

ANALISI PRELIMINARE DELLA MANUTENIBILITÀ DEL CODICE SVILUPPATO CON XP

Raimund Moser, Barbara Russo, Marco Scotto, Alberto Sillitti, Giancarlo Succi
Centro per l'Ingegneria del Software Applicata, Libera Università di Bolzano
[rmoser, brusso, scotto, asillitti, gsucci]@unibz.it

Uno degli obiettivi principali dell'Extreme Programming consiste nel produrre codice di alta qualità a costi più contenuti rispetto alle metodologie tradizionali. La manutenibilità del codice rappresenta uno dei punti più critici nel ciclo di vita del software ed il suo costo è di norma superiore a quello dello sviluppo iniziale. Questa analisi preliminare si focalizza sullo studio di alcune metriche di riuso estratte da un prodotto sviluppato tramite XP. I risultati di questa analisi preliminare sembrano confermare la riusabilità del codice prodotto con XP.

1. Introduzione

Una delle priorità fondamentali dell'Extreme Programming consiste nel soddisfare il cliente attraverso uno sviluppo iterativo, continua collaborazione col cliente accettando cambiamenti nei requisiti e consegnando continuamente al cliente codice contenente funzionalità importanti per lui (<http://www.agilemanifesto.org/>). Le pratiche XP sono studiate per raggiungere questi obiettivi. In particolare, le seguenti pratiche [1]:

- pianificazione iterativa ed informale
- design semplice
- refactoring continuo del codice
- programmazione a coppie
- test first
- integrazione continua

Queste pratiche sono state pensate per essere utilizzate sia durante lo sviluppo che durante la manutenzione e sembra che, almeno in parte, abbiano rispettato le promesse [11, 16]. Kent Beck afferma che lo stato di manutenzione è lo stato normale di un progetto XP [1] e comprende bene le differenze ed i problemi di un sistema in sviluppo e di uno in manutenzione. Beck afferma che lo sforzo necessario per modificare del codice che si trova già in produzione è circa il doppio di quello per un sistema sotto sviluppo e, comunque, XP non è in grado di evitare la morte entropica di un sistema software anche se può aumentarne la vita.

Fred Brooks ha affermato che il costo di manutenzione di un programma molto usato è maggiore del 40% del suo costo di sviluppo totale [3]. Questa cifra è stata anche confermata da altri studi più recenti [6]. Per questo motivo, anche in progetti XP la manutenibilità è un fattore fondamentale per il successo di un prodotto software.

La manutenibilità è una metrica di qualità del software che comprende diverse proprietà interne ed esterne di un prodotto e del suo processo di sviluppo [8]. In questo studio preliminare prendiamo in considerazione solo le proprietà interne che sono disponibili durante lo sviluppo e ne monitoriamo l'evoluzione temporale. Non prendiamo in considerazione nessuna proprietà esterna del prodotto e nessuna metrica di processo.

L'articolo è organizzato come segue: nella sezione 2 presentiamo la metodologia di ricerca ed il modello utilizzato; nella sezione 3 analizziamo un caso di studio; infine, nella sezione 4 tracciamo le conclusioni.

2. Il modello di manutenibilità

Lo scopo di questa ricerca consiste nel verificare se le pratiche XP facilitino o meno lo sviluppo di codice mantenibile. Con manutenibilità si intendono diverse caratteristiche di qualità di un prodotto software [8]. In questa ricerca consideriamo solo le proprietà interne che sono rilevanti per la manutenibilità. In particolare sono state considerate le seguenti metriche:

- Le metriche Object-Oriented di Chidamber e Kemerer (CK) [5]
- Le metriche usate per verificare la testabilità del software object-oriented [4, 13]
- Il Maintainability Index (*MI*) proposto da Oman [15]

I motivi per cui sono state scelte queste metriche sono:

1. Alcune delle metriche considerate, come le CK, sono tra quelle meglio comprese e testate
2. Esistono strumenti per raccogliere queste metriche in modo automatico e non invasivo. Questo è un requisito essenziale per raccogliere dati da un team XP senza influenzare negativamente lo sviluppo [10]

Esistono molti studi empirici che mettono in relazione le metriche CK con la qualità e manutenibilità del software:

- Li ed Henry [12] evidenziano che le metriche CK sono utili per predire la manutenibilità
- Basili *et al.* [2] hanno analizzato la relazione tra le metriche CK e la qualità del codice

I risultati di questi ed altri studi suggeriscono che la maggior parte delle metriche CK sono utili indicatori di qualità per progetti implementati con metodologie tradizionali. Questi tipi di studi non sono comuni in ambienti agili e, inoltre, non viene analizzata l'evoluzione di tali metriche durante lo sviluppo per capire se la manutenibilità migliora o peggiora nel tempo.

