Università degli Studi di Urbino Carlo Bo
Corsi di Laurea in Informatica Applicata
Programmazione e Modellazione a Oggetti, a.a. 2021/2022

Relazione per il Progetto del Corso di Programmazione e Modellazione a Oggetti

Sara Montagna*

Dicembre 2021

Il presente documento vi mostra quali devono essere gli aspetti affrontati nella relazione del vostro progetto, e come devono essere organizzati. Per ciascuna delle sezioni del documento sarà fornita una descrizione di ciò che ci si aspetta venga prodotto dal team di sviluppo. Il modello della relazione segue il processo tradizionale di ingegneria del software fase per fase (in maniera ovviamente semplificata). La struttura della relazione non è indicativa ma **obbligatoria**. Gli studenti dovranno produrre un documento che abbia la medesima struttura, non saranno accettati progetti la cui relazione non risponda al requisito suddetto.

1 Analisi

In questa sezione andrà fatta l'analisi dei requisiti e quella del problema, ossia verranno elencate le cose che l'applicazione dovrà fare (requisiti) e verrà descritto il dominio applicativo (analisi del problema). In fase di analisi, è molto importante tenere a mente che non vi deve essere alcun riferimento al design né tantomeno alle tecnologie implementative, ovvero, non si deve indicare come il software sarà internamente realizzato. La fase di analisi, infatti, precede qualunque azione di design o di implementazione.

1.1 Requisiti

Nell'analisi dei requisiti dell'applicazione si dovrà spiegare cosa l'applicazione dovrà fare. Non ci si deve concentrare sui particolari problemi, ma esclusivamente su cosa si desidera che l'applicazione faccia. A corredo dell'analisi dei requisiti va incluso uno **Diagramma dei Casi d'Uso**.

1.2 Modello del dominio

In questa sezione si descrive il modello del dominio applicativo, individuando le entità in gioco e in che modo sono in relazione le uno con le altre. Non vanno inserire idee circa possibili soluzioni implementative, ovvero sull'organizzazione interna del software. Infatti, la fase di analisi va effettuata prima del progetto: né il progetto né il software esistono nel momento in cui si effettua l'analisi. La discussione di aspetti propri del software (ossia, della soluzione al problema) appartengono alla sfera della progettazione, e vanno discussi successivamente.
È obbligatorio fornire uno schema UML del dominio, che diventerà anche lo scheletro della parte di modello dell'applicazione, ovvero dei suoi elementi costitutivi (in ottica MVC - Model View Controller): se l'analisi è ben fatta, dovreste ottenere una gerarchia di concetti che rappresentano le entità che compongono il problema da risolvere. Un'analisi ben svolta prima di cimentarsi con lo sviluppo rappresenta un notevole aiuto per le fasi successive: è sufficiente descrivere a parole il dominio, quindi estrarre i sostantivi utilizzati, capire il loro ruolo all'interno del problema, le relazioni che intercorrono fra loro, e reificarli in interfacce.

2 Design

In questo capitolo si spiegano le strategie messe in campo per soddisfare i requisiti identificati nell'analisi. Si parte da una visione architetturale, il cui scopo è informare il lettore di quale sia il funzionamento dell'applicativo realizzato ad alto livello. In particolare, è necessario descrivere accuratamente in che modo i componenti principali del sistema si coordinano fra loro. A seguire, si dettagliano alcune parti del design, in particolare quelle maggiormente rilevanti al fine di chiarificare la logica applicativa, ovvero la logica con cui sono stati affrontati i principali aspetti dell'applicazione e progettate le componenti principali.

2.1 Architettura

Questa sezione spiega come le componenti principali del software interagiscono fra loro. In particolare, qui va spiegato se e come è stato utilizzato il pattern architetturale model-view-controller. Se non è stato utilizzato MVC, va spiegata in maniera molto accurata l'architettura scelta, giustificandola in modo appropriato. Se è stato scelto MVC, vanno identificate con precisione le interfacce e classi che rappresentano i punti d'ingresso per modello, view, e controller. Si raccomanda di separare bene controller e model, facendo attenzione a non includere nel secondo strategie d'uso che appartengono al primo. In questa sezione vanno descritte, per ciascun componente architetturale che ruoli ricopre (due o tre ruoli al massimo), ed in che modo interagisce (ossia, scambia informazioni) con gli altri componenti dell'architettura. Si raccomanda inoltre di porre particolare attenzione al design dell'interazione fra view e controller: se ben progettato, sostituire in blocco la view non dovrebbe causare alcuna modifica nel controller (tantomeno nel model).

