coerenza, meno "codice" su cui fare debug e possibilità di scrivere i test.

•Generalmente l'accesso a sorgenti di dati implica la conoscenza e l'utilizzo delle relative modalità di accesso. Questo in applicazione distribuite causa un dipendenza tra la logica di business (EJB, CORBA servant, …) e la logica di accesso ai dispositivi di persistenza come database relazionali (RDBMS), database object oriented (OODBMS).

L'intento del pattern DAO (Data Access Object) è di disaccoppiare la logica di business dalla logica di accesso ai dati. Questo si ottiene spostando la logica di accesso ai dati dai componenti di business ad una classe DAO rendendo gli EJB indipendenti dalla natura del dispositivo di persistenza. Questo approccio garantisce che un eventuale cambiamento del dispositivo di persistenza non comporti modifiche sui componenti di business. Inoltre legando il componente EJB ad un particolare tipo di data repository, ne si limita il riuso in contesti differenti. Parancoe permette di scrivere in molto semplice i DAO, che porta ad una scrittura molto semplice di un CRUD (Create, Retrieve, Update, Delete).

•Parancoe integra vari strumenti software e ha un'Impostazione che spinge lo sviluppatore ad usare Spring al meglio.

•Parancoe ha un sistema di fixtures per scrivere facilmente unit test sul database.

•Parancoe fornisce un sistema di plugin per arricchire facilmente di funzionalità un'applicazione.

•Parancoe offre un Archetype Maven che permette di iniziare subito a sviluppare la propria applicazione, partendo da una base già sufficientemente ricca e consolidata, con un meccanismo di build basato su Maven che la rende indipendente da uno specifico IDE.

2.2. Punti deboli

•Parancoe soffre ancora di un documentazione non sufficiente

•Lo sviluppo dell'interfaccia utente ancora forse troppo di basso livello, basato esclusivamente su JSP (ma questo non è necessariamente un punto debole).

•La massa critica di utilizzatori è ancora insufficiente, il che non permette di scoprire abbastanza in fretta i bug, e di introdurre nuove funzionalità e plugin.

3. Parancoe e l'open source

Essendo un progetto open source Parancoe soddisfa i seguenti criteri:

•La ridistribuzione di Parancoe è libera e gratuita

Quindi la sua licenza non potrà limitare nessuna persona dal vendere o donare Parancoe come componente di una distribuzione aggregata di software contenenti programmi di varia origine. La licenza non potrà richiedere royalties o altri pagamenti per tali vendite.

•Codice Sorgente

la libreria Parancoe deve include il codice sorgente, e permette la distribuzione così come per la forma compilata. Il codice è sempre disponibile nel sito del progetto di Parancoe.

Non si puo' migliorare un programma senza modificare il codice. Per questo motivo l'obiettivo del project leader di Parancoe è rendere l'evoluzione facile. Viene fatto il possibile per rendere facili le modifiche.

•Prodotti Derivati

La licenza di Parancoe deve permette modifiche e prodotti derivati, e permette loro di essere distribuiti sotto le stesse condizioni della licenza del software originale.

•Integrità del Codice Sorgente dell'Autore

La licenza di Parancoe può impedire la redistribuzione del codice sorgente in forma modificata solo se la licenza consentirà la distribuzione di patch files con il codice sorgente al fine di modificare il programma all'installazione. La licenza deve esplicitamente permettere la distribuzione del software costruito da un diverso codice sorgente. La licenza può richiedere che i lavori derivati abbiano un nome diverso o versione diversa dal software originale.

•Nessuna Discriminazione contro Persone o Gruppi

La licenza non discrimina alcuna persona o gruppo di persone.

•Nessuna Discriminazione contro Campi d'Applicazione

La licenza non impedisce a nessuno di far uso di Parancoe in un ambito specifico. Per esempio, non potrà impedire l'uso di Parancoe nell'ambito di un'impresa, o nell'ambito della ricerca genetica.

•Distribuzione della Licenza

I diritti allegati a un programma devono valere a tutti coloro cui il programma è redistribuito senza necessità dell'emissione di una addizionale licenza da parte dei licenziatari.