Un aspetto importante della manutenibilità è la testabilità. Sono stati proposti diversi modelli per misurare la testabilità di un sistema object-oriented [4] basati sulle metriche CK e sulla complessità ciclomatica di McCabe [14]. Oman *et al.*

[15] hanno proposto il Maintainability Index (M) basato sull'analisi di numerosi sistemi di Hewlett-Packard.

Due delle sei metriche CK (Number of Children e Depth of the Inheritance Tree) non sono state considerate in questo studio visto che nel progetto considerato assumono valori che sono pressoché costanti.

Per definire i valori per cui queste metriche indicano un buon livello di manutenibilità sarebbe necessario analizzare diversi progetti. Poiché al momento questo non è possibile a causa della mancanza di dati su progetti comparabili, assumiamo che un incremento delle metriche CK e della complessità ciclomatica produca codice complesso e, quindi, si riduca la manutenibilità del codice. Pensiamo che sia possibile distinguere una evoluzione del codice che non peggiora la manutenibilità da una evoluzione che la peggiora analizzando come aumentano i valori delle metriche CK e della complessità paragonate al numero di linee di codice (LOC).

Si consideri $M_i \in M = \{MCC, WMC, CBO, RFC, LCOM\}$ un sottoinsieme delle metriche che influenzano la manutenibilità sia a livello di singola classe che in media su tutte le classi del progetto. Supponiamo anche che esista una funzione f_i che restituisca il valore di M_i dato il numero di righe di codice e altri parametri P a noi ignoti al tempo t . Siccome siamo interessati solo alla dipendenza tra M_i e LOC nel tempo non facciamo nessun'altra ipotesi su f_i e possiamo scrivere la seguente equazione:

$$M_i(t) = f_i(t, LOC, P) \quad (1)$$

Questa equazione afferma che la misura di manutenibilità M_i cambia durante lo sviluppo in base a t , LOC , e un altro parametro P . Se M_i aumenta rapidamente con il LOC , la sua derivata rispetto a questa variabile è alta ed influenza in maniera negativa la manutenibilità del prodotto finale. Mentre, se la derivata di M_i rispetto a LOC è costante o negativa, la manutenibilità non peggiora anche se la dimensione del codice aumenta. Formalmente, possiamo definire il Maintainability Trend MT_i per la metrica M_i nel periodo di tempo T nel seguente modo:

$$MT_i = \frac{1}{T} \sum_{t_k} \frac{\partial f_i(t_k, LOC, P)}{\partial LOC} \approx \frac{1}{T} \sum_{t_k} \frac{\Delta M_i}{\Delta LOC}(t_k) \quad (2)$$

Per ottenere il trend generale usiamo un approccio molto semplice che verrà raffinato in futuro: calcoliamo la media delle derivate di M_i rispetto a LOC ad ogni intervallo di tempo in cui calcoliamo le metriche.

Se MT_i in una iterazione è costante o negativo l'evoluzione è considerata positiva per la manutenibilità. Invece, se MT_i è positivo l'evoluzione è considerata negativa per la manutenibilità.

L'obiettivo di questo studio consiste nel determinare se le pratiche XP producono codice mantenibile. A questo scopo le ipotesi nulle della nostra ricerca sono:

- H^1_0 : Il Maintainability Trend (MT_i) è maggiore durante le ultime iterazioni (presenta un trend di crescita durante lo sviluppo)
- H^2_0 : Il Maintainability Index (MI) è monotono decrescente durante lo sviluppo

3. Caso di studio

L'obiettivo di questo caso di studio è quello di rispondere alle domande di ricerca definite nella sezione precedente. Come prima cosa abbiamo monitorato lo sviluppo di un sistema software industriale sviluppato con XP ed abbiamo raccolto le metriche CK, la complessità ciclomatica ed il numero di righe di codice. Poi abbiamo calcolato il Maintainability Trend ed abbiamo usato un test statistico per verificare se è possibile o meno rifiutare l'ipotesi nulla.

