IL PROGETTO CON UML: SUGGERIMENTI A PARTIRE DALLO UNIFIED PROCESS

La metodologia dello Unified Process è sicuramente piuttosto complessa, ma dà indicazioni preziose sui passi da seguire per la progettazione del software usando al meglio i diagrammi UML. Pertanto una sua versione semplificata, applicabile sempre, può seguire le fasi riportate nel seguito.

Occorre ricordare sempre che i passi della analisi e della progettazione non devono essere vissuti come imposizioni, bensì come un ausilio al mantenimento della precisione e, conseguentemente, di un migliore controllo sull'andamento dei lavori e del progetto in toto.

1. Prima raccolta "informale" dei requisiti

E' la prima fase del lavoro, se del caso congiunto con il cliente, ha lo scopo di produrre un documento informale, scritto in linguaggio naturale, che spieghi molto brevemente le caratteristiche del prodotto. In pratica questo primo documento serve a circoscrivere i limiti del lavoro successivo. Deve essere breve e il più possibile preciso e indicare con precisione l'argomento del progetto successivo.

Questo documento dovrebbe avere le seguenti caratteristiche:

1. Indicare chi è (o si immagina che sia) il cliente finale;
2. Indicare i requisiti che il cliente o il mercato richiede;
3. Definire di che tipo di lavoro si tratta;
4. Indicare vincoli imposti da altri software o sistemi o ambienti esistenti con cui si debba operare o entro cui il risultato del progetto debba operare;
5. Descrivere "informalmente" e a grandi linee il lavoro da svolgere.

2. Stesura del Glossario

In questa fase si deve definire la terminologia del progetto, identificando con precisione le entità (persone, ruoli, luoghi, oggetti materiali, eventi, strutturazioni ecc...) coinvolte nel sistema del mondo reale (ovvero del dominio di business) che hanno importanza per il sistema informatico obiettivo del progetto. E' importante identificare con precisione le entità, allo scopo sia di definire meglio i loro scenari d'uso (Passo 3, gli Use Case), sia di individuare le Classi Entità (Passo 4, il Class Diagram d'analisi).

Il risultato di questa fase è il documento Glossario, che definisce con precisione tutti i termini corrispondenti alle entità coinvolte, evitando ambiguità.

3. Stesura degli Use Case

In questa fase devono essere individuati con precisione gli scenari d'uso del sistema, ovvero, in modo più generale, gli scenari di interazione fra il sistema e gli attori, ovvero le entità esterne al sistema con cui esso interagisce e comunica. I passi necessari in questa fase possono essere così suddivisi:

1. Definizione esatta dei boundary o confine del sistema (entro sistemi particolarmente complessi questa fase può anche essere applicata a sotto-sistemi);
2. Identificazione e definizione degli attori, ossia delle entità esterne con cui il sistema (o i sotto-sistemi) oggetto dell'analisi interagiscono e comunicano;
3. Individuazione dei vari scenari di uso/interazione fra sistema ed attori, che corrisponderanno ai singoli casi d'uso, identificati dalle singole ellissi nel diagramma; si ricordi che i casi d'uso descrivono cosa si vuole che il sistema faccia, non come questo comportamento deve essere implementato (modello black box);
4. Definizione delle interazioni entro i singoli casi d'uso; tali interazioni, strutturate nella forma della richiesta dell'attore cui corrisponde una risposta del sistema (tenendo conto anche di eventuali comunicazioni asincrone o autonome del sistema quali ad esempio allarmi), andranno a costituire i campi descrizione dei singoli casi d'uso (operazione detta in gergo "srotolamento" dello use case);
5. Esame dei diagrammi così ottenuti e delle loro descrizioni per potere procedere alla raccolta a fattore comune di parti fra i singoli use case entro diagrammi, facendo uso delle relazioni extends ed include definibili tra i vari casi d'uso;
6. Il passo 5 può essere iterato più volte; occorre tenere conto della granularità del problema e del grado di definizione e precisione che si vuole raggiungere; inoltre occorre tenere presente che un singolo caso d'uso spesso dà origine ad una singola maschera (sia essa maschera testuale, singola window in ambiente GUI o pagina Web); infine si tenga presente che spesso da un caso d'uso deriverà anche un caso di test durante la fase di test del sistema informatico realizzato.

