Continuous TDD

Michał Olejnik

Politechnika Wrocławska
Informatyka

22 grudnia 2009

Agenda

Agenda

Agenda

Agenda

Agenda

Tradycyjne podejście

W klasycznych językach kompilowanych (takich jak **Pascal**, **C**, **C++** czy **Java**) moment budowy programu jest w pełni uzależniony od decyzji programisty.

Pierwsze środowiska programistyczne wspierały właśnie taki model programowanie - nie były w stanie poinformować programisty o występujących w kodzie błędach leksykalnych, do chwili, gdy on sam nie zdecydował się na przeprowadzenie weryfikacji.

Podejście to było uzasadnione w czasach, kiedy kompilacja nawet prostych programów była zadaniem wymagającym dużych zasobów i mogła trwać dość długo.

Tradycyjne podejście

W klasycznych językach kompilowanych (takich jak **Pascal**, **C**, **C++** czy **Java**) moment budowy programu jest w pełni uzależniony od decyzji programisty.

Pierwsze środowiska programistyczne wspierały właśnie taki model programowanie - nie były w stanie poinformować programisty o występujących w kodzie błędach leksykalnych, do chwili, gdy on sam nie zdecydował się na przeprowadzenie weryfikacji.

Podejście to było uzasadnione w czasach, kiedy kompilacja nawet prostych programów była zadaniem wymagającym dużych zasobów i mogła trwać dość długo.

Tradycyjne podejście

W klasycznych językach kompilowanych (takich jak **Pascal**, **C**, **C++** czy **Java**) moment budowy programu jest w pełni uzależniony od decyzji programisty.

Pierwsze środowiska programistyczne wspierały właśnie taki model programowanie - nie były w stanie poinformować programisty o występujących w kodzie błędach leksykalnych, do chwili, gdy on sam nie zdecydował się na przeprowadzenie weryfikacji.

Podejście to było uzasadnione w czasach, kiedy kompilacja nawet prostych programów była zadaniem wymagającym dużych zasobów i mogła trwać dość długo.

Continuous compilation

Wzrost mocy obliczeniowych komputerów sprawił, że główny powód stosowania tradycyjnego podejścia do kompilacji stał się nieaktualny. Dlatego we współczesnych środowiskach programistycznych stosowana jest ciągła kompilacja (continuous compilation).

Ciągła kompilacja oznacza, że IDE automatycznie kompiluje modyfikowane przez programistę klasy i dzięki temu jest w stanie niemal natychmiastowo wykryć wprowadzenie do kodu błędu składniowego i poinformować o nim użytkownika.

Test-Driven Development

Test-Driven Development to zaliczana do metodyk Agile technika wytwarzania oprogramowania. Polega ona na wielokrotnym powtarzaniu następującej sekwencji kroków:

1. Dodanie nowego testu dla nowej funkcjonalności
2. Uruchomienie testów i zobaczenie, że nowy nie udaje się
3. Implementacja nowej funkcjonalności
4. Uruchomienie testów i zobaczenie, że kończą się powodzeniem
5. Refaktoryzacja kodu

Test-Driven Development

Test-Driven Development to zaliczana do metodyk Agile technika wytwarzania oprogramowania. Polega ona na wielokrotnym powtarzaniu następującej sekwencji kroków:

1. Dodanie nowego testu dla nowej funkcjonalności
2. Uruchomienie testów i zobaczenie, że nowy nie udaje się
3. Implementacja nowej funkcjonalności
4. Uruchomienie testów i zobaczenie, że kończą się powodzeniem
5. Refaktoryzacja kodu

Test-Driven Development

Test-Driven Development to zaliczana do metodyk Agile technika wytwarzania oprogramowania. Polega ona na wielokrotnym powtarzaniu następującej sekwencji kroków:

1. Dodanie nowego testu dla nowej funkcjonalności
2. Uruchomienie testów i zobaczenie, że nowy nie udaje się
3. Implementacja nowej funkcjonalności
4. Uruchomienie testów i zobaczenie, że kończą się powodzeniem
5. Refaktoryzacja kodu

Test-Driven Development

Test-Driven Development to zaliczana do metodyk Agile technika wytwarzania oprogramowania. Polega ona na wielokrotnym powtarzaniu następującej sekwencji kroków:

