Università degli Studi di Milano
Facoltà di Scienze e Tecnologie
Corso di Laurea in Informatica

PROGETTAZIONE DI UN'APPLICAZIONE E CONSEGU—DENZE DI UN APPROCCIO "CONTINUOS REFACTORING"

Relatore: Prof. Carlo Bellettini

Tesi di:
Samuel Gomes Brandão
Matricola: 803939

Anno Accademico 2013-2014

Dedica

Ai miei genitori Mônica e Zilmar
Aos meus pais Mônica e Zilmar

Prefazione

Ciao! Ci vuole scrivere una prefazione!

Organizzazione della tesi

La tesi è organizzata come segue:

- nel Capitolo 1 ....

Ringraziamenti

Vorrei ringraziare a Paolo Venturi, Diego Costantino, al prof. Dr. Carlo Bellettini.

Indice

Dedica **ii**

Prefazione **iii**

Ringraziamenti **iv**

1 Introduzione **1**

2 Attività Preliminari **2**

2.1 La consistenza dei dati 3

2.2 Le scelte tecnologiche a partire dai dati 4

3 Svolgimento delle Attività **6**

3.1 *Experimentation Driven Workflow* 7

3.2 La naturale migrazione verso un *workflow* prototipale 8

3.3 Non regression test 9

3.4 Refactoring guidato - SOLID 10

Capitolo 1

Introduzione

Spazi Unimi è il nome dato a un progetto per l'ottenimento di dati sugli spazi dell'Università degli Studi di Milano, sviluppato durante il Tirocinio Interno per la laurea triennale in Informatica all'UNIMI, da Samuel Gomes Brandão, Diego Costantino e Palo Venturi. L'idea nasce a partire dalle proposte del progetto Campus Sostenibile, promosso dall'UNIMI e dal Politecnico di Milano, e si sviluppa posteriormente in autonomia, sotto l'orientamento del Prof. Dr. Carlo Bellettini.

Il progetto parte con lo sviluppo di un'applicazione software con lo scopo principale di estrarre, validare e correggere dati ottenuti da diverse sorgenti, procedendo in seguito alla loro integrazione. I dati vengono mantenuti su un database da venir utilizzato per futuri progetti attraverso l'uso di una specifica Application Programming Interface (API) con architettura REST (Representational State Transfer).

Le informazioni integrate riguardano la topologia e la destinazione d'uso degli edifici universitari e i loro spazi, con particolare importanza alla elaborazione e presentazione delle piante interne e localizzazione di stanze precise. In questo modo, siamo in grado di fornire accuratamente informazioni sulla localizzazione di palazzi, aule, o stanze a secondo della loro tipologia d'uso (bagni, biblioteca, sale studio, ecc).

Su questa relazione descrivo il processo di sviluppo della suddetta applicazione da zero, con particolare rilievo alle sfide per la buona progettazione e all'utilizzo di un concetto che ho chiamato "continuos refactoring" [1]

[1] Si veda il capitolo 3

Capitolo 2

Attività Preliminari

> Experience is simply the name we give our mistakes
>
> Oscar Wilde

La prima sfida per lo sviluppo di un progetto è quella di capirlo, e può richiedere tempo e dedicazione considerevole. Molto spesso però il capire avviene - e lo può soltanto - durante lo sviluppo stesso.

Il primo passo è stato quello di pensare ai casi d'uso che volevamo coprire con la nostra applicazione. Da questo punto abbiamo proceduto verso la comprensione dei dati e delle informazioni disponibili e la scelta delle tecnologie più adeguate per la loro elaborazione. Le sorgenti dati che dovevamo integrare erano due principali, a loro volta suddivise in più tipologie di informazione e formati di provenienza.

Dal dipartimento di Edilizia dell'UNIMI abbiamo ottenuto le piante architettoniche dei palazzi utilizzati dall'Università, in formato DWG - un formato file proprietario, e una serie di fogli elettronici con informazioni dettagliate sui palazzi e sulle loro stanze, in speciale quelle utilizzate per scopi didattici.

Con l'aiuto anche della divisione di Sistemi Informativi, rappresentata in speciale da Vincenzo Pupillo, abbiamo ottenuto in formato testuale CSV (comma separated values) le informazioni utilizzate dal sistema Easyrooms, che mira a fornire informazioni rispetto all'uso didattico degli spazi (eventi, lezioni, capienza delle aule, lauree, eccetera).

