lynx group

Tech JobsDay

Il controllo della qualità del codice nei processi di sviluppo è inutile

MA ANCHE NO

Chi sono?

Filippo Testini

15 anni di esperienza nel mondo della consulenza IT

CTO di Lynx S.p.A. e Senior Solutions Architect

Alcune tecnologie su cui ho lavorato:

PL-SQL | J2EE | .NET | Mobile | Frontend HTML 5 | Cloud

...sei un riempi-buchi, insomma!

...e cosa fa un CTO, di grazia?

Filippo Testini

CTO di Lynx S.p.A.

lynx group

Cos’è la qualità del software?

Senza volerci dilungare troppo, possiamo immaginare due mondi:

La qualità analizzata **staticamente**, tipicamente da tool di analisi automatica

La qualità analizzata **dinamicamente**, tipicamente da processi interni di review del codice e/o da ulteriori tool che verificano l’interazione tra le componenti

Le analisi statiche

Esempio di metriche tipiche:

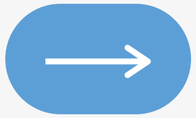
- Coverage degli Unit Test
- Indice di complessità ciclomatica sulle singole funzioni
- Rapporto tra la complessità ciclomatica totale rispetto alle righe di codice
- Quantità di bug e code smell
- Indice di duplicazione del codice
- Presenza o assenza di documentazione architetturale e tecnica

lynx group

Le analisi dinamiche

Oltre alle metriche collezionate da tool appositi, possono prevedere review del codice atte a sincerarsi che il software rispetti i pattern architetturali o talune regole di dettaglio preventivamente concordate, o verifica che tali scelte continuino a essere attuali.

Esempi di domande da farsi durante una review:



La natura monolitica di questo applicativo continua a essere valida oppure avrebbe più senso un approccio distribuito a microservizi?

La gestione dello stato applicativo tramite Redux ha effettivi benefici in termini di manutenibilità oppure converrebbe usare un altro approccio?

Ma anche, ancora più nel dettaglio: il codice ripetuto in fase di preparazione degli Unit Test ha senso di essere accentrato per facilitare futuri sviluppi?

lynx group

E se non monitoro nulla?

A lungo andare, il non monitorare e non manutenere la qualità del codice porta a:

- Difficoltà a manutenere il codice
- Complessità di lettura e comprensione
- Aumento del rischio di regressioni non gestite

Queste metriche sono facilmente monitorabili anche a livello manageriale, e al deteriorarsi sono un indicatore quasi certo di scarsa qualità nel codice.

Ma davvero è importante?

La risposta evidente dopo aver lavorato anche pochi mesi nel mondo dell'IT è…

…no.

Pochissime aziende di consulenza non hanno un processo di monitoraggio della qualità, e allora chi siamo noi per dire che si sbagliano?

Perché dovrebbe allora essere importante?

Esempio: La complessità ciclomatica

E’ il numero di condizioni contenute all'interno di un determinato blocco di codice – tipicamente una funzione o un metodo, che si traduce perciò in un insieme di ramificazioni logiche che l'algoritmo può seguire.

Prendiamo come esempio una semplice funzione:

```
function hello() {
 return "Hello World!";
}
```

Questa funzione restituisce la stringa “Hello World”, e rappresenta una prima incarnazione ideale di un nostro algoritmo, che nasce semplice e poi cresce nel tempo.

Esempio: La complessità ciclomatica

Cominciamo a complicarla un po’:

```
function helloName(name) {
     if (name !== undefined && name.length > 0 && name !== "Gianpusillipo") {
         return `Hello ${name}!`;
     }
     return "Hello World!";
}
```

La complessità ciclomatica adesso è di 4, ma tutto sommato non ci sono ancora particolari criticità.

Adesso, aggiungiamo un altro strato di complessità.

