Software Engineering
process report template

Andrea Torchi Francesco Giovanelli Giuseppe Tempesta

Alma Mater Studiorum – University of Bologna
via Venezia 52, 47023 Cesena, Italy
andrea.torchi@studio.unibo.it
francesco.giovanelli@studio.unibo.it
giuseppe.tempesta2@studio.unibo.it

1 Introduction

Questo modulo rappresenta una fotografia, in un questo istante, del processo di sviluppo del software. Avere una documentazione completa circa tutte le fasi (Analisi, Progettazione, Implementazione, Testing) è importante per far si che si possa avere la giusta documentazione e, soprattutto, il giusto flusso di sviluppo, che parte dai requisiti, fino ad arrivare ad un prodotto finale.
Ai problemi ci si può approcciare in modo olistico (top-down), o riduzionistico (bottom-up). Cosa intendiamo?
TOP-DOWN, cioè procedendo dall'alto verso il basso, quindi partendo da specifiche di alto livello, analizzando i sottosistemi ed arrivando ad una soluzione; questo approccio è in linea con una visione OLISTICA, ovvero che non si concentra sulle singole parti, ma ha una visione "più d'insieme" o di alto livello.
BOTTOM-UP, cioè procedendo dal basso verso l'alto, quindi partendo più da componenti, per poi passare alla loro sintesi. Questo approccio è in linea con una visione RIDUZIONISTA, opposta alla precedente, la quale mira a porre l'attenzione sulle singole parti di un certo sistema, piuttosto che sull'insieme intero.

2 Vision

La "Visione" è una frase che fa capire come ci si approcci alle cose, ovvero come affrontare problemi. Lo scopo dell'analisi dei requisiti è quello di capire il problema, analizzando i requisiti ed evidenziando aspetti problematici, successivamente, produrre (in modo formale) uno o più modelli che rappresentino il sistema.
Possibile affrontare un problema partendo da zero? Senza alcuna ipotesi? Molto difficile, quasi impensabile! Partire dal foglio bianco significa non avere alcuna ipotesi tecnologica (come se si partisse assolutamente privi di informazioni, non ho nulla su cui orientarmi).
La questione che adesso ci si pone è l'affrontare un problema partendo da ipotesi

tecnologiche e poi arrivando ad una soluzione utilizzando gli strumenti che si conoscono (approccio bottom-up). L'approccio giusto, in ogni caso, è decidere il più tardi possibile quale tecnologia utilizzare, poichè si vuole trovare quella giusta(la tecnologia), al momento giusto. Non si costruisce in funzione della "scatola lego" (cioè dei pezzi che hai a disposizione), ma si analizza quello che si vuol fare e, solo dopo si cerca la "scatola lego" più opportuna a per quello che si vuol realizzare.
La visione adottata in questo caso è: "Dalle tecnologie alla analisi e al progetto logico e ritorno alle tecnologie." Cosa significa?
Dopo aver capito che comunque si deve partire dalle tecnologie (e quindi non si può partire da zero, senza alcuna ipotesi tecnologica), si fanno delle ipotesi su cosa fare. Poi ci si occupa di analisi dei requisiti. Successivamente si ritorna alle tecnologie per poter dire se si ha un "abstraction-gap". Quando si può dire se si è in presenza di una cosa del genere? Se nella business logic per ogni byte se ne devono scrivere 100 per l'infrastruttura, significa che c'è un abstraction gap enorme. Quindi la tecnologia che sto utilizzando è insufficiente, o meglio, inappropriata per il mio problema.
Si parla di "tecnology-lock" se l'applicazione è strettamente contaminata dalla tecnologia, ovviamente potrebbe rappresentare un problema. Questo accade quando la scelta della tecnologia viene fatta prima rispetto le scelte di analisi/progetto; si contamina/rende strettamente dipendente il sistema finale dalla tecnologia.
Altra cosa importante da tenere a mente è: "non c'è codice senza progetto, non c'è progetto senza analisi e non c'è analisi senza requisiti".