Entrambi i dipartimenti ci hanno aiutato con totale disponibilità e trasparenza, e senza il loro aiuto il nostro progetto non sarebbe mai stato portato a buon fine.

Con queste informazioni in mano ne abbiamo incominciato l'analisi, cercando di capire non solo le loro criticità, ma anche dove si sovrapponevano, completavano o fossero ridondanti. Al primo contatto ci sembravano perfette: avevamo informazioni di destinazione delle stanze, la loro capienza, accessibilità a disabili, e le potevamo localizzare sulle piante architettoniche utilizzando il loro codice identificativo. Dalle piante ottenevamo anche la localizzazione di aree di interesse come biblioteche, sale studio, bagni, spazi di ristorazione, eccetera.

2.1 La consistenza dei dati

Sotto uno sguardo più attento, però, l'immagine mentale che ci eravamo costruiti di quei dati incominciò a rilevare i suoi difetti: le fessure venivano come errori di battitura, l'utilizzo duplice di codici identificativi per i palazzi, piante architettoniche fuori scala o semplicemente disegno della stessa stanza più di una volta sullo stesso file. Spesso i disegni venivano ripetuti sulla stessa posizione, probabilmente frutto di un'operazione di copia e incolla interrotta a metà. All'occhio umano saltavano facilmente gli errori, che riuscivamo a correggere facilmente, guardandoli e riconoscendo dei pattern di riferimento, ma per un elaboratore no sarebbe così facile. Soltanto a un programmatore interessato ad estrarre informazioni in modo automatico con l'uso di un elaboratore questi errori avrebbero causato danni, e allora ci toccava gestirli.

Con buona probabilità questi errori sono stati accumulati lungo gli anni, ed è totalmente comprensibile che ci siano, se consideriamo ad esempio che la maggior parte delle piante architettoniche di cui disponevano sono state disegnate in formato cartaceo e solo posteriormente trasferite in formato digitale, quando la costruzione dell'edificio era già finita. Sono inoltre state fatte e raccolte lungo periodi significativi, create da persone diverse, e perciò è difficile mantenere degli *standard* nella loro rappresentazione digitale. Ovviamente nel processo di trasferimento da cartaceo a digitale dettagli vengono persi e errori introdotti, e le versioni digitali non passano per la stessa meticolosa verifica a cui vengono sottoposte quelle utilizzate per la costruzione degli edifici.

La molle di dati era significativa, in speciale per i file DWG: più di 700, ognuno con dimensione media di 4.5Mb, e casi estremi di fino a 44Mb. Questi file contenevano tutte le informazioni edili: tubature, finestre, porte, scale, sezioni dei palazzi, disegno dei muri, cortili, terrazze, eccetera. Ci è stato necessario meno di una settimana di contemplazione e analisi di quei dati per capire che l'unico modo di giudicarne la loro qualità e se erano in grado di soddisfare le nostre esigenze sarebbe stato incominciando con la loro estrazione e cercando di imparare sul processo. Queste caratteristiche hanno determinato tanti aspetti dello sviluppo, addirittura il nostro workflow, che frequentemente si è dimostrato "REPL Driven" o "Experimentation Driven".

Nessuna assunzione

Dalle analisi iniziali abbiamo concluso che potevamo fare poche o nessuna assunzione sulla qualità, formato, presenza o consistenza dei dati. Alcuni esempi di assunzioni che abbiamo concluso non essere possibili e che avrebbero agevolato significativamente lo sviluppo sono la presenza o meno di codici identificativi univoci per i palazzi dai file testuali, la presenza di un identificazione univoca del piano e palazzo a cui ogni file DWG si riferisce, e l'univocità degli identificativi di piani.

2.2 Le scelte tecnologiche a partire dai dati

Questa natura dei dati trasformava tante delle nostre richieste in *"Wicked Problems"*, problemi la cui comprensione avviene direttamente durante la loro risoluzione e non è possibile *a priori*. Questo è stato uno dei primi motivi che ci ha fatto scegliere Python 3 come linguaggio di riferimento per l'estrazione e l'elaborazione: la capacità di scrivere velocemente degli script in grado di estrarre e produrre dettagliate analisi delle caratteristiche dei dati.

