Gli strumenti CASE per l'ingegneria del software

L'ingegneria del software, la materia che si occupa di descrivere e migliorare il processo di produzione di nuovo software, descrive anche un'intera classe di software di supporto per assistere il processo di produzione e di sviluppo del software stesso

di Diego Pettenò

Parliamo dei cosiddetti strumenti CASE (Computer Aided Software Engineering, ingegneria del software assistita dal computer).

Il contributo dato da questa classe di software alla qualità dei prodotti finali è spesso sottovalutato, poiché in passato lo sviluppo di questi strumenti era poco pratico o poco conveniente e il loro impiego non veniva considerato vantaggioso per diverse ragioni, a cominciare dal costo dei software disponibili (proporzionale alla difficoltà del loro sviluppo), e finendo con la quantità di spazio di immagazzinamento su memorie di massa necessario per memorizzare le informazioni.

Lo stesso sviluppo di tali strumenti era spesso troppo arduo, un singolo gruppo di sviluppatori non sarebbe mai riuscito a creare software CASE allo stesso tempo generico e dettagliato per adattarsi alle differenti necessità di altri gruppi di programmatori, sia perché aumentando la flessibilità di un programma spesso si finisce col renderlo troppo complicato per essere semplice da utilizzare, sia perché è necessaria la conoscenza dei diversi modi in cui il programma verrà utilizzato per sapere come scriverlo.

Ma come spesso accade nel mondo del software, bastano pochi anni perché un'idea che fino a poco tempo prima sarebbe stata considerata irrealizzabile diventi abbordabile grazie ai cambiamenti che avvengono sia nel modo di sviluppare il software sia nel campo dell'hardware medio su cui il software sarà eseguito. Per questo motivo ad oggi ci sono diversi strumenti CASE che vengono diffusamente utilizzati nei processi di sviluppo del software, sia proprietario che libero, sia professionistico sia amatoriale.

Il contributo del software libero

Una delle ragioni primarie nell'espansione del software CASE è sicuramente legata al software libero. Per sua stessa natura, il software libero tende ad essere sviluppato da persone con storie personali diverse, provenienti non solo da nazioni e culture diverse (di cui si possono interessare i sociologi, ma non di sicuro sviluppatori), ma soprattutto da scuole di sviluppo e progettazione diverse. La maggior parte degli sviluppatori impara un metodo di sviluppo e di progettazione ricavandolo dai propri insegnanti, su carta o di fatto (uno sviluppatore che ha imparato a programmare sul campo da un programmatore più anziano avrà sicuramente un metodo di lavoro diverso rispetto ad uno studente uscito dall'università che ha imparato a programmare attraverso i suoi professori: il primo avrà un approccio più orientato alla funzionalità degli strumenti con cui sviluppa, mentre lo studente avrà un approccio più formale, considerando che la maggior parte dei professori utilizzano tale approccio, che quindi valuterà gli strumenti inizialmente per quello che gli viene detto vengono utilizzati).

A causa di queste differenze, è spesso necessario uniformare il metodo di lavoro all'interno di un progetto di software libero, e questo avviene solitamente tramite l'utilizzo di strumenti CASE che aiutino a standardizzare il processo d'insieme.

Un altro contributo dato dal software libero allo sviluppo degli strumenti CASE è relativo direttamente al lato tecnico dello sviluppo del software libero stesso. Come si è dichiarato in precedenza, difficilmente un gruppo limitato di sviluppatori riuscirebbe a sviluppare un software che si applichi a casi molto generali: lo sviluppo di un tale software richiederebbe molti mesi di analisi, interviste e prove pratiche per poter raccogliere dati sul metodo di utilizzo dell'applicativo, ma questo solitamente è un costo troppo alto per un'azienda che volesse vendere il proprio prodotto, poiché ai requisiti di risorse non corrisponde una certezza di introito, il software sviluppato potrebbe molto facilmente essere troppo generico per essere utilizzato in modo pratico, oppure si potrebbe finire con il crearlo troppo legato al processo di produzione delle grandi aziende, troppo complesso e intricato per i medi-piccoli sviluppatori che formerebbero il maggiore pubblico per un tale software; una soluzione è quella di sviluppare un software generico da modificare per il cliente di volta in volta, ma questo potrebbe essere troppo oneroso per i piccoli sviluppatori, e coloro che potessero permettersi una tale spesa potrebbero, molto probabilmente, sviluppare il software di supporto in casa.