Il prodotto di questo passo è l'insieme completo degli use case inseriti entro uno o più use case diagram, ognuno corredato di adeguata descrizione, strutturata chiaramente in forma di request-response e considerando sia il percorso principale di interazione (basic course) sia gli eventuali percorsi alternativi (alternative courses), quali quelli che si verificano in presenza di errori nei dati introdotti ecc... Il diagramma e le descrizioni devono essere ben strutturati, chiari ed esaurienti, in quanto tutti i passi successivi si baseranno su di essi.
Siccome gli use case diagram non esprimono direttamente relazioni di flusso logico/temporale fra i loro componenti, può essere utile esplicitare tali relazioni attraverso un activity diagram derivato, che definisca le attività associate ai singoli use case (potrebbero in tal caso essere necessarie ulteriori scomposizioni od aggregazioni) e le relazioni logiche e temporali che tra esse intercorrono. Da questo diagramma deriva, più o meno direttamente, anche il diagramma di navigazione fra le finestre o maschere che costituiscono l'interfaccia esterna utente dell'applicazione in progetto.
Se necessario si possono aggiungere ulteriori descrizioni, che rendano più preciso il tutto.

4. Stesura del Class Diagram di Analisi

In questa fase, che parte dal Glossario realizzato in fase 2 e dallo/dagli Use Case Diagram realizzati in fase 3, deve essere realizzato il diagramma delle classi di analisi. Tale diagramma deve indicare chiaramente tutte le classi entità, ossia le classi definibili come "proiezioni" nel dominio della applicazione software delle entità del dominio del problema dove la applicazione software andrà ad operare, più eventuali altre classi individuate nel corso dell'analisi che siano di importanza per i concetti funzionali che definiscono i requisiti del progetto. In pratica nel diagramma, che è l'equivalente da un punto di vista del ruolo (e l'evoluzione da un punto di vista storico e metodologico) del diagramma Entità-Relazioni (ER) usato nelle metodologie di sviluppo più tradizionali, devono essere chiaramente indicate:

1. Tutte le classi entità che fanno parte del dominio del problema;
2. Gli attribuiti caratteristici di tali classi, eventualmente procedendo alla individuazione dei singoli attributi o dei gruppi che consentano una identificazione univoca delle istanze delle classi, ovvero dei singoli oggetti; tali attributi costituiscono le chiavi;
3. Le associazioni che tra tali classi intercorrono, ossia tutti i legami logici che tra esse intercorrono; queste associazioni (che corrispondono alle relazioni dei diagrammi ER) sono importanti perché in sede implementativa di codice indicheranno anche la visibilità necessaria tra le classi, cioè quali altre classi (eventualmente appartenenti ad altri package o namespace) una certa classe dovrà vedere, definendo quindi la loro interdipendenza;
4. I versi di tali associazioni (ad esempio, se la classe magazzino deve conoscere la classe prodotto, non è sempre vero il viceversa);
5. Le molteplicità di tali associazioni (es. uno-a-molti, molti-a-molti), l'eventuale necessità di definire classi di associazione (ad esempio la proprietà dell'auto che svolge il ruolo di classe di associazione fra proprietario ed auto);
6. Eventuali rapporti di inclusione legati a tali associazione, suddivisi fra aggregazione e composizione; si ricordi che l'eliminazione di una composizione, indicata con il diamante nero, elimina anche tutti i suoi elementi componenti, mentre l'eliminazione di una aggregazione, indicata con il diamante bianco, non elimina anche i componenti, che hanno anche una natura indipendente;
7. Eventuali rapporti di ereditarietà fra le classi, ottenuti applicando i principi di generalizzazione e specializzazione, ovvero "raccogliendo a fattore comune" attributi e metodi o aggiungendone di nuovi;
8. Si ricordi che da questo diagramma, eventualmente passando attraverso un diagramma EER, deriverà anche la base dati relazionale dell'applicazione: i rapporti di molteplicità devono essere chiari perché dalle associazioni derivano le relazioni tra le chiavi che collegano le tabelle entro la base dati;
9. I metodi delle classi possono ancora non essere completamente definiti in questa fase.