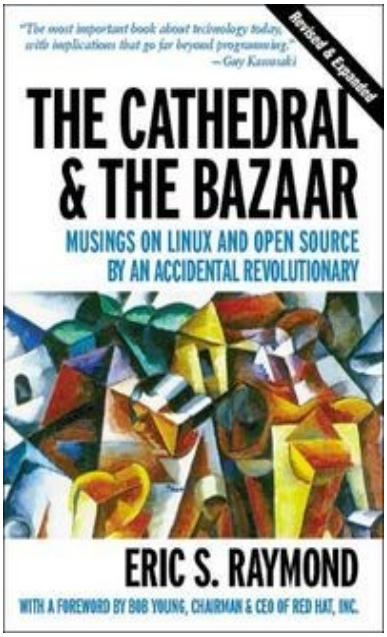
•La Licenza non deve essere Specifica a un Prodotto

I diritti allegati alla libreria Parancoe non devono dipendere dal fatto che la libreria faccia parte di una distribuzione particolare. Se Parancoe viene estratto da tale distribuzione e usato o distribuito nei termini della licenza del programma, tutte le parti a cui il programma viene ridistribuito devono avere gli stessi diritti garantiti in occasione della distribuzione originale del software.

•La Licenza non deve Porre Vincoli su Altro Software

La licenza non deve porre restrizioni su altro software che è distribuito insieme al software licenziato. Per esempio, la licenza non dovrà insistere che tutti gli altri programmi distribuiti sugli stessi supporti siano software open-source.

4. Parancoe esempio di modello a Bazaar



La Cattedrale e il Bazaar è un saggio sullo sviluppo del software scritto da Eric S. Raymond. Esso descrive un nuovo modello di sviluppo, il cui esempio più famoso ed efficace è la modalità di costruzione del Kernel Linux. L'autore per verificare le proprie ipotesi decide di utilizzare lo sviluppo collaborativo per il programma fetchmail e nel saggio viene descritta la genesi e lo sviluppo del progetto analizzando le interazioni con gli altri sviluppatori e i vantaggi rispetto al modello classico. Questo saggio viene usualmente considerato il manifesto del movimento open source.

Il saggio descrive in sostanza due contrapposte modalità di sviluppo del software libero:

- Nel modello a Cattedrale il programma viene realizzato da un numero limitato di "esperti" che provvedono a scrivere il codice in quasi totale isolamento. Il progetto ha una suddivisione gerarchica molto stretta e ogni sviluppatore si preoccupa della sua piccola parte di codice. Le revisioni si susseguono con relativa lentezza e gli sviluppatori cercano di rilasciare programmi il più possibile completi e senza bug. Il programma Emacs, il GCC e molti altri programmi si basano su questo modello di sviluppo.
- Nel modello a Bazaar il codice sorgente della revisione in sviluppo è disponibile

liberamente, gli utenti possono interagire con gli sviluppatori e se ne hanno le capacità possono modificare e integrare il codice. Lo sviluppo è decentralizzato e non esiste una rigida suddivisione dei compiti, un programmatore di buona volontà può modificare e integrare qualsiasi parte del codice. In sostanza lo sviluppo è molto più anarchico e libero, da qui il nome di modello a Bazaar. Il Kernel Linux e molti programmi utilizzano questo nuovo modello di sviluppo associativo.

La tesi centrale di Raymond è che "Dato un numero sufficiente di occhi, tutti i bug vengono a galla". Questa affermazione Raymond la chiama la "Legge di Linus" e ritiene questa legge il motivo centrale del successo del progetto del Kernel Linux. Prima dell'avvento di Linux si riteneva che ogni progetto di una certa complessità avesse bisogno di essere adeguatamente gestito e coordinato altrimenti si riteneva che il progetto sarebbe collassato sotto il peso di moltissime revisioni e modifiche prodotte da più persone e tra di loro incompatibili. Il progetto Linux invece riuscì a dimostrare che questo non accadeva e che anzi col crescere del numero di sviluppatori anche la qualità e l'affidabilità del software migliorava.

Il modello a Cattedrale è un modello tipico delle aziende commerciali. Queste normalmente non rilasciano il codice sorgente e una nuova revisione del programma può richiedere anni mentre il modello a Bazaar è un modello che si è molto diffuso nell'ambiente del software libero dato che consente a ogni utente di essere un beta tester dei programmi. Lo stesso utente può modificare e integrare il programma se lo desidera. Questo consente un rapporto stretto tra utilizzatori e programmatori, un rapporto paritario che ben si adatta alla filosofia del software libero. Consente inoltre un attento controllo del codice in modo da eliminare rapidamente la maggior parte dei bug, cosa impossibile per un software prodotto con la modalità a Cattedrale dove solo un numero limitato di persone lavora sul codice.

La modalità a cattedrale è la stessa metodologia di sviluppo che viene utilizzata dagli sviluppatori delle enciclopedie commerciali dove un numero limitato di esperti si preoccupa di compilare tutte le voci. La modalità a bazaar invece è quella utilizzata dalla Wikipedia dove ogni lettore se lo desidera può integrare e migliorare i contenuti e dove la verifica delle modifiche apportate al testo è gestita dagli stessi utenti.

Il progetto Parancoe corrisponde al classico prodotto open source descritto dal saggio “La cattedrale e il bazar” come modello a bazaar:

•Ogni buon lavoro software inizia dalla frenesia personale di uno sviluppatore

Da numero progetti abbiamo imparato che “la necessità è la madre di tutte le invenzioni”. Infatti Parancoe nasce come risposta ad un'esigenza personale: I membri del progetto Parancoe sono persone che necessitavano di uno stack di sviluppo consolidato e produttivo per la realizzazione di applicazioni Web. Per esempio non era presente fino a quel momento un modello che stabilisse una buona architettura per l'applicazione Web. Ogni volta si tendeva a ripartire da zero riaffrontando, molte volte e con strumenti diversi, gli stessi problemi.

•I bravi programmatori sanno cosa scrivere. I migliori sanno cosa riscrivere (e riusare)

Al progetto Parancoe possono partecipare tutti i programmatori, anche chi non si considera tra i più bravi. La caratteristica di costoro è una sorta di ozio costruttivo: sanno

che si ottiene il meglio non per le energie impiegate ma per il risultato raggiunto, e che quasi sempre è più facile iniziare da una buona soluzione parziale piuttosto che dal nulla assoluto. Linus Torvalds, per esempio, non ha mai cercato di riscrivere Linux da zero. È invece partito riutilizzando codici e idee riprese da Minix. Nello stesso modo in Parancoe non si è deciso di partire da zero ma sono stati integrati più strumenti software ben sviluppati come Spring e Hibernate.

•Preparati a buttarne via uno; dovrai farlo comunque

Spesso non si riesce a comprendere davvero un problema fino alla prima volta in cui si prova a implementarne la soluzione. La seconda volta forse se ne sa abbastanza per riuscirci. Eric Raymond nel suo trattato dice: “Per arrivare alla soluzione, preparati a ricominciare almeno una volta”: Parancoe attualmente è alla versione 2. Dopo la prima versione è maturato insieme ai suoi sviluppatori che hanno prodotto la nuova versione.

•Trattare gli utenti come co-sviluppatori è la strada migliore per ottenere rapidi miglioramenti del codice e debugging efficace

La scelta di rendere Parancoe da subito un progetto aperto a tutti ha contribuito notevolmente ad indirizzarlo verso il mondo Open Source. La distribuzione con sorgente visibile ha reso un miglioramento veloce di Parancoe grazie al contributo di tutti gli utilizzatori che hanno avuto modo di toccare, personalizzare, correggere la libreria che utilizzano e che altri utilizzeranno dopo di loro. Il risultato è una libreria stabile e aggiornata con frequenza. L'apertura del codice ha reso possibile anche la condivisione di esperienza e competenze di molti professionisti che, nel tempo libero e senza alcun tipo di compenso, collaborano allo sviluppo di Parancoe semplicemente perché serve a semplificare il loro lavoro e perché hanno la passione di imparare dagli altri sviluppatori.

•Distribuisci presto. Distribuisci spesso. E presta ascolto agli utenti

Il project leader di Parancoe infatti ha creato un User Group dove gli utenti possono chiedere aiuto. La disponibilità degli sviluppatori a rispondere è molto alta, forse perché sono consapevoli che in questo modo gli utenti sono più stimolati ad avvicinarsi al prodotto, conoscerlo e magari arrivare al punto d'aver l'interesse a contribuire allo sviluppo.

5. Licenza Apache 2.0 in Parancoe

Parancoe è stato licenziato con licenza Apache 2.0 per permetterne l'utilizzo della libreria nella maniera più libera possibile. Il che porterebbe ad avere meno scrupoli su come usarla e di conseguenza si ha un più veloce diffusione, che è l'obiettivo che vuole ottenere il project leader.

La licenza Apache è nata proprio a questo scopo. Infatti permette di scaricare e utilizzare il software Apache, in tutto o in parte, per scopi personali, aziendali o commerciali.