Grazie agli ideali del software libero, è più semplice iniziare a sviluppare un software tagliato per un determinato scopo, sviluppato appositamente per un gruppo di sviluppatori, e aspettarsi che altri gruppi di programmatori estendano,

astraggano o dettaglino tale software, riducendo la necessità di analisi e interviste. Inoltre lo sviluppo di soluzioni su misura diviene più pratico, poiché non è più necessario chiederne la modifica ai creatori originali, ma chiunque può modificare il software per come è necessario, dedicando un numero limitato di sviluppatori interni che già conoscono il flusso del processo nel proprio ambito.

Ma seppure il software libero ha una primaria importanza nella diffusione degli strumenti CASE, e difatti molti di tali strumenti sono disponibili come software libero, non si tratta sicuramente della sola ragione per cui questi strumenti sono ora più diffusi che in passato.

Molti strumenti CASE memorizzano informazioni durante tutte le fasi di sviluppo di un programma, quindi la quantità di spazio su memorie di massa di cui necessitano cresce con la crescita delle dimensioni del software (e di fatto un software solitamente cresce di dimensioni durante la sua vita, e mai diminuisce). Anche solo cinque anni fa era un sogno quasi impossibile avere un computer con due dischi rigidi da 240 GiB, mentre ora si tratta di una realtà molto comune; anche se i costi dei dischi rigidi professionali per server e workstation aziendali non sono scesi con la stessa curva di quelli per normali PC, si ha comunque un prezzo per MiB ben più basso del passato, che rende possibile l'immagazzinamento delle grosse quantità di dati generati dagli strumenti CASE.

Progettare, realizzare, mantenere

Esistono strumenti CASE per assistere a tutte le fasi del processo di produzione del software: alcuni sono pensati e realizzati per assistere i progettisti che lavorino prima dell'inizio dello sviluppo vero e proprio del software, altri sono pensati per aiutare la gestione del software dal momento in cui si inizia la sua programmazione per tutta la sua estensione di vita, mentre altri assistono il mantenimento sul medio e lungo termine.

Ci sono poi molti software che non si considerano spesso come strumenti CASE, come i compilatori, perché sono parte integrante del processo di sviluppo del software, o perché sono stati sviluppati prima che la definizione di CASE fosse coniata. Si tratta perlopiù di "assistenti" allo sviluppo, come estensioni o miglioramenti del compilatore o strumenti di debug. La loro presenza nella categoria degli strumenti CASE può essere messa in discussione, ma non ci sono dubbi che, facendo parte del processo di sviluppo del software, siano anche parte della più grande catena dell'ingegneria del software.

Ovviamente ci sono casi in cui, per supportare la produzione di un particolare software, si vanno ad utilizzare strumenti che non sono stati pensati e sviluppati per tale scopo. Anche se questi fanno parte di determinati processi di sviluppo, di un'azienda o di un gruppo di programmatori, non è ovviamente possibile considerarli strumenti CASE, nello stesso modo in cui un browser Web utilizzato per la ricerca di informazioni non possa essere considerato strumento di un'inchiesta giornalistica.

C'è poi da dire che la maggior parte di questi strumenti non ha un risultato ultimo, perché quello di cui si occupano è fornire un supporto durante tutto il processo di sviluppo. Quando gli strumenti generano informazioni direttamente, si tratta solitamente di informazioni estemporanee che vengono utilizzate e rese già obsolete dagli sviluppatori. Gli strumenti più importanti, però, creano soltanto una grossa quantità di informazioni che continuano a venire utilizzate da quegli strumenti stessi, e da cui è comunque possibile recuperare dati più intelligibili (seppur spesso con dubbia utilità) che possono essere utilizzati per creare statistiche e grafici utili nelle presentazioni.

Supporto alla progettazione

A seconda del metodo di sviluppo che si sta seguendo, la progettazione iniziale, prima di iniziare lo sviluppo di un software, ha una minore o maggiore importanza; in ogni caso, anche nei metodi di sviluppo agile e di programmazione estrema (XP, eXtreme Programming), in cui i programmatori si concentrano soprattutto sulle fasi di sviluppo e di test incrementale del software, una delle prime fasi del processo di sviluppo del software è appunto la progettazione (preceduta dalla raccolta di informazioni riguardo al software che si sta sviluppando).