3 Goals

L'obbiettivo principale è produrre e sviluppare software con criteri di qualità, in termini di prodotto e di processo.

4 Requirements

In questa sezione vengono riportati i requisiti, che vengono forniti dal direttamente dal cliente.

5 Requirement analysis

Durante la fase di analisi dei requisiti, svolta dall'analista dei requisiti, si analizzano dettagliatamente questi ultimi, forniti dal cliente, al fine di esplicitarli in maniera non formale, ovvero in linguaggio naturale. Il tutto viene fatto al fine di avere un quadro chiaro dei requisiti ricevuti.

5.1 Use cases

5.2 Scenarios

5.3 (Domain)model

La modellazione può essere eseguita in maniera formale o non formale. Nel primo caso si utilizza un linguaggio (UML, QActor, ecc.) tramite il quale è possibile esprimere modelli comprensibili non solo dall'uomo ma anche da una macchina. Nel caso non formale si descrivono le entità in gioco, mediante linguaggio naturale, in termini di struttura, interazione e comportamento.

5.4 Test plan

6 Problem analysis

Scopo dell'analisi del problema è la produzione di un modello formale. Cosa significa? Che può essere eseguito su una macchina. Come può essere fatto? Attraverso un linguaggio (di modellazione, eventualmente) che riesca a catturare gli aspetti fondamentali del sistema, ovvero al giusto livello di dettaglio, mettendo in primo piano aspetti rilevanti, e lasciando in "background" dettagli. Il modello è espresso in termini di STRUTTURA, INTERAZIONE e COMPORTAMENTO. Limitatamente al nostro caso, ci è fornito un linguaggio di modellazione custom, QActor, grazie al quale modelliamo il sistema.

6.1 Logic architecture

6.2 Abstraction gap

6.3 Risk analysis

7 Work plan

Ci occupiamo di scegliere l'opportuno processo produttivo, in cui ci sono due scuole in contrasto: l'agile (di cui scrum è un esempio) e model based.
Agile development consiste in lavorare in gruppi auto organizzati, multifunzionali e orientati allo stretto contatto con il cliente finale, al fine di ottenere prodotti utilizzabili in tempo rapido, di migliorare costantemente i prodotti stessi e di rimanere flessibili a ipotetici cambiamenti resisi necessari nel tempo.
Model based development impone che,prima di trattare il codice, sia necessario occuparsi della fase di modellazione. Solo in un secondo momento si passerà alla fase di sviluppo vero e proprio, tenendo opportunamente aggiornata la parte di modello.
In questa fase entra in gioco anche il testing, che è fondamentale per assicurare solidità e riddure al minimo i rischi di bug critici in fase di produzione. I test possono essere fatti non solo alla fine della costruzione del sistema, ma anche, e soprattutto, durante. Un'ulteriore aspetto da considerare è che i test possono anche essere resi automatizzati.

8 Project

La fase di progetto, svolta dal progettista, prende in input il lavoro svolto dall'analista, ovvero il modello formale che descrive il sistema e propone, a sua volta, un ulteriore modello, nel quale vengono specificati e risolti problemi lasciati in sospeso dall'analista (e.g. quale protocollo?). Anche in questo caso il sistema è specificato in termini di STRUTTURA, INTERAZIONE e COMPORTAMENTO.

8.1 Structure

8.2 Interaction

8.3 Behavior

9 Implementation

10 Testing

Molto importante la fase di testing in cui viene testato il prodotto ottenuto. Il tutto dovrebbe essere fatto il linea con la filosofia di testare tutto al fine di provare il sistema il prima possibile.
Possibile parlare di testing manuale o automatico. Nel primo caso occorrerà dare in pasto al sistema dei valori ed assicurarsi che l'output sia in linea con le aspettative. Nel secondo caso si possono utilizzare strumenti di testing automatico (e.g. JUNIT), i quali, una volta eseguiti, testano le componenti specificate, riportando in output eventuali problemi e dove essi si sono verificati.

11 Deployment

12 Maintenance

See [?] until page 11 (CMM) and pages 96-105.

13 Information about the author

Photo of the author

References