Oltre a questo aspetto, anche i seguenti punti hanno contribuito alla scelta di Python:

- La presenza delle *list comprehentions*, un potente strumento per l'esecuzione di operazioni di trasformazione e filtraggio dei dati, specialmente se associate a funzioni per l'aggregazione di dati e agli iteratori di Python 3.
- L'esistenza di una forte comunità di sviluppatori e entusiasti per il linguaggio, specialmente in Italia.
- Ampia presenza di librerie di supporto per le attività che dovevamo eseguire (lettura delle piante, elaborazioni geometriche, ecc).
- Essendo Python un nuovo linguaggio per tutti i partecipanti al progetto, rappresentava una positiva sfida didattica.
- La scelta di Python non sarebbe limitante per la continuazione del progetto in futuro da parte di altri studenti/tesisti, in quanto è anche insegnato all'università.

Per quanto riguarda la scelta del DBMS (Database Management System) è stata l'inconsistenza dei dati a guidare la scelta: le informazioni presenti sui diversi palazzi non erano omogenee in termini quantitativi ne qualitativi. Su qualche edifici disponevamo di più informazioni topologiche mentre su altri quasi nessuna. Anche le informazioni inerenti all'edificio stesso (come il suo nome rappresentativo o scopo d'utilizzo - ad esempio "Dipartimento di Informatica") non sempre erano presenti o validi, e ciò si ripeteva anche sugli altri dati. L'uso di un DBMS relazionale avrebbe comportato degli *schema* con considerevoli campi null e denormalizzato. Per questi motivi abbiamo scelto un DBMS che seguisse un modello di memorizzazione *schemaless*.

Per le note caratteristiche prestazionali, supporto nativo a calcoli su coordinate geografiche, presenza di forte comunità, documentazione chiara e completa e la diversità di librerie aggiuntive disponibili, abbiamo scelto MongoDB come DBMS di riferimento.

Capitolo 3

Svolgimento delle Attività

Abbiamo incominciato il progetto affrontando quello che ci sembrava l'aspetto più difficile, cioè l'estrazione dei dati delle piante architettoniche. Abbiamo convertito i file DWG in DXF utilizzando uno strumento gratuito, dato che era il formato richiesto dalla libreria dxfgrabber [1] che abbiamo utilizzato.

Il primo passo era quello di estrarre le stanze, e in seguito le etichette e testi associati a ogni stanza, il disegno dei muri e finestre (per scopi estetici principalmente), la posizione delle porte, scale e ascensori. Ognuno di questi elementi si è rivelato più complicato di quanto ce ne aspettavamo.

La lettura delle stanze è stata poco problematica. Le difficoltà maggiori sono sorte in fase di ottenimento delle etichette appartenenti alle stanze e nell'associarle. L'unico vincolo fra le etichette della stanza e il suo contorno era la posizione del testo e del disegno, senza nessun vincolo sintattico. Nel 90% dei casi ciò non ci causava problemi, ma in un 10% significativo causava errori scomodi, specialmente in presenza di stanze piccole o strette (come corridoi ad esempio), in cui l'etichetta, per mancanza di spazio, poteva venir collocata al di fuori del contorno della stanza, nelle sue vicinanze, potendo addirittura finire all'interno del contorno di altre stanze. All'occhio umano era semplice capire a quale stanza tale etichetta appartenesse, ma lo stesso era molto difficile in termini algoritmici.

[1] Libreria *opensource*, sorgente disponibile in https://bitbucket.org/mozman/dxfgrabber

3.1 *Experimentation Driven Workflow*

Per *Experimentation Driven Workflow* intendo una modalità pratica di sviluppo del *software* che attraverso l'esplorazione pratica del dominio applicativo, dei problemi da risolvere, e della natura dei dati, si prefigge di:

- Garantire una comprensione più approfondita del problema.
- Rivelare in che modo il linguaggio utilizzato, i suoi moduli o librerie esterne possono contribuire alla risoluzione del problema
- Capire caratteristiche importanti/comuni su una molle considerevole di dati.
- Eventualmente generare codice da venir utilizzato nell'implementazione finale.