Il processo che conduce al diagramma finale è ovviamente iterativo e può dirsi stabilizzato quando tutte le relazioni (in senso ampio) fra le classi sono chiaramente individuate. Il Class Diagram di Analisi è fondamentale per tutti i passi di progetto che seguono.

5. Scelta architetturale e definizioni conseguenti

La scelta architetturale è un passo fondamentale, in quanto i passi successivi sono da essa condizionati. Esistono comunque regole generali importanti che aiutano nello svolgimento, quali il pattern Model-View-Controller ed il conseguente approccio multicanale alla realizzazione delle interfacce utenti. Seguendo tale metodo si separa nettamente la interfaccia utente vera e propria (View), che ha lo scopo di presentare semplicemente dati all'utente ed è ovviamente soggetta ai vincoli dal tipo di mezzo o canale utilizzato (interfaccia a finestre grafiche, Web, PDA, cellulare, Set-Top Box TV...), dal reattore agli eventi trasmessi dall'utente (Controller), che usa i metodi forniti dagli strati interni dell'applicazione (Model e relativi Adapter) per garantire all'utente i servizi associati agli eventi inviati dall'utente stesso. Grazie all'approccio multicanale, eventualmente corredato dall'uso di altri strati di Adapter, diviene possibile riutilizzare (almeno in buona parte) il controller (ed ovviamente gli strati sottostanti) cambiando solo la view quando si cambia canale, passando, ad esempio, da una applicazione Window ad una Web sostituendo alla finestra il servlet.
Nel Web in particolare, seguendo il metodo MVC-2, il servlet fa il ruolo di adapter del controller, al cui interno stanno poi gli adapter del model, mentre la view è implementata con una pagina JSP, permettendo riadattamenti grafici estremamente rapidi.
La scelta dell'architettura deve anche segnalare limiti e criticità nel sistema che sarà realizzato.
L'output di questa fase sono documenti tecnici architetturali, che saranno poi corredati da eventuali Component Diagram e Deployment Diagram solo al termine della fase di progetto vera e propria.

6. Definizione del Class Diagram di Progetto

In questa fase occorre definire chiaramente tutte le classi che fanno parte dell'applicazione software da implementare. Il Class Diagram di Progetto è l'elenco completo delle classi, con tutte le loro relazioni e su di esso si basa anche il dimensionamento della fase di sviluppo (ovvero scrittura vera e propria del software).
Il processo che permette di giungere al diagramma delle classi di progetto è necessariamente iterativo. Si parte dal diagramma delle classi di analisi e devono essere inserite tutte le classi di servizio, ossia le classi infrastrutturali, non necessariamente derivate dalla fase di analisi, che permettono al programma nel suo insieme di operare correttamente ed in modo efficiente. Le classi di servizio sono ovviamente fortemente dipendenti nella loro struttura dall'architettura scelta e da eventuali framework utilizzati nel progetto. Se un diagramma di analisi ben fatto può essere spesso utilizzato con diverse tecnologie ad oggetti, ovvero essere punto di partenza per progetti analoghi realizzati su piattaforme diverse, un diagramma di progetto è chiaramente molto più influenzato dalla tecnologia usata. Il processo usa anche altri diagrammi UML.