Per riuscire a scrivere strumenti di supporto funzionali, utili e comprensibili da un vasto pubblico per la progettazione del software, è stato definito un linguaggio che può essere utilizzato per descrivere la struttura di un programma secondo una serie di aspetti, si tratta di UML (Unified Modelling Language, linguaggio di modellazione unificato). UML descrive come si possono realizzare una serie di diagrammi che a loro volta descrivono il software che si sta progettando. I diagrammi più comuni sono sicuramente quello delle classi (che descrive i legami di ereditarietà tra le classi e i loro membri), quello di attività (una versione modificata del classico diagramma di flusso (flow chart) pensata per descrivere il flusso delle funzioni di un programma) e quello dei casi d'uso (use-case) (che permette di definire il modo in cui diverse persone utilizzano un dato software).

Per quanto sia logicamente possibile realizzare i diagrammi UML con un foglio e una matita, o con un generico software di grafica, uno strumento CASE offre più di queste semplici funzionalità.

I software di progettazione UML, come Umbrello e bouml (come software libero) o Poseidon (proprietario). Questi programmi non sono solo programmi di grafica specializzati, ma forniscono supporto aggiuntivo, come la possibilità di definire una singola classe una sola volta e riutilizzarla nei vari diagrammi, delle classi e di struttura, correlandole a livello logico e grafico. Oltre a questo, quasi tutti i software di progettazione UML hanno una funzione, più o meno elaborata, per la generazione di codice a partire dai diagrammi disegnati; si tratta solitamente solo di uno schele-

tro delle classi definite nello schema, con i nomi e le firme delle funzioni, senza la loro definizione.

Per quanto nessun software è ancora abbastanza sofisticato dal riuscire a generare del codice che funzioni senza ritocchi qua e là, poiché i diagrammi UML sono solo delle descrizioni della struttura, dell'uso o del flusso dei dati, ma per esempio gli header con le dichiarazioni delle classi derivate dal diagramma relativo sono un buon punto di partenza per la loro implementazione, e un riferimento per gli utilizzatori di tali classi che possono iniziare lo sviluppo di altri componenti del software in parallelo, senza dover attendere la completa realizzazione delle loro dipendenze.

Supporto allo sviluppo

Quasi nessun software viene scritto immediatamente nella sua forma ultima, ma tende invece ad evolvere a partire da uno scheletro (o dal nulla) fino a diventare pronto per il rilascio. Molto spesso, poi, il codice sorgente del software rilasciato non rimane fermo, ma si evolve anche dopo il rilascio per la creazione di versioni successive, per la correzione di bug e così via.

Per questa ragione con il termine "sviluppo" non ci si vuole solo riferire alla fase di lavorazione che si pone tra la progettazione e la convalida, ma a tutte quelle fasi che partono nel momento in cui si inizia a lavorare al codice sorgente, comprese quelle necessarie per la correzione degli errori individuati nella fase di test e quelle necessarie per l'aggiornamento del software per adattarsi alle richieste dei committenti durante tutto l'arco di vita del suo supporto. È quindi di primaria importanza conoscere quali modifiche sono state effettuate nel tempo, il loro motivo e il loro autore.

Per questa ragione, uno dei più importanti strumenti di ingegneria assistita è sicuramente il software per la gestione del codice sorgente (SCM, Source Code Management), ovvero quel software che permette di salvare e visionare la storia di un insieme di file contenenti codice sorgente (o altro) memorizzando tutte le modifiche che avvengono tra una versione e l'altra dei file. Un simile risultato si ottiene utilizzando un metodo non-CASE, ovvero il backup periodico dei sorgenti (giornaliero, orario, o nel momento del rilascio del software), ma tramite l'uso di un SCM è solitamente possibile memorizzare assieme alle varie modifiche anche un commento relativo alle modifiche effettuate, e a seconda di quanto sofisticato sia il software SCM utilizzato è possibile riprendere i sorgenti come stavano ad un certo punto del tempo e iniziare uno sviluppo parallelo a partire da questi.