Esempio: La complessità ciclomatica

```
function helloName(name) {
        if (name !== undefined && name !== null && name.length > 100) {
                console.log("Wow, very long name, be careful!");
        } else if (name !== undefined && name !== null && name.length < 100) {
                console.log("Short name, ok.")
        }

        if (name !== undefined && name !== null && name.length > 0 && name !== "" && name !== "Gianpusillipo" && name !== "Filippo") {
                return `Hello ${name}!`;
        } else if (name === undefined) {
                return "Hello Undefined!";
        } else if (name === null) {
                return "Hello Null!";
        } else if (name.length === 0 || name === "") {
                return "Hello Empty!";
        } else if (name === "Gianpusillipo") {
                return "Oh, dear, what the heck of a name is Gianpusillipo?";
        } else if (name === "Filippo") {
                return "Welcome, master!";
        }

        return "Hello World!";
}
```

Esempio: La complessità ciclomatica

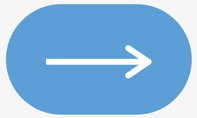
Criticità:

Complessità ciclomatica elevata (maggiore di 10)

Mancanza del rispetto del Single Responsability Principle

Difficoltà nella scrittura di Unit Test, come naturale conseguenza dei due punti precedenti

Ma si può davvero migliorare?



Esempio: La complessità ciclomatica

```
function helloNameRefactored(name) {
              manageConsoleLog(name);

              return manageName(name);
}

function manageConsoleLog(name) {
              if (name !== undefined && name !== null) {
                        if (name.length > 100) {
                                  console.log("Wow, very long name, be careful!");
                        } else {
                                  console.log("Short name, ok.")
                        }
              }

 }

function manageName(name) {
             if (name === undefined) {
                        return "Hello Undefined!";
             }
            if (name === null) {
            return "Hello Null!";
            }
            …altre condizioni…
            return `Hello ${name}!`;
}
```

Esempio: La complessità ciclomatica

Cosa abbiamo fatto?

- Scomposto in sottofunzioni
- Rispettato il SRP
- Scomponendo, normalizzata la complessità ciclomatica delle singole sottofunzioni

Sì, ok, ma perché?

Non possiamo, per come siamo programmati, affrontare problemi complessi nella loro interezza

Questo porta a una parvenza di chiarezza nell’immediato, ma a un deterioramento della comprensione anche nel brevissimo periodo

Il codice sarà incomprensibile sia a noi, sia a chiunque altro

Ok, mi hai convinto… e ora?

Quali tool per l’analisi statica?

…sceglieteli in base all’esigenza.

Quali processi per l’analisi dinamica?

…sceglieteli in base all’esigenza.

…l’importante è che le decisioni siano pensate, valutate e, quanto più possibile, condivise!

Le resistenze

Tipicamente, i più restii ad applicare un processo di monitoraggio della qualità sono i ~~manager.~~

Un manager, se sta facendo bene il suo lavoro (e ce ne sono tanti, tantissimi lì fuori) difficilmente fa resistenza a un processo che possa migliorare la manutenibilità del codice.

I tecnici, invece, non si fidano delle soluzioni facili ai problemi difficili, ma in questo caso la soluzione non è affatto facile, anzi!

lynx group

Come affrontare le resistenze

Primo esempio di resistenza: «Gli sviluppi dureranno di più»

Risposta **breve**:
non è vero.

Risposta **meno breve**:
il tempo per scrivere codice di qualità non è affatto maggiore, e anche se in taluni casi lo fosse il tempo risparmiato per la successiva evoluzione e manutenzione sarebbe, nel totale, di gran lunga inferiore.

Come affrontare le resistenze

Secondo esempio di resistenza: «Questa regola è sbagliata!»

Risposta **breve**:
non è vero.

Risposta **meno breve**:
le regole vanno analizzate, se possibile concordate (ma dipende dal contesto e dalla dimensione globale e da variabili di ampio spettro), ma poi vanno rispettate.
Nessuno sarà mai contento al 100% e tutto è sempre migliorabile, ma il processo, quando esiste, va rispettato da tutti. E non presupponiamo che, dietro una regola decisa da qualcun altro, non ci siano ragionamenti non sempre condivisibili con un team magari di 300 sviluppatori: impariamo a fidarci dell’azienda e della struttura.

Come affrontare le resistenze

Terzo esempio di resistenza: «Questo processo non può funzionare con noi!»

Risposta **breve**:
probabilmente è vero.

Risposta **meno breve**:
il processo va pensato e calato sulla specifica realtà, perciò è normale che non esista un processo che vada bene per tutti. E’ per questo che va deciso, monitorato a sua volta ed eventualmente migliorato. Ma non avere un processo non può essere meglio di avere un processo migliorabile.

E ora?

…non lasciatevi spaventare dal tema della qualità.

…cerchiamo, come tecnici, di avere un’attitudine positiva verso i problemi.

…vogliate bene a ciò che fate: è questa, la qualità!

lynx group

lynx group

Ciao 👋

👉 www.lynxspa.com

👉 Filippo Testini / Linkedin