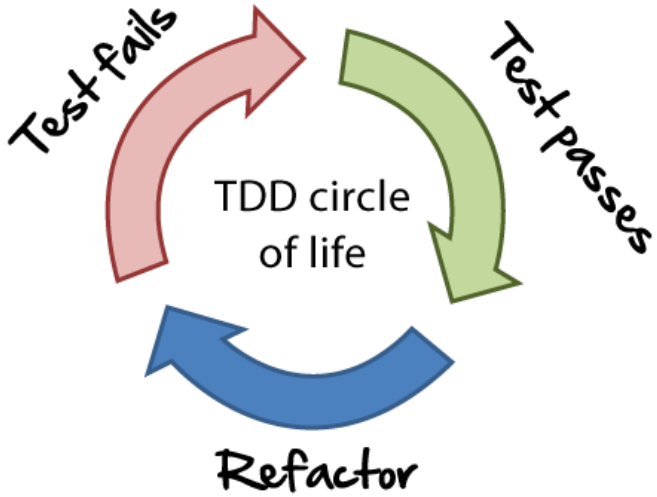
Testalo Per Piacere,
Temo Danni Devastanti

Spunti su Transformation Priority Premise & Test Driven Development

Le tre fasi del TDD (Test Driven Design)

L’evoluzione del codice segue un pattern ciclico:



- Scrivere il prossimo test (che solitamente fallisce)
- Correggere il codice per farlo passare (Transformation)
- Migliorare/semplificare il codice (Refactoring)

Refactoring e Transformation

Refactoring

- Modifica la **struttura** del codice
  - eliminazione ripetizioni
  - miglioramento leggibilità
  - estrazione metodi
  - …
- Non modifica il comportamento
- È «protetto» dai test esistenti

Refactoring e Transformation (2)

Transformation

- Rende il codice più generico
- Modifica il **comportamento** del codice
- È l’unico sistema per far passare i test

…e adesso?

Dopo il refactoring dobbiamo fare delle scelte:

- Quale test scrivo?
- Come lo faccio diventare verde?

Consigli per gli acquisti

- Quale test scrivo?
  - Scegliere il test più semplice (ma utile) in base alla complessità
  - Limitare i gradi di libertà
  - Se necessita trasformazioni troppo complicate, cercarne un altro
- Come lo faccio diventare verde?
  - Far passare il test usando l'implementazione più semplice
  - Effettuare solo modifiche indispensabili
  - Restare sulla stessa trasformazione finché possibile
  - È complicato? Esiste un test più semplice?

TPP Transformations table

#	Transformation	Start code	End code
1	{} -> nil {}	[return]	nil
2	Nil -> constant	[return] nil	[return] “1”
3	Constant -> constant+	[return] “1”	[return] “1” + “2”
4	Constant -> scalar	[return] “1” + “2”	[return] argument
5	Statement -> statements	[return] argument	[return] min(max(0, argument), 10)
6	Unconditional -> conditional	[return] argument	if(condition) [return] 1 else [return] 0
7	Scalar -> array	dog	[dog, cat]
8	Array -> container	[dog, cat]	
9	Statement -> tail recursion	a + b	a + recursion
10	If -> loop	if(condition)	loop(condition)
11	Statement -> recursion	a + recursion	recursion
12	Expression -> function	today – birth	CalculateBirthDate()
13	Variable -> mutation	day	var Day = 10; Day = 11;

TPP e TDD in Murphy style

- Red Phase: Il test fallirà, ma per il motivo sbagliato
- Green Phase: Dovrai sempre usare la transformation più onerosa
- «Forse non servirà, ma almeno è fatto» → non dovevi farlo
- «Ho già preparato alcune cose» → non dovevi farlo adesso
- «È un po' complicato, ma funziona» → Ne hai fatto un pezzo troppo grande
- Non funziona → Dovevi lasciarlo fare al prossimo navigator ☺

Conclusioni

Alcuni punti chiave:

- Evitare di introdurre cose di cui NON abbiamo (ancora) bisogno
- Scegliere la strada più semplice
- Risolvere il problema più piccolo possibile
- Non attendere per il refactoring

Bonus stage: TPP for kids

Mio figlio mi ha chiesto cosa ho imparato…

L’esempio del recinto:

Non ho animali? → non faccio nulla.

Ho un cane? → un paletto con il guinzaglio

Ho un cane feroce? → un recinto

Thanks for watching!

Sources:

- Agile technical practices distilled (P.M. Santos, M. Consolaro, A. Di Gioia)
- https://blog.cleancoder.com/
- https://it.wikipedia.org/wiki/Test_driven_development
- https://elliotchance.medium.com/the-transformation-priority-premise-tpp-3e5dc08d445e

Stefano Bellinelli - stefano.bellinelli@eoc.ch