Il software SCM libero più conosciuto ed utilizzato è sicuramente CVS (Concurrent Versioning System, sistema di versionamento concorrente), basato sul più datato RCS (Revision Control System, sistema di controllo delle revisioni), che espande con il supporto per la gestione contemporanea di più file, l'accesso da parte di utenti multipli e tramite rete. CVS è stato sicuramente uno dei punti chiave nello sviluppo comunitario del software libero; grazie alla sua disponibilità e relativa semplicità di impostazione, è uno dei più software più utilizzati per la gestione dei sorgenti di software libero e Open Source, specialmente perché negli anni sono nati e fioriti diversi servizi che forniscono agli autori di tale software spazio web e altri strumenti per la gestione dei progetti, tra cui anche SCM (SourceForge.net, BerliOS, GNU Savannah, . . .). Purtroppo l'età di CVS si sta dimostrando un grosso punto a sfavore, mancando molte delle funzioni disponibili in altri software, come Subversion, tra cui la gestione dei cambiamenti su file diversi come una singola modifica.

Esiste poi una serie di software SCM "distribuiti", che rimuovono la necessità di un singolo server a cui accedere per inviare le modifiche, e invece le spostano sui diversi computer su cui queste avvengono (e, volendo, su un server pseudo-centralizzato).

Anche se tutti (o quasi) gli SCM permettono di visualizzare le modifiche tra una versione e l'altra e recuperare le informazioni relative tramite il software stesso, solitamente tali informazioni vengono recuperate e visualizzate tramite più amichevoli interfacce web, come ViewVC, WebCVS, WebSVN, gitweb e così via. Questi software sono strettamente legati e dipendenti dal software SCM stesso, anche se sono solitamente sviluppati separatamente e da gruppi separati di sviluppatori.

Durante lo sviluppo, poi, è solitamente necessario avere a disposizione la documentazione delle interfacce interne ed esterne del software, (quelle esterne sono necessarie anche una volta che il software è stato completato), se esistono. Per quanto sia possibile scrivere tale documentazione separatamente, per facilitare questo compito sono stati sviluppati diversi software che interpretano i sorgenti ed estraggono i commenti lì presenti per generare una documentazione di riferimento, se formattati in modo opportuno. Solitamente questi software generano la documentazione in formati diversi, HTML tra questi in genere, ma possono anche generarla in un formato utilizzabile da un altro programma per fornire informazioni contestuali nel codice sorgente.

Purtroppo realizzare un software unico che possa utilizzarsi per documentare qualsiasi linguaggio è molto difficile, non solo perché non esiste un interprete universale che si adatti a tutte le regole di sintassi dei vari linguaggi, ma anche perché ogni linguaggio ha caratteristiche diverse (sintassi di chiamata, strutture dati, . . .) che devono essere tenute in considerazione nella creazione della documentazione. Per questa ragione esistono diversi software che generano la documentazione a partire da sorgenti di diversi linguaggi.

Tra i software liberi disponibili è da segnalare Doxygen, che utilizza una sintassi compatibile a quella di JavaDoc e al software proprietario utilizzato per documentare le librerie Qt di Trolltech, limitato a linguaggi più o meno compatibili

con C, C++ e Java, che ha la possibilità di generare diagrammi delle classi UML basandosi sul codice effettivamente scritto, permettendo un rapido confronto tra la progettazione e la realizzazione effettiva.

Oltre a questi software di supporto strettamente CASE, ci sono altri strumenti che si possono considerare parte dei processi CASE anche se non vengono utilizzati solamente nel contesto dell'ingegneria del software, e anzi sono, per gli sviluppatori che li utilizzano, parte stessa dello sviluppo.

Si tratta di quegli strumenti utilizzati per il debug, ovvero per la ricerca e correzione degli errori di programmazione all'interno di un prodotto in via di sviluppo.

La fase di debug non può essere saltata da nessuno sviluppatore, professionista o amatoriale, singolo o in gruppo, di software libero o proprietario, qualsiasi sia il metodo di sviluppo utilizzato.

Si tratta di parte integrante dello sviluppo del software e per questo è sicuramente l'attività per la quale sono stati scritti più software di supporto.

Il primo software importante per questa attività è il cosiddetto "debugger dinamico", utilizzato per verificare lo stato di un processo in esecuzione determinato sul codice sorgente. Questo tipo di strumento è altamente dipendente dal linguaggio di programmazione utilizzato, dal compilatore o interprete scelto, e dal sistema operativo. Per quanto le funzioni principali siano molto chiare (fermare l'esecuzione, analizzare le variabili, eseguire passo-passo), con il tempo i vari debugger hanno iniziato a fornire funzioni avanzate, come interfacce grafiche o l'accesso remoto tramite seriale o TCP/IP.