Consente l'uso di software Apache in packages o in qualsiasi distribuzione.

La licenza richiede di includere una copia della licenza in qualsiasi ridistribuzione e permette di fornire una chiara attribuzione a Apache Software Foundation per eventuali distribuzioni che includono il software Apache.

La licenza non richiede di includere il sorgente del software Apache stesso, o di qualsiasi

modifica potrebbe essere fatta ad esso, in qualsiasi ridistribuzione che lo include.

è possibile quindi apportare miglioramenti al codice Apache e ridistribuire il risultato ma, naturalmente, deve sempre essere soggetto ai termini della licenza Apache. Si può ridistribuire il codice modificato gratis, o venderlo, o tenerlo per se. Basta ricordare che il codice originale è ancora coperto dalla licenza Apache ed è necessario rispettare i suoi termini. Anche se si cambia ogni singola riga del codice Apache che si sta utilizzando, il risultato è ancora basato sul codice di licenza della Fondazione. Si può distribuire il risultato con una licenza diversa, ma è necessario riconoscere l'uso di Apache Foundation's software . Altrimenti sarebbe rubare.

6. Busines Model

Il Business Model è un framework che si occupa di valorizzare gli aspetti economici, sociali e/o altre forme di valore all'interno di un azienda.
Osterwalder sintetizzo le differenze di ogni modello fornendo così un Template utilizzabile dalle aziende per descrivere il proprio Business Model. Il Template è composto di quattro sezioni:

-Infrastruttura

è composto dalle capacità e competenze necessarie per eseguire il Business Model aziendale, dalla rete di alleanze con altre aziende necessaria per continuare il proprio business e dalle strategie atte a perseguire il massimo beneficio per l'azienda e i suoi clienti.

-Offerta

Sono l'insieme di prodotti e i servizi che rappresentano il valore offerto dall'azienda a una certo segmento di mercato.

-Clienti

La sezione comprende non solo la fascia di clienti a cui sono rivolti i prodotti e servizi dell'azienda, ma anche il canale di distribuzione ovvero Il modo in cui l'azienda distribuisce i prodotti e servizi ai clienti. Comprende anche il marketing e le strategie di distribuzione. Molto importanti sono le relazioni col cliente, il rapporto messo in atto dall'azienda con i sui clienti.

-Finanze

Comprende i costi dell'azienda dovuti alla produzioni di beni e servizi(Costi delle materie prime, del personale, ecc...), ed anche tutti i ricavi ottenuti dalla vendita dei prodotti creati.

Esempi di Business Model esistenti:

- **Sell technical support services:** viene venduto il supporto tecnico, non il software.
- **Distribute software:** è il modello seguito da linux, consiste nella distribuzione del

software che comprende: il Linux Kernel, una serie di applicazioni(la maggior parte provengono dalla GNU foundation) e una serie di tool e script, solitamente un packaging system boot e init scripts, system configuration tools e patches da terze parti.

- **Found the projects using donations:** basare il progetto su un sistema a donazione è un modo di ottenere il denaro necessario per supportare lo sviluppo attuale e futuro tipico dei progetti free e open source. E il modello di Business più adottato dalle comunità di volontari, ma si applica bene anche alle grandi comunità commerciali. Il sistema di donazioni è regolamentato da leggi nazionali.
Questo modello è utilizzato dalla Mozilla foundation, che riceve donazioni da privati, ma soprattutto da grandi aziende.
- **Build and sell hardware and software configurations:** ci sono varie possibilità:
    - *Un produttore di software che rivende hardware con il software installato.* Tutto il software che viene distribuito come stand-alone o pre-installing su una soluzione hardware. è possibile che venga separata la vendita dei servizi e del supporto tecnico, o venduto in un unica soluzione hardware, configurazione e software.
    - *Un produttore di hardware che vende hardware con software preconfigurato.* All'inizio il target di mercato erano persone con grande conoscenza specifica, ora il mercato richiede prodotti che siano altamente configurabili. Per questo le grandi aziende hanno cominciato a distribuire prodotti con firmware free e open source preinstallati. Ci sono molti esempi di prodotti che usano questo modello: routers, switches, firewalls, smart phones, PDA, videocamere, TV, riproduttori e registratori audio e video.
- **Release proprietary versions of the software:** quando il software originale viene modificato, perchè sono state aggiunte nuove features o sono stati corretti dei bug e il software risultante viene venduto con licenza proprietaria.
- **Release proprietary add-ons, components**: prevede la vendita di un componente(un plug-in o una estensione) con licenza commerciale, pagando una piccola tarifa per la licenza e/o la sua distribuzione.
- **Dual licensing software:** Il codice sorgente viene distribuito con due licenze. Una licenza free come la GNU GPLv2 e una licenza commerciale. La prima rende il codice libero di essere scaricato, usato, modificato e distribuito. La seconda è ottenibile solo dai clienti che pagano per averlo. I clienti hanno garanzie che gli altri non hanno. Con la licenza commerciale si ha la possibilità di personalizzare, applicare modifiche e estensioni al software senza l'obbligo di rilasciare il codice sorgente.
- **Offer each new version of a software only to paying customers:** un altro modello attuabile è offrire un nuovo prodotto con licenza free e open source per un tempo limitato tramite il pagamento di una tariffa. Ci sono due metodi per questo modello:
    - i clienti possono sottoscrivere un piano col quale possono entrare in possesso del prodotto non appena viene rilasciato.L'adesione può scadere dopo un certo numero di mesi. Dopo questo lasso di tempo il prodotto viene rilasciato a chiunque gratuitamente.
    - I clienti possono acquistare il prodotto non appena viene rilasciato. Lo stesso prodotto verrà distribuito gratuitamente dopo qualche tempo.