Come strumento per l'esplorazione abbiamo scelto la REPL (*Read Eval Print Loop*) di Python stessa. In questo modo abbiamo potuto esplorare i problemi/dati nell'ambiente stesso su cui avremo dovuto poi implementare le soluzioni. In questo modo, il *workflow* basico è composto dai seguenti passi:

- Comprendere teoricamente la natura del problema
- Utilizzare la REPL per conoscere meglio i dati e le sfide implementative
- (Opzionale) Creare un prototipo o bozza che permetta la risoluzione del problema e l'esplorazione di ulteriori aspetti. ???? dettagliare ???

Il prototipo/bozza differisce da un'implementazione finale principalmente per aspetti di completezza (può non gestire ad esempio casi limite) e qualità del codice e prestazioni (viene scritto il più veloce possibile in modo da non bloccare il *workflow*, ma non necessariamente è un'implementazione efficace o chiara).

La nostra soluzione per il problema delle etichette delle stanze è nata direttamente dall'approccio di esplorazione utilizzato. Con l'uso della REPL, abbiamo analizzato tutti i file di cui disponevamo e le loro stanze, associando un'etichetta a ognuna delle stanze. Se a una stanza venissero associate più etichette, la scartavamo. Con questa strategia siamo stati in grado di mappare etichette al 98% delle stanze disponibili. Del 2% senza etichette, meno di un quarto erano di tipo didattico (aule, laboratori, ecc), e allora per i nostri scopi potevamo stimare un tasso di *recall* del circa 99% con precisione molto simile, e ciò era più che soddisfacente.

Infatti, durante lo sviluppo, l'uso della REPL per l'esplorazione di questi dati prima dell'effettiva sintesi degli algoritmi e scrittura del codice ci ha permesso di affrontare i problemi in modo più efficiente. Ci permetteva subito di esperimentare con i dati, capire caratteristiche comuni o analizzare i loro attributi, per poi migrare quelle conclusioni verso un'implementazione effettiva.

A volte il ottenevamo dalla REPL un modulo già semilavorato, a volte semplicemente una media campionaria che ci permetteva di avere più confidenza nelle scelte prese. Quando una precisione assoluta e deterministica non era possibile, l'analisi delle qualità dei dati giustificava assunzioni che ci permettevano di procedere a un'estrazione **sufficientemente o statisticamente buona**, superando le difficoltà dell'analisi e la difficoltà di correggere errori umani. In altre parole, ci permetteva di continuare anche quando in termini assoluti sarebbe stato sbagliato farlo.

3.2 La naturale migrazione verso un *workflow* prototipale

Molti componenti dell'applicazione finale sono nati a partire da questi esperimenti con la REPL, incentivando un approccio alla progettazione di tipo prototipale. Da queste versioni prototipali ottenute dalla sperimentazione, ottenevamo versioni compiute del prodotto eseguendo delle continue trasformazioni e operazioni di *refactoring*. Perché funzionasse correttamente, le prove sulla REPL dovevano essere estremamente veloci, senza preoccupazioni con la qualità del codice prodotto e senza neanche attività di *testing* formale.

Il principale vantaggio era quello di poter conoscere la natura dei problemi o i limiti e le potenzialità delle nostre scelte algoritmiche, senza dover pagare un prezzo troppo alto se mai fosse necessario tornare in dietro. Fino a questo punto non si trattava di design di software, ma di sola esplorazione e comprensione. Una volta capito meglio il *wicked problem* in questione, lo sviluppo proseguiva in una di due possibili strade: scartare il prototipo per rifarlo da capo, o rielaborarlo.

Nel caso venisse scartato, un approccio di tipo Test Driven Design (TDD) veniva utilizzato per la sua ricostruzione. In questo modo potevamo centralizzare gli sforzi

nella definizione del design e dell'interfaccia della componente, dato che le procedure per la risoluzione del problema erano già state esplorate.

In qualche casi, però, ci sembrava evidente che una nuova versione fatta da zero avrebbe avuto molto in comune con il prototipo in mano. In questi casi abbiamo scelto di rielaborarlo e risistemarlo, in particolare assicurandoci di:

- Ripensare le sue interfacce utilizzando una metodologia ispirata al TDD
- Gestire casi estremi e eccezioni
- Stabilire una modularità adeguata
- Rispettare gli standard per la buona progettazione
- Aggiungere *test coverage*, in speciale test di unità di tipo *white box*, *non regression test* e test di accetazione.