Il software libero più utilizzato per questo compito è sicuramente GNU Debugger (gdb).

Oltre a questo, esistono due principali classi di strumenti che sono pensati per supportare tale attività, si tratta degli strumenti per l'analisi statica e l'analisi dinamica.

I primi sono utilizzati per analizzare i sorgenti del software durante lo sviluppo, alla ricerca di chiamate errate o problematiche, per evitare errori comuni, mentre gli strumenti di analisi dinamica sono utilizzati sul software in esecuzione per verificarne il comportamento, per cercare eventuali errori nella gestione della memoria o nell'utilizzo dei parametri forniti.

Mentre l'analisi statica viene eseguita sull'intero codice, l'analisi dinamica può essere eseguita solo sulle parti di codice che sono realmente eseguite mentre lo strumento è attivo, quindi per quanto solitamente più sofisticata, non può considerarsi un metodo sicuro per la verifica di un intero software.

Bisogna però mettere le mani avanti quando si parla di questi strumenti, facendo una netta distinzione tra i linguaggi compilati e quelli interpretati, ovvero tra i linguaggi come C, C++ e Java (per quanto quest'ultimo sia parzialmente interpretato) e i linguaggi di scripting come Perl, Python, Ruby e PHP, utilizzati professionalmente soprattutto nella creazione di Web Applications (Perl, PHP e Ruby in particolare).
Mentre per i primi sono disponibili molti software di analisi sia statica che dinamica, per gli ultimi la cosa si fa molto più complessa, poiché non è solitamente possibile effettuare un'analisi statica di uno script interpretato, perché le variabili possono assumere valori e tipi di dato che cambiano al momento dell'esecuzione, e per l'analisi dinamica è difficile separare l'esecuzione degli script dall'esecuzione dell'interprete e della chiamata alle funzioni di libreria necessarie.

L'analisi dell'esecuzione dei software scritti utilizzando linguaggi interpretati, quindi, deve essere effettuata dall'interno del programma stesso, solitamente senza software che la possa assistere.

Gli strumenti di analisi statica hanno avuto un ampio sviluppo dagli albori della programmazione moderna fino ad oggi, il software più conosciuto è sicuramente LINT, sviluppato appunto negli anni '70 quando i compilatori C non effettuavano un rigido controllo sul tipo dei parametri passati alle funzioni, permettendo dunque di chiamare le funzioni di libreria con parametri diversi da quelli attesi (interi per stringhe e viceversa per esempio).

Si tratta però di un software ormai obsoleto, poiché tutti i compilatori moderni già fanno tale controllo, e buona parte dei controlli statici che venivano effettuati da LINT. Il compilatore diventa dunque anche un importante strumento di ingegneria del software che, correttamente utilizzato, permette di migliorare la qualità del software sviluppato.

Mentre il software di analisi statica per un dato linguaggio di programmazione sovente funziona su qualsiasi sistema operativo, salvo i normali problemi di portabilità del software, il software di analisi dinamica dipende anche molto dal sistema operativo (e dall'architettura hardware), si tratta quindi di software particolare, che nel software libero è rappresentato da Valgrind (per GNU/Linux) e da DTrace (per Solaris e FreeBSD). Ma anche le librerie di sistema moderne hanno alcune funzioni di analisi che possono indicare se il programma sta comportandosi in modo corretto con la memoria.

Supporto al test e alla convalida

La fase di test e di convalida è spesso sottovalutata sia da programmatori amatoriali sia da professionisti; si tratta della fase in cui si va a controllare che il software sviluppato sia conforme ai desideri del cliente (o in generale degli utenti del software stesso). L'uso di software CASE per questa fase ha un'importanza maggiore che per le altre: non è solo un ausilio che semplifica il lavoro degli sviluppatori, ma può anche essere l'unico modo per avere dei risultati affidabili riguardo i test: se alcuni test fondamentali sono sempre effettuati manualmente non è impossibile né raro che il test sia effettuato in modo leggermente diverso di volta in volta, rendendo i risultati inattendibili o addirittura inutili. Per questa ragione è importante che i controlli siano automatizzati il più possibile, e per questo sono stati sviluppati diversi software CASE per migliorare l'affidabilità dei controlli. Si deve però sapere che più un controllo è complesso e avanzato, più è

probabile che individui dei falsi positivi per codice scritto in modo poco ortodosso o comunque non consono.