2.2 Design dettagliato

In questa sezione si possono approfondire alcuni elementi del design con maggior dettaglio. Mentre ci si attende principalmente (o solo) interfacce negli schemi UML delle sezioni

precedenti, in questa sezione è necessario scendere in maggior dettaglio presentando la struttura di alcune sottoparti rilevanti dell'applicazione. È molto importante che, descrivendo la soluzione ad un problema, quando possibile si mostri che non si è re-inventata la ruota ma si è applicato un design pattern noto. È assolutamente inutile, ed è anzi controproducente, descrivere classe-per-classe (o peggio ancora metodo-per-metodo) com'è fatto il vostro software: è un livello di dettaglio proprio della documentazione dell'API (deducibile dalla Javadoc). **È necessario che ciascun membro del gruppo abbia una propria sezione di design dettagliato, di cui sarà il solo responsabile**.
Si divida la sezione in sottosezioni e, per ogni aspetto di design che si vuole approfondire, si presenti:

1. una breve descrizione in linguaggio naturale del problema che si vuole risolvere, se necessario ci si può aiutare con schemi o immagini;

2. una descrizione della soluzione proposta, analizzando eventuali alternative che sono state prese in considerazione, e che descriva pro e contro della scelta fatta;

3. uno schema UML che aiuti a comprendere la soluzione sopra descritta;

4. se la soluzione è stata realizzata utilizzando uno o più pattern noti, si spieghi come questi sono reificati nel progetto (ad esempio: nel caso di Template Method, qual è il metodo template; nel caso di Strategy, quale interfaccia del progetto rappresenta la strategia, e quali sono le sue implementazioni).

La presenza di pattern di progettazione correttamente utilizzati è valutata molto positivamente. L'uso inappropriato è invece valutato negativamente.

3 Sviluppo

3.1 Testing automatizzato

Il testing automatizzato è un requisito di qualunque progetto software che si rispetti, e consente di verificare che non vi siano regressioni nelle funzionalità a fronte di aggiornamenti. Per quanto riguarda questo progetto è considerato sufficiente testare le funzionalità core, a patto che si utilizzano suite specifiche (e.g. JUnit) per il testing automatico.

3.2 Metodologia di lavoro

Ci si aspetta, leggendo questa sezione, di trovare conferma alla divisione operata nella sezione del design di dettaglio, e di capire come è stato svolto il lavoro di integrazione. Andrà realizzata una sotto-sezione separata per ciascuno studente che identifichi le porzioni di progetto sviluppate, separando quelle svolte in autonomia da quelle sviluppate in collaborazione. Diversamente dalla sezione di design, in questa è consentito elencare package/classi, se lo studente ritiene sia il modo più efficace di convogliare l'informazione. Si ricorda che l'impegno deve giustificare circa 30-40 ore di sviluppo (è normale e fisiologico che approssimativamente la metà del tempo sia impiegata in analisi e progettazione).

3.3 Note di sviluppo

Questa sezione, come quella riguardante il design dettagliato va svolta singolarmente da ogni membro del gruppo. Ciascuno dovrà mettere in evidenza eventuali particolarità del suo metodo di sviluppo, ed in particolare:

- Elencare (fare un semplice elenco per punti, non un testo!) le feature avanzate del linguaggio e dell'ecosistema Java che sono state utilizzate. Le feature di interesse sono (solo):
  - Progettazione con generici, ad esempio costruzione di nuovi tipi generici, e uso di generici bounded. Uso di classi generiche di ibreria non è considerato avanzato.
  - Uso di lambda expressions
  - Uso di Stream, di Optional o di altri costrutti funzionali
- Sviluppo di algoritmi particolarmente interessanti non forniti da alcuna libreria

Si ricorda che i pattern di design non vanno messi qui. L'uso di pattern di design (come suggerisce il nome) è un aspetto avanzato di design, non di implementazione, e non va in questa sezione.