3.1. Il progetto e la raccolta dati

Il progetto sotto analisi è un sistema software commerciale sviluppato presso VTT ad Oulu (Finlandia) in Java ed è stato un successo dal punto di vista del cliente, consegnato in tempo e senza superare il budget. Il processo di sviluppo utilizzato è una versione modificata di XP che comprende tutte le pratiche XP tranne le *metafore* e il *customer on-site* che è stato sostituito da un manager che si incontrava giornalmente con il team di sviluppo e con il cliente. Il team di sviluppo era composto da due coppie di programmatori che hanno lavorato per otto settimane. Il progetto si è concluso in cinque iterazioni: una iterazione da una settimana, tre iterazioni da due settimane e una iterazione finale di una settimana. Il sistema consiste di 30 classi Java per un totale di 1770 righe di codice.

3.2. Risultati

Nella nostra analisi abbiamo considerato i cambiamenti giornalieri delle metriche. La Figura 1 presenta un grafico dell'evoluzione della metrica ΔM_i divisa per ΔLOC .

Dalla Figura 1 è evidente che la variazione della manutenibilità rispetto alle linee di codice è circa costante durante lo sviluppo. Solo in alcuni giorni vi è stata una crescita molto sensibile di questa metrica, ciò significa che la crescita è di pari passo con quella del numero di righe di codice. Inoltre, le metriche relative al coupling e alla complessità hanno un trend decrescente e tendente a zero. Secondo noi questo è un indicatore di un buon livello di manutenibilità del prodotto finale. La metrica di coesione $LCOM$ presenta un comportamento diverso visto che ha grandi variazioni durante lo sviluppo. Comunque, numerosi studi hanno messo in discussione il significato di questa metrica [7] e il suo impatto sulla manutenibilità non è ancora chiaro.

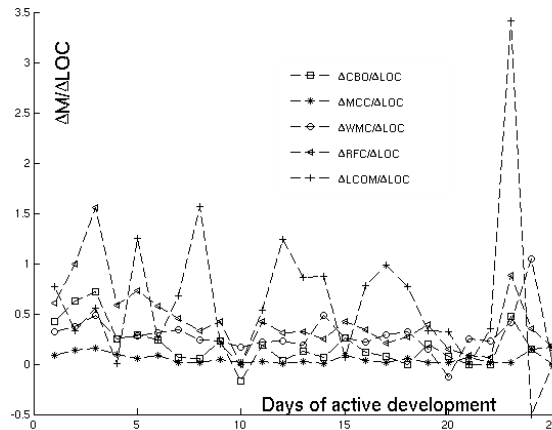


Figura 1. Evoluzione della derivata della manutenibilità rispetto alle linee di codice

Se si calcola il Maintainability Trend per ogni iterazione si ottiene un risultato paragonabile. Nelle iterazioni 2 e 4 la complessità e le metriche di coupling (*CBO*, *WMC*, *MCC*, *RFC*) crescono molto più lentamente che nelle iterazioni 1 e 3. Questi risultati sono consistenti con le attività svolte nel progetto visto che nelle iterazioni 2 e 4 una parte significativa del tempo è stata dedicata ad attività di refactoring che si suppone incrementi la manutenibilità [17].

Per verificare se il Maintainability Trend per le ultime due iterazioni sia effettivamente maggiore che nelle prime tre (prima ipotesi nulla), abbiamo usato il test di Wilcoxon [9] che ci permette di rifiutare la prima ipotesi nulla con un livello di significatività pari ad $\alpha=0.01\%$. Questo significa che nessuna delle metriche considerate cresce più velocemente di quanto non faccia il sistema impedendo al sistema di diventare più complesso e difficile da comprendere. In particolare, non è presente un incremento repentino nelle fasi finali del progetto, ma esiste un trend decrescente quando vengono aggiunte nuove funzionalità.

Per verificare la seconda ipotesi nulla, abbiamo tracciato l'evoluzione del Maintainability Index per ogni rilascio. La Figura 2 evidenzia che *MI* decresce rapidamente dal rilascio 1 al 3 ma presenta un trend diverso dal rilascio 3 al 5. All'inizio il *MI* decresce molto rapidamente ma poi si mantiene pressoché costante verso la fine del progetto. Questo può essere messo in relazione con le attività di refactoring svolte in particolare nella quarta iterazione.