Un esempio di software CASE per il supporto della fase di test si trova in quelle soluzioni per la creazione dei cosiddetti (test unit, unità di test), come per esempio JUnit e C++Unit, che permettono di semplificare la scrittura di test in modo che possano eseguirsi in modo analogo anche se vanno a testare parti differenti del software. Le unità di test poi possono comporsi in "batterie di test di regressione", che se eseguiti regolarmente dopo modifiche importanti ed estensive permettono di riconoscere subito errori nelle modifiche che possono essere più difficili da individuare successivamente.

Ma se i test di regressione sono più parte dell'attività di debug che della fase di convalida, si possono scrivere unità di test più complesse che verificano il comportamento delle interfacce esterne del software, piuttosto che singole funzioni o singoli componenti interni del software. In questo caso si tratta di un valido strumento CASE per la convalida, anche se molto spesso le funzionalità da convalidare richiedono l'intervento di un utente reale, specie per i software con interfaccia grafica complessa.

Supporto alla manutenzione

C'è un'altra classe di strumenti CASE che deve molto, per il suo sviluppo, al software libero: si tratta dei cosiddetti bug trackers, quei software, utilizzati sia da progetti di software libero che da aziende sviluppanti software di qualsiasi tipo (ma anche da alcune aziende che vendono servizi), che permettono di segnalare errori e richieste di funzionalità per un determinato software, di commentare sulle richieste, e di indicare la risoluzione, il risolutore e altre informazioni relative alle richieste. Solitamente tali software sono delle Web Applications, per permetterne un semplice accesso da parte di tutti gli utenti o clienti, anche se non mancano alcuni software che permettono di inviare le richieste ad un server centrale tramite particolari protocolli.

Il software libero più utilizzato per questo genere di supporto è sicuramente Bugzilla, sviluppato dal progetto Mozilla (autori del browser Firefox e del client di posta Thunderbird) per la gestione dei propri bug, ma successivamente utilizzato da molti altri progetti, grossi e non, come KDE, GNOME, FreeDesktop, Novell, Gentoo e altri ancora. Essendo software libero, pur non essendo di semplice installazione e gestione, quasi ogni installazione di Bugzilla è differente dalle altre; oltre alle modifiche alla veste grafica, che sono la parte più semplice della personalizzazione, alcune installazioni permettono la segnalazione guidata dei bug, utilizzando pagine diverse da quella originale, che facilitano la categorizzazione delle richieste.

È anche vero che moltissimi progetti non utilizzano altri tracker a parte quelli forniti dal loro servizio di hosting (SourceForge, BerliOS e così via), poiché non possono permettersi un server dedicato per Bugzilla (che richiede molte risorse per funzionare adeguatamente). A causa delle richieste tecniche necessarie per far funzionare Bugzilla, e la sua complessa gestione, anche quei progetti che non utilizzano servizi di hosting integrati cercano quando possibile soluzioni alternative per la gestione delle richieste. Un altro software molto utilizzato è Mantis, di più basso profilo, meno configurabile, ma molto più leggero, non avendo particolari richieste in risorse o in dipendenze.

Similmente esiste un software chiamato Trac, che unifica in una singola applicazione un Wiki, un'interfaccia Web al software SCM Subversion e un sistema di tickets (equivalente ad un bug tracker). Per quanto meno elaborato di Bugzilla e Mantis, Trac può essere facilmente utilizzato per la gestione dei bug di progetti di media dimensione. Si tratta comunque non di un semplice software per la gestione dei bug ma piuttosto di un cosiddetto "ambiente CASE integrato", ovvero di un software che permette di accedere, attraverso la stessa interfaccia, alle informazioni relative alle richieste, alla documentazione e alla storia dei sorgenti.

Ambienti integrati

Come si è visto in precedenza, esistono i cosiddetti "ambienti CASE integrati", (integrated CASE environments), ovvero software che permette l'accesso, tramite una singola interfaccia, a diversi strumenti CASE. Un esempio è Trac, software che fornisce un Wiki, un'interfaccia a Subversion come software SCM, un sistema di ticket e anche un sistema di release dei file. Similmente i software utilizzati da SourceForge e da GForce e GNU Savannah (i primi due proprietari, il terzo software libero), forniscono diversi strumenti utili per l'ingegneria del software (gestione delle release, gestione dei bug e delle richieste, gestione della documentazione e così via), sotto un'unica interfaccia permettendo la collaborazione e la coordinazione tra sviluppatori.