3.3 Non regression test

Questo *workflow* fece si che nessun componente prototipale rimanesse intoccato per molto tempo. In conseguenza dell'aggiunta di nuove funzionalità o di cambiamenti del codice esistente, per ad esempio gestire più casi, le componenti venivano rielaborate, ritrasformate e pulite. In sostanza, il *refactoring* di pezzi preesistenti diventò parte essenziale del processo di sviluppo, e per renderlo possibile abbiamo dato più importanza ai textitnon regression test e a una modularità e incapsulamento che ci permettessero di cambiare il codice con sufficiente confidenza.

Avere un insieme di test su cui ci fidavamo è stato essenziale per rendere possibile e veloce lo sviluppo in queste modalità. Un aspetto curioso da discutere è proprio questo: in che modo aumentare il grado di affidabilità fornito da una *suite* di test?

Un altro contesto in cui l'analisi statistica dei dati e dei file è stata essenziale è stato durante la fase di integrazione di informazioni sulle stanze e piani di ogni edificio. Alla fine della lettura e estrazione, abbiamo scoperto che non esisteva nessun standard identificativo per i piani che venisse rispettato da tutte le sorgenti. Non c'era un modo diretto per associare due piani dello stesso palazzo su sorgenti diverse, e ciò rendeva l'integrazione dell'informazione delle stanze molto difficile.

Con l'uso della REPL, abbiamo stabilito però che sul 99% dei piani era possibile trovare almeno una stanza, utilizzando il codice identificativo, in tutte e tre le

sorgenti. Sapendo in quale piano una stanza veniva rimappata in ogni sorgente ci dava una forte indicazione di come quei piani dovrebbero venir relazionati. L'unico problema erano i casi estremi, in cui due stanze di uno stesso piano su una sorgente venivano rimappate su piani diversi in un'altra sorgente, o quando per qualche piano nessuna associazione di stanza con le altre sorgenti era possibile. L'algoritmo finale che abbiamo concepito gestisce tutti i possibili casi, e se ne accorge quando non riesce a trovare un'associazione fra piani di due o più sorgenti.

In questo modo all'utente vengono segnalati i conflitti, sia quelli che abbiamo risolto in modo automatico che quelli la cui risoluzione non è possibile. Con l'uso di questa strategia e una serie di eurisitche per renderla più efficiente, siamo stati in grado di associare il 99% dei piani correttamente. L'1% che avanza è costituito da precisamente tre casi, due dei quali si presenta solo a causa di un errore di battitura dei dati originali (stanze identificate in modo sbagliato), e comunque vengono tutti e tre segnalati per la revisione dell'utente.

3.4 Refactoring guidato - SOLID

Lorem Ipsum dolor sit

"We cannot solve the problems we have created with the same thinking we used in creating them." - A. Einstein

Bibliografia

[1] M. Gotti, I linguaggi specialistici, Firenze, La Nuova Italia, 1991.

[2] R. Wellek, A. Warren, Theory of Literature , 3rd edition, New York, Harcourt, 1962.

[3] A. Canziani et al., Come comunica il teatro: dal testo alla scena. Milano, Il Formichiere, 1978.

[4] Ministry of Defence, Great Britain, Author and Subject Catalogues of the Naval Library, London, Ministry of Defence, HMSO, 1967.

[5] H. Heine, Pensieri e ghiribizzi. A cura di A. Meozzi. Lanciano, Carabba, 1923.

[6] L. Basso, "Capitalismo monopolistico e strategia operaia", Problemi del socialismo, vol. 8, n. 5, pp. 585-612, 1962.

[7] L. Avirovic, J. Dodds (a cura di), Atti del Convegno internazionale Umberto Eco, Claudio Magris. Autori e traduttori a confronto (Trieste, 27-28 novembre 1989), Udine, Campanotto, 1993.

[8] E.L. Gans, The Discovery of Illusion: Flaubert's Early Works, 1835-1837, unpublished Ph.D. Dissertation, Johns Hopkins University, 1967.

[9] R. Harrison, Bibliography of planned languages (excluding Esperanto). `http://www.vor.nu/langlab/bibliog.html`, 1992, agg. 1997.