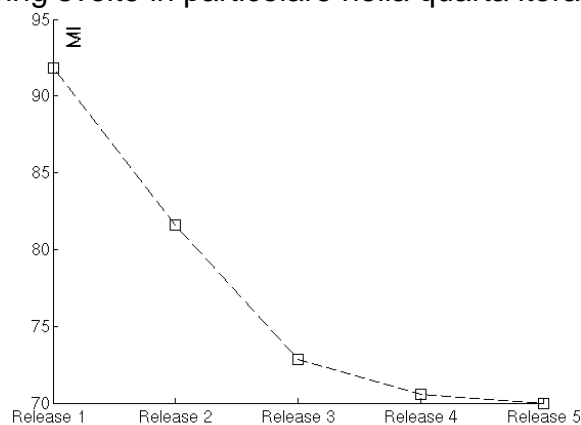


Figura 2. Evoluzione del Maintainability Index per release

Riassumendo, possiamo rifiutare l'ipotesi H^1_0 ma non la H^2_0 . Nel primo caso sembra che XP aiuti a prevenire un deterioramento del codice durante lo sviluppo a causa di un aumento eccessivo della complessità e del coupling. Nel secondo caso è necessaria un'ulteriore analisi riguardo all'applicabilità del Maintainability Index.

4. Conclusioni

Questo studio si è proposto di verificare se le pratiche XP aiutino a produrre codice mantenibile. I risultati ottenuti in questo studio preliminare sono principalmente due:

1. XP sembra favorire lo sviluppo di codice mantenibile
2. Il modello proposto per analizzare la manutenibilità può essere utilizzato per identificare anomalie nel processo di sviluppo, individuare azioni che fanno ridurre la manutenibilità del codice e rimediare tempestivamente

Ringraziamenti

Ringraziamo in particolare il team di sviluppo di VTT, ad Oulu, che si è mostrato disposto ad installare il nostro tool per la raccolta dati.

Riferimenti bibliografici

- [1] Beck K. (2004) *Extreme Programming Explained: Embrace Change*, Addison-Wesley
- [2] Basili V., Briand L., and Melo, W. L. (1996) "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, **22**(10): 267-271, October
- [3] Brooks F. (1975) *The Mythical Man-Month*, Addison-Wesley
- [4] Bruntink M., and van Deursen A. (2004) "Predicting Class Testability Using Object-Oriented Metrics," in *Proceedings of the Fourth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)*
- [5] Chidamber S., Kemerer C. F. (1994) "A metrics suite for object-oriented design," *IEEE Transactions on Software Engineering*, **20**(6): 476-493, June
- [6] Coleman D., Lowther B., and Oman P. (1995) "The Application of Software Maintainability Models in Industrial Software Systems," *Journal of Systems Software* **29**, 1 (April 1995): 3-16
- [7] Counsell S., Mendes E., Swift S. (2002) "Comprehension of object-oriented software cohesion: the empirical quagmire," *Proceedings of the 10th International Workshop on in Program Comprehension*, Paris, France, 27-29 June, Page(s): 33 - 42
- [8] Fenton N., Pfleeger S. L. (1997) *Software Metrics A Rigorous & Practical Approach*, PWS Publishing Company, Boston, pp.408
- [9] Hollander M., Wolfe D. A. (1973) *Nonparametric statistical inference*, New York: John Wiley & Sons, pp. 27-33
- [10] Johnson P. M., Kou H., Agustin J. M., Chan C., Moore C. A., Miglani J., Zhen S., Doane W. E. (2003) "Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined," *Proceedings of the 2003 International Conference on Software Engineering*, Portland, Oregon
- [11] Layman L., Williams L., Cunningham L. (2004) "Exploring Extreme Programming in Context: An Industrial Case Study," *Agile Development Conference 2004*, p. 32-41
- [12] Li W., Henry S. (1993) "Maintenance Metrics for the Object Oriented Paradigm," *Proceedings of the First International Software Metrics Symposium*, Baltimore, MD, pp. 52-60

- [13] Lo B. W. N., and Shi H. (1998) "A preliminary testability model for object-oriented software," *Proceedings of International Conference on Software Engineering: Education and Practice*, 26-29 Jan. 1998 Page(s): 330 – 337
- [14] McCabe T. (1976) "Complexity Measure," *IEEE Transactions on Software Engineering*, 2(4): 308-320, December 1976
- [15] Oman P., Hagemester J. (1994) "Constructing and Testing of Polynomials Predicting Software Maintainability," *Journal of Systems and Software* 24, 3 (March 1994): 251-266
- [16] Poole C., Murphy T., Huisman J. W., Higgins A. (2001) "Extreme Maintenance," *17th IEEE International Conference on Software Maintenance (ICSM'01)*, p. 301
- [17] Ratzinger J., Fischer M., Gall H. (2005) "Improving Evolvability through Refactoring," *Proceedings 2nd International Workshop on Mining Software Repositories, MSR'05*, Saint Louis, Missouri, USA