Nel caso di parancoe si è scelto di non richiedere il pagamento per l'utilizzo e non è

previsto nessun servizio di supporto. Il software in questione è semplicemente uno strumento che servirà a chi lo ha sviluppato per realizzare in modo più efficiente e efficace applicazioni web richieste dai rispettivi clienti

7. Strumenti di collaborazione

Frederick Phillips Brooks, esperto di ingegneria del software, ha scritto il libro The Mythical Man-Month. In questo libro è descritta famosa legge di Brooks:

“Poiché la costruzione del software è intrinsecamente un lavoro sistemico, ovvero un esercizio di complesse interrelazioni, essa richiede un notevole sforzo comunicativo. Ciò finisce per imporre la rapida diminuzione del tempo di lavoro individuale causato dall’ulteriore suddivisione dello stesso. Aggiungere nuove persone quindi allunga i tempi, invece di accorciarli.”

Fred Brooks osserva che la complessità di un progetto aumenta in modo quadratico con l'aumentare del numero di partecipanti. Quando sono coinvolte solo poche persone, ognuno può facilmente parlare con tutti gli altri, ma quando invece sono coinvolte centinaia di persone, non è più possibile per ogni persona rimanere costantemente consapevole di ciò che stanno facendo tutti gli altri .

Dal momento che quasi tutte le comunicazioni in progetti open source avvengono in forma scritta, si sono evoluti sistemi complessi per il routing e l'etichettamento dei dati in modo appropriato; per ridurre al minimo le ripetizioni, per l'archiviazione e recupero dei dati, per la correzione di cattive o obsolete informazioni.

Generalmente nei progetti open source attivi ci sono soggetti spesso eseguono complesse operazioni manuali al fine di garantire che le informazioni vengano instradate correttamente. Ma l'intero sforzo dipende in ultima analisi, sul sostegno del software.

Gli strumenti standard di collaborazione per la gestione delle informazioni sono:

•sito Web: centralizza il tutto, serve come titolo informativo dal progetto per il pubblico. Il sito web può anche servire come interfaccia amministrativa per gli altri strumenti del progetto. Per il progetto Parancoe è stato creato infatti il sito web http://www.parancoe.org/ come punto di riferimento. Dal sito si puo' accedere al sito web di google code dove è possibile scaricare i sorgenti, all'user group, ai contatti dei membri del progetto, e alla documentazione.

•Mailing list: è il mezzo di comunicazione del progetto più attivo. Una buona gestione delle mailing list non è affatto semplice. Non si tratta solo di annullare l'iscrizione degli utenti al momento della richiesta. è necessaria anche la presenza di un moderatore per evitare lo spam. La mailing list è una necessita' anche per il progetto Parancoe quindi è stato creata la mailing list sfruttando gli strumenti offerti da google. In questo caso, essendo un

progetto con qualche decina di membri, è lo stesso project leader ad essere il moderatore.

8. Processi di sviluppo software

Un processo di sviluppo software è una struttura imposta per lo sviluppo di un prodotto software.

