DESIGN PATTERN E FRAMEWORK: MATTONI E CEMENTO DELLA MODERNA PROGETTAZIONE DEL SOFTWARE

FELICE PESCATORE

Con questo secondo articolo continuiamo ad analizzare i moderni scenari di sviluppo del software e come in esso il ruolo dell'ingegnere assuma contorni sempre meno sfumati. Nonostante l'ingegneria del software soffra ancora della mancanza di metodologie forti e consolidate negli anni e della frenetica evoluzione delle tecnologie disponibili, negli ultimi anni si sono consolidati strumenti che hanno permesso di "nobilitare" l'attività di sviluppo del software permettendo di studiare il progetto ad un livello più alto. Ci riferiamo ai **Design Pattern** ed ai **Framework**.

I **Design Pattern** sono una soluzione nota e rodata ad un problema ricorrente. Praticamente ogni volta che ci si trova davanti ad una scelta architetturale per la risoluzione di un problema si può fare riferimento ad una sorta di "catalogo" che presenta micro-architetture astratte già utilizzate, funzionanti ed implementante nei più diffusi linguaggi di sviluppo. Formalizzati tipicamente attraverso il linguaggio UML (Unified Modeling Language, indipendente dalla tecnologia di sviluppo), i design pattern permettono di concentrare gli sforzi sugli aspetti funzionali del sistema. La bravura di un buon progettista sta nel saper individuare all'interno del proprio blue-print le parti cruciali ed il loro modo di interagire al fine di selezionare il pattern che più si adatta al contesto. Nella pratica di tutti i giorni pattern come il Factory o il Singleton sono utilizzati comunemente per realizzare sistemi efficienti, chiari e ben organizzati. Si pensi, per esempio, al contributo dato dal pattern Model View Control (MVC) nello sviluppo di una web application, introdotto con lo scopo di realizzare una separazione netta tra i moduli che la compongono.

Una combinazione di pattern, proprio grazie alla loro intrinseca genericità, rappresenta concettualmente un **Framework**, ovvero una serie di elementi funzionali che collaborano tra di loro formando un design riutilizzabile per una specifica classe di software. E proprio da questo connubio nasce la similitudine introdotta dall'articolo: i pattern possono essere considerati il "cemento" che tiene insieme uno o più framework e questi ultimi rappresentano di conseguenza i "mattoni" del nostro sistema.

Un framework, quindi, può essere visto come lo "scheletro" di un'applicazione, offrendo una soluzione generica ai problemi di un ben determinato dominio e come tale va personalizzato (attraverso gli strumenti messi a disposizione dal framework stesso) per rispondere alle specifiche esigenze di contesto. Il vantaggio nell'utilizzo dei framework consiste nel fatto che le decisioni più importanti riguardanti il design sono già state prese, permettendo uno sviluppo più veloce ed una migliore manutenzione. Rispetto ad una libreria funzionale un framework realizza quello che comunemente va sotto il nome di **principio di Hollywood**: *"Don't call us, we call you"*. In pratica mentre una libreria viene "chiamata" dal programma che la utilizza e quindi è l'elemento passivo, nel caso dei framework accade l'inverso, ovvero è il framework a "chiamare" i componenti custom relativi allo specifico problema.

Nei diagrammi seguenti si evidenzia come lo *Scratch Development* (implementazione da Zero) dedica i primi mesi di sviluppo alle scelte di Design e alla relativa implementazione, al contrario di ciò che avviene utilizzando un framework.

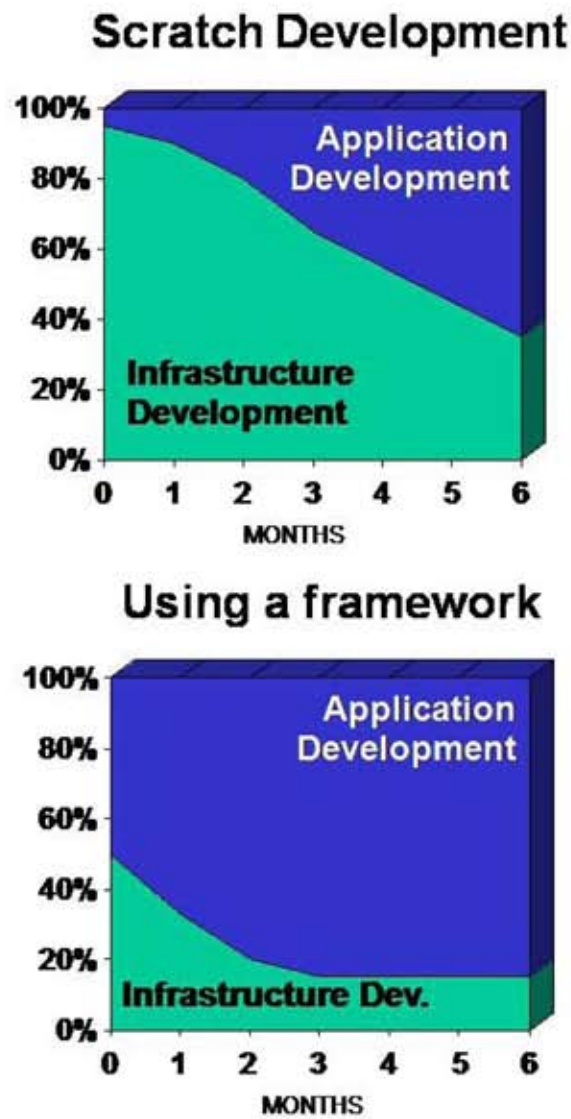

Figura 1 - Scratch Development e Framework Based Development

La disponibilità di un framework valido permette di poter affrontare rapidamente diversi problemi di uno

stesso dominio grazie alla facilità di riuso e reagire in modo efficiente ai cambiamenti dei requisiti.
Un framework non è però solo codice, infatti un sistema può definirsi tale solo se dispone di:

- **Knowledge base**, ovvero una buona base documentale che descrive gli elementi caratterizzanti del sistema;
- **Document templates**, ovvero i dettagli relativi ai requisiti, ai use case, alle specifiche di design, ai test plans, ecc;
- **Design patterns**, standards, ovvero utilizza standard e soluzioni note rodate;
- **Code library**, una nutrita libreria per le esigenze di verticalizzazione;
- **Code templates and generators**, generatori di stub basati su template.

Nonostante esistano molteplici framework, con notevoli differenze tra loro, di base essi risolvo questioni quali: *Persistenza dei Dati e relativi Strumenti di Accesso, Lookup delle Risorse, Concorrenza, Logging, Transizioni, Sicurezza, Gestione degli Errori, Gestione degli Utenti, Scheduling, Reporting.*

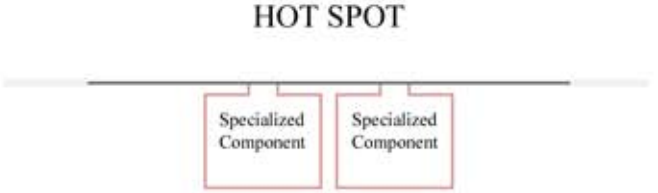

Figura 2 – Hot Spot

Attualmente possiamo dividere i Framework in due grosse categorie:

- *Framework di sviluppo*, come dotNet e Java al cui interno trovano posto sia dei framework funzionali che collezioni di librerie.
- *Framework applicativi*, come, ad esempio, il pluripremiato Ruby on Rails per lo sviluppo web che fa del citato MVC la ragion d'essere.

Quando non usare un framework

Dalla breve analisi proposta potrebbe sembrare che i framework sono la panacea di tutti i mali. In realtà non sempre però il loro utilizzo è auspicabile. Ad esempio uno dei settori in cui è assolutamente sconsigliato utilizzare un framework è quello in cui i requisiti non funzionali impongono un'architettura in grado di sfruttare in modo estremamente efficiente anche l'ultima frazione di risorse disponibili (si pensi alle applicazioni real-time o ai sistemi di controllo degli impianti industriali) o i contesti in cui è inutile introdurre architetture troppo raffinate con numerose funzionalità extra che porterebbero solo ad una complicazione dell'intero ciclo di sviluppo.
Come ogni altro strumento informatico è essenziale fare un'accurata analisi e verificare l'opportunità o meno dell'utilizzo di un framework piuttosto che una libreria o altri sistemi di supporto.

Il Ruolo dell'esperto e delle Community

Nel processo di realizzazione di un framework gioca un ruolo fondamentale la capacità dell'ingegnere del software di analizzare il dominio applicativo del problema sintetizzandone gli elementi basilari al fine di realizzare un'architettura generalizzata da verticalizzare attraverso le tecniche evidenziate. Un framework di successo evita ai suoi utilizzatori di ricorrere ad artifizi per personalizzarlo ed è in grado di evolvere mantenendo la compatibilità con le verticalizzazioni realizzate (grazie a meccanismi quali i *contracts* o le *interface*). L'ingegnere ha un ruolo cruciale anche nella scelta delle tecnologie di base per la realizzazione del nuovo framework, spalmando il risultato della fase di analisi sulle soluzioni esistenti ed utilizzando gli strumenti idonei più aggiornati. Un nuovo framework raramente nasce da zero ma si basa su altri framework specialistici portando così a compimento uno dei principi fondamentali nell'ingegneria del software: *il riuso*.
Lo specialista deve quindi avere un'ampia conoscenza dello stato dell'arte delle tecnologie esistenti, in modo da accelerare lo sviluppo dei nuovi prodotti concentrandosi sull'aspetto di Business, piuttosto che sprecare tempo e risorse sugli aspetti per i quali già esiste una soluzione valida (chi, oggi, realizzerebbe un sistema di persistenza dei dati da zero, piuttosto che ricorrere ad uno dei framework disponibili?).
L'evoluzione del web ha poi impresso una notevole accelerazione alla realizzazione e all'utilizzo dei framework, introducendo un ulteriore tassello nel quadro che stiamo analizzando: le *Community*. Oggi giorno è difficile considerare un framework veramente valido se non dispone di una schiera di sviluppatori ed utilizzatori (community, appunto) che ne seguono l'evoluzione, apportano continui miglioramenti e supportano gli utilizzatori terzi dando vita ad una vera e propria Knowledge Base.

Concludendo

A valle di questa sintetica introduzione all'argomento possiamo affermare che Pattern e Framework sono due facce della stessa medaglia: i pattern rappresentano la visione astratta del sistema, individuando oggetti, collaborazione e responsabilità. Un framework, al contrario, è un'istanza concreta del sistema che incorpora pattern a livello di micro-architettura.
L'utilizzo di entrambi porta alla realizzazione di solidi ed efficienti sistemi software specializzati nella risoluzione di particolari classi di problemi permettendo al team di sviluppo di concentrarsi sui requisiti funzionali senza troppo preoccuparsi delle problematiche di contorno.