1. I diagrammi di interazione (sequence, che pone enfasi sulla sequenza temporale delle interazioni, e collaboration, che pone enfasi sulla dipendenza fra le classi) sono di importanza fondamentale sia per la definizione dei metodi che le classi offrono le une alle altre (e dei loro argomenti e valori di ritorno), sia per l'individuazione di eventuali "colli di bottiglia" che vengono risolti con l'inserimento di nuove classi. In teoria ad ogni use case corrisponde almeno un sequence o collaboration diagram: infatti ogni corso di eventi individuato nell'analisi con gli use case dovrebbe produrre una precisa sequenza temporale di invocazione di metodi all'interno dell'insieme delle classi costituenti il sistema software. Non sempre è

però indispensabile realizzarli tutti, specie nei casi di corsi di eventi molto simili, nel qual caso bastano le opportune descrizioni di accompagnamento.
2. I diagrammi di attività, che derivano anch'essi dagli Use Case, dando ad essi una sequenza temporale e logica, possono aiutare molto nella definizione della Mappa di Navigazione fra le finestre, consentendo di definire completamente l'interfaccia utente di un applicativo ed eventualmente di realizzare i prototipi d'analisi (finestre vuote vere e proprie o gli schematics).
3. I diagrammi di stato sono anch'essi molto importanti per valutare l'evoluzione temporale delle singole classi (o meglio degli oggetti da esse istanziati) o di sotto-sistemi che esse vanno a costituire, aiutando ad individuare eventuali condizioni critiche o colli di bottiglia.

L'obiettivo finale è comunque la realizzazione del Class Diagram di Progetto, completo di tutte le classi. Spesso per motivi di chiarezza (specialmente in progetti grandi dove le classi sono molto numerose) il diagramma viene diviso in package, associazioni di classi corrispondenti ad unità funzionali, indicando esternamente ad essi solo i legami che fra i singoli package intercorrono. Ciascun package viene poi rappresentato completamente entro un diagramma di secondo livello. Quasi sempre questa suddivisione funzionale viene anche portata a livello implementativo servendosi delle aggregazioni tipiche dei linguaggi (package del Java, namespace di C#). L'obiettivo deve essere sempre quello di avere un diagramma leggibile, che serve come mappa per lo sviluppo. Da questo diagramma possono anche essere generati gli scheletri delle classi attraverso opportuni strumenti CASE (ad esempio PoseidonUML), oppure essere ottenuti i Fogli di Specifica, ossia i documenti che descrivono ciascuna classe con attributi, metodi, vincoli e controlli da implementare.

7. Definizione delle strutture di contorno

Usando i diagrammi realizzati in precedenza si arriva a definire le parti implementative di contorno del progetto, che devono essere opportunamente documentate come segue.

1. Definizione della base di dati, attraverso un EER, eventualmente corredato dagli script di creazione delle tabelle e vincoli che genera la base dati nello specifico DBMS scelto.
2. Definizione dell'insieme dei singoli componenti software (package, DLL, Jar ecc...) che devono essere prodotti, con l'indicazione delle loro interdipendenze, attraverso un opportuno Component Diagram o più di uno.
3. Definizione della distribuzione dei componenti sulla o sulle piattaforme di produzione prescelte, attraverso uno o più opportuni Deployment Diagram.
4. Stesura di opportuni documenti Readme ed altro che corredano il progetto e la installazione; in particolare devono essere chiaramente indicati eventuali limiti e/o malfunzionamenti della/e piattaforma software ed hardware utilizzata.
5. Stesura dell'opportuno manuale utente dell'applicazione, secondo i criteri stabiliti.
6. Definizione delle scadenze e pianificazione dell'esecuzione temporale del progetto, in base ai dimensionamenti svolti e alle risorse a disposizione.
7. Definizione dei test e dei singoli casi di test.
8. Pianificazione del collaudo e dell'entrata in produzione.
9. Definizione della successiva fase di manutenzione.