I processi standard consigliati per un buon sviluppo del software sono:

•Estrazione dei requisiti (detta “analisi dei requisiti”): è un compito importante nella creazione di un prodotto software perché generalmente i clienti hanno un'idea astratta di ciò che vogliono come risultato finale, ma non hanno idea di quello che il software dovrebbe fare.

•Specifica: in questo processo si descrive il proprio software in modo rigoroso. Le specifiche sono importanti per le interfacce esterne, che devono rimanere fisse. Un buon modo per determinare se le specifiche sono sufficientemente precise è quello di avere dei verificatori che controllano se i requisiti e i casi d'uso sono logicamente coerenti.

•Architettura software: si riferisce a una rappresentazione astratta del sistema. L'architettura serve a garantire che il software sia conforme ai requisiti del prodotto, e anche per assicurare che le esigenze future possono essere affrontate. Nell'architettura vengono anche descritte le interfacce tra il sistema di software e altri prodotti software, l'hardware sottostante o il sistema operativo.

•Implementazione: è la parte del processo in cui viene scritto il codice del programma.

•Testing: è parte integrante del processo di implementazione ed è uno dei piu' importanti processi di sviluppo. Questa parte del processo assicura che i bug siano riconosciuti.

•Documentazione: serve per la futura manutenzione e il miglioramento. La documentazione viene redatta durante tutto lo sviluppo.

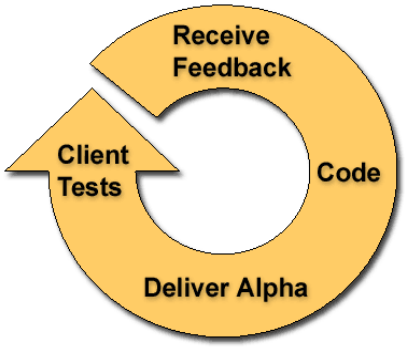
•Deployment: dopo che il codice è stato adeguatamente testato, è approvato per il rilascio e venduto o altrimenti distribuito

•Supporto: è un processo importante perché una grande percentuale di progetti software fallisce perché le persone sono spesso restie a migrare ad un nuovo software sconosciuto. Nella fase di distribuzione, è molto importante tenere corsi di formazione per i nuovi clienti del software e distribuire documentazione sull'uso del programma.

•Manutenzione: serve a far fronte ai nuovi problemi o a nuove esigenze. La manutenzione del software può richiedere molto più tempo dello sviluppo iniziale. Può essere necessario modificare codice che non va bene per il design originale, come per correggere un problema imprevisto, o può essere che un cliente abbia richiesto più funzionalità e nuovo il codice deve essere aggiunto per soddisfare le nuove richieste.

•Sistemi di Bug Tracking: vengono usati strumenti per consentire ai team di sviluppo di interfacciarsi con il cliente o gruppi di test per identificare eventuali problemi. Questi strumenti software forniscono un processo personalizzabile per acquisire, revisionare, riconoscere e rispondere ai problemi segnalati.

Successivamente ai processi standard (definiti anche “metodologie pesanti” per i vecchi metodi come il modello a cascata, metodologie iterative per i metodi come il modello a spirale), sono nate le metodologie agili di sviluppo software. Nell'ingegneria del software, per metodologia agile (o leggera) o metodo agile si intende un particolare metodo per lo sviluppo del software che coinvolge quanto più possibile il committente, ottenendo in tal modo una elevata reattività alle sue richieste. La gran parte dei metodi agili tentano di ridurre il rischio di fallimento sviluppando il software in finestre di tempo limitate chiamate iterazioni che, in genere, durano qualche settimana. Ogni iterazione è un piccolo progetto a sé stante e deve contenere tutto ciò che è necessario per rilasciare un piccolo incremento nelle funzionalità del software: pianificazione, analisi dei requisiti, progetto, implementazione, test e documentazione.



Agile development as implemented by The Smart Method
This cycle normally completes every development day!

Anche se il risultato di ogni singola iterazione non ha sufficienti funzionalità da essere considerato completo deve essere rilasciato e, nel susseguirsi delle iterazioni, deve avvicinarsi sempre di più alle richieste del cliente. Alla fine di ogni iterazione il team deve rivalutare le priorità di progetto.

I metodi agili preferiscono la comunicazione in tempo reale, preferibilmente faccia a faccia, a quella scritta (documentazione). Il team agile è composto da tutte le persone necessarie per terminare il progetto software. Come minimo il team deve includere i programmatori ed i loro clienti.