Ma se la maggior parte degli ambienti CASE integrati sono Web Applications, per rendere più semplice la collaborazione tra diversi sviluppatori, esistono anche ambienti integrati di sviluppo (IDE, Integrated Development Environment) che forniscono un'interfaccia unica a strumenti CASE differenti, come possono essere i software per la scrittura dei sorgenti (editor di testo), i compilatori e gli strumenti SCM.

I software più conosciuti per questi compiti sono sicuramente i software proprietari di Borland e Microsoft, che hanno iniziato lo sviluppo di tali ambienti, ma Eclipse di IBM e KDevelop del progetto KDE, entrambi esempi di software libero, stanno guadagnandosi un buon pubblico.

La maggior parte degli ambienti di sviluppo integrati forniscono funzionalità di supporto strettamente per lo sviluppo, e già il supporto per il software SCM è una funzionalità avanzata, non disponibile in tutti gli ambienti e non soprattutto non per tutti i software che possono utilizzarsi.

Quello che invece manca nella maggior parte degli ambienti di sviluppo è l'integrazione degli strumenti di sup-

porto alla progettazione.

Non è dunque possibile avere un singolo ambiente CASE integrato, seppure lo sviluppo del software CASE è ora molto più avanzato che negli scorsi anni. Anche le diverse Web Applications che forniscono supporto CASE non sono integrate tra loro, seppure l'esteso uso di XML riesce a permettere una generica integrazione di tante applicazioni Web, nulla è stato sviluppato per permettere l'integrazione dei vari software per la gestione dei progetti sulla falsariga di SourceForge.

Per ciò che riguarda l'integrazione dei bug trackers, per quanto Canonical (l'azienda che finanzia lo sviluppo della distribuzione Ubuntu) abbia in progetto lo sviluppo di un software, che si appoggi al loro già sviluppato LaunchPad, per l'integrazione di vari sistemi di tracking, nulla è ancora sviluppato che permetta semplicemente di collegare bug in diversi siti, ma Bugzilla già fornisce la possibilità di esportare i dati relativi ai bug tramite XML, il che fornisce uno spunto per l'utilizzo dei dati di un sito su un altro.

Tramite il giusto uso degli strumenti CASE elencati durante le varie fasi che compongono il processo di produzione del software è possibile far evolvere un software dalla sua origine fino ad un prodotto finale senza perdite di informazioni e senza richiedere la produzione di molta documentazione cartacea. La progettazione, la documentazione degli algoritmi, la documentazione delle interfacce, la storia del codice, degli errori e delle richieste segnalati saranno disponibili e, se viene usato un ambiente integrato, saranno pieni di riferimenti incrociati.

È dunque possibile, per esempio, sapere la ragione per cui è stata effettuata una modifica, perché il comportamento del software è stato cambiato in risposta alle richieste dei suoi utenti e così via; è possibile recuperare la documentazione prodotta durante la fase di progettazione e confrontarla con il codice che è stato realizzato; è possibile sviluppare il software in parallelo per affrontare problemi diversi, facendo evolvere due prodotti distinti a partire da una base di codice comune.

Purtroppo, il software di supporto CASE non è ancora abbastanza sviluppato per conformarsi adeguatamente ad alcuni metodi di sviluppo che si discostano troppo dai metodi generali che l'ingegneria del software ha tracciato, perché per esempio c'è software che non possiede alcuna fase di progettazione complessa e semplicemente si evolve durante la fase di realizzazione partendo da una base molto semplice e ascoltando le richieste degli utenti; in tal caso il supporto CASE importante è quello relativo al software SCM e ai bug trackers, mentre il software di progettazione non avrebbe più ragione per essere usato.

Si tratta quindi di un campo di sviluppo dell'ingegneria del software ancora aperto all'innovazione, e in cui ogni sviluppatore ha la possibilità di portare nuove idee e nuove implementazioni, grazie alla grande quantità di software libero già disponibile cui contribuire. Software scritto da sviluppatori per gli sviluppatori.

Diego Pettenò <flameeyes@gentoo.org>