è da sottolineare che i clienti di Parancoe sono gli sviluppatori stessi e questo a portato a seguire più uno sviluppo a metodologia agile che non a seguire il vecchio iter standard.

L'obiettivo della metodologia agile è la piena soddisfazione del cliente e non solo l'adempimento di un contratto. Gli sviluppatori di Parancoe avevano il più alto interesse a soddisfare tutti i requisiti perché sarebbe stato solo a loro vantaggio. L'uso di queste metodologie, inoltre, serve ad abbattere i costi di sviluppo del software.

I principi su cui si basa una metodologia leggera che segua i punti indicati dall'Agile Manifesto, sono solo quattro:

- le persone e le interazioni sono più importanti dei processi e degli strumenti (ossia le relazioni e la comunicazione tra gli attori di un progetto software sono la miglior risorsa del progetto);
- è più importante avere software funzionante che documentazione (bisogna rilasciare nuove versioni del software ad intervalli frequenti, e bisogna mantenere il codice semplice e avanzato tecnicamente, riducendo la documentazione al minimo indispensabile);
- bisogna collaborare con i clienti al di là del contratto (la collaborazione diretta offre risultati migliori dei rapporti contrattuali);
- bisogna essere pronti a rispondere ai cambiamenti più che aderire al progetto (quindi il team di sviluppo dovrebbe essere autorizzato a suggerire modifiche al

progetto in ogni momento).

Si può capire perché i progetti open source, dove la comunicazione tra persone è valorizzata e molto sostenuta, tendano ad adottare una metodologia agile.

Parancoe utilizza il metodo TDD(Test-Driven Development), metodologia *agile* che appartiene alla tecnica dell'Extreme Programming.
E' un'alternativa al solito metodo a cascata (*waterfall*), secondo il quale, dopo un'adeguata analisi e progettazione, si inizia a scrivere il codice e solo alla fine si passa al testing e al debug.
Il TDD, invece, (previa analisi e progettazione, ovviamente) consiste nello scrivere il proprio codice ripetendo ciclicamente questi passi:

•**scrivo il test prima ancora che esista il codice da testare:** Nel test-driven development, ogni nuova funzione inizia con la scrittura di un test. Questa prova deve inevitabilmente fallire perché è scritto prima che la funzione sia stata implementata. (Se non fallisce, allora la proposta della "nuova" funzionalità è eliminata) Per scrivere un test, lo sviluppatore deve comprendere chiaramente la funzione di specifiche e le esigenze. Lo sviluppatore può compiere questo attraverso i casi d'uso e le storie che coprano i requisiti e le condizioni di eccezione. Ciò può anche implicare una variante, o la modifica di un test esistente. Questa è una caratteristica di differenziazione dei test-driven development rispetto lo scrivere unit test dopo che il codice è scritto: focalizza l'attenzione degli sviluppatori sui requisiti prima di scrivere il codice, una differenza sottile ma importante.

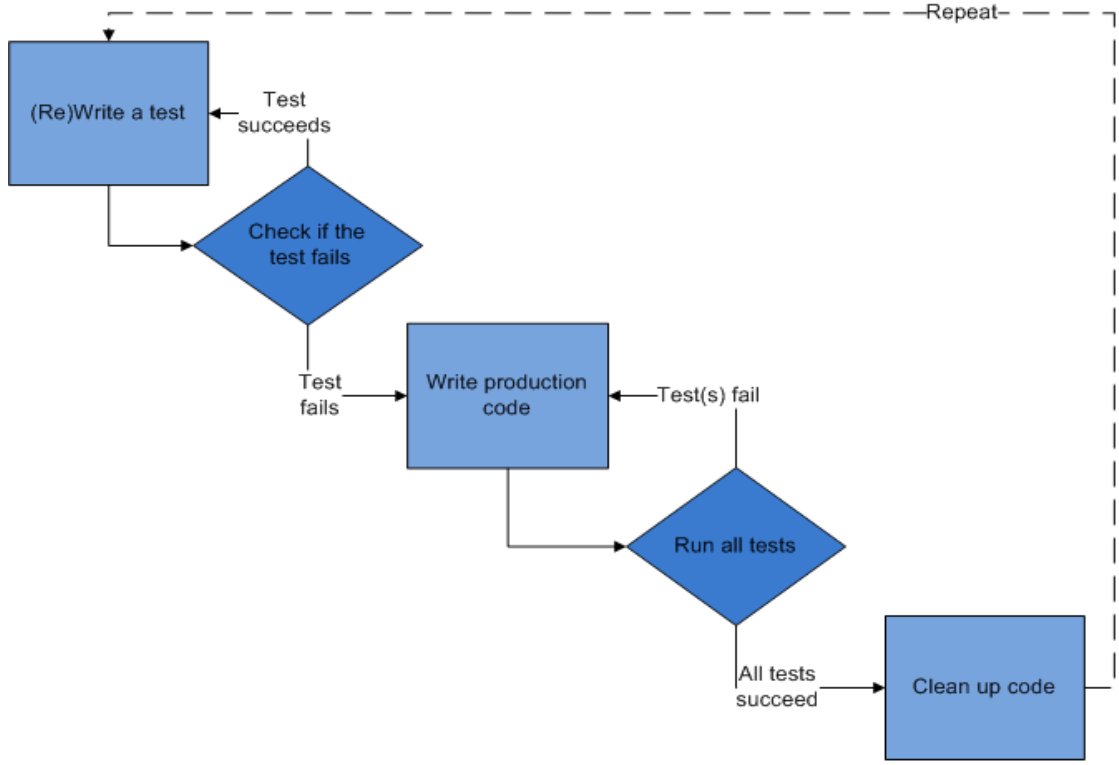
•**il test (ovviamente) fallisce:** Il nuovo test provoca il fallimento del sistema di test in quanto la nuova funzionalità che il test cerca di verificare non è ancora stata implementata. Se il test non dovesse fallire significa che la nuova funzionalità che si vuole implementare è già presente e quindi non è necessario spendere altro tempo su questa funzionalità.
.
Questa fiducia aumenta (anche se non ne da la garanzia) che si sta testando la cosa giusta, e passerà solo nei casi previsti.

•**scrivo il codice minimo che faccia passare il test:** Il nuovo codice scritto in questa fase non sarà perfetto, e può, per esempio, superare il test in un modo poco elegante. Questo è accettabile in quanto ci si occuperà in fasi successive a migliorare e perfezionare il codice.

•**il test passa:** Se il passa test, il programmatore può essere certo che il codice soddisfi tutte le condizioni testate. Questo è un buon punto da cui iniziare la fase finale del ciclo.

•**eseguo il refactoring del codice:** Ora il codice può essere ripulito, se necessario. In questa fase lo sviluppatore può sistemare il codice precedentemente scritto controllano che le modifiche fatte non cambino l'esito dei test.

Per ogni nuova funzionalità da aggiungere va seguita questa procedura.



Test-driven development offre più di una semplice convalida di correttezza, può anche guidare la progettazione di un programma. Focalizzando l'attenzione sui prima sui test, si deve immaginare come la funzionalità saranno utilizzati dai clienti (nel primo caso, i casi di test). Quindi, il programmatore si preoccupa solo con l'interfaccia e non la realizzazione.
Esso consente a un programmatore di concentrarsi sul compito a portata di mano, casi eccezionali e la gestione degli errori non sono considerati inizialmente. Prove per creare queste circostanze estranee sono implementati separatamente. Un altro vantaggio è che il test-driven development, se usato correttamente, garantisce che tutto il codice scritto è coperto da una prova. Questo può dare al team di programmazione, e gli utenti successivi, un maggiore livello di fiducia nel codice.
TDD può portare a rendere il codice più modulare, flessibile, e estensibile. Questo effetto avviene spesso in quanto il metodo richiede che gli sviluppatori pensino al software in termini di unità di piccole dimensioni che possono essere scritti e testati in modo indipendente e integrato insieme più tardi. Questo porta a più piccoli, le classi più mirata, accoppiamento lasco, e le interfacce più pulite.

9. Conclusione

Analizzando Parancoe ci siamo quindi reso conto che nel suo piccolo possiede tutte le caratteristiche che ha generalmente un buon progetto open source. L'importanza che gli

sviluppatori e il project leader hanno dato alla qualità della gestione del progetto ha dato origine ad un progetto open source che avrà la possibilità di evolvere in futuro nello stesso interesse dei membri che lo hanno fondato e che lo dovranno usare per il loro lavoro.

10. Ringraziamenti

Ringraziamo Lucio Benfante, project leader del progetto Parancoe, per la collaborazione e per per la sua grande disponibilità che ha sempre dimostrato nel rispondere alle nostre domande, come da buon project leader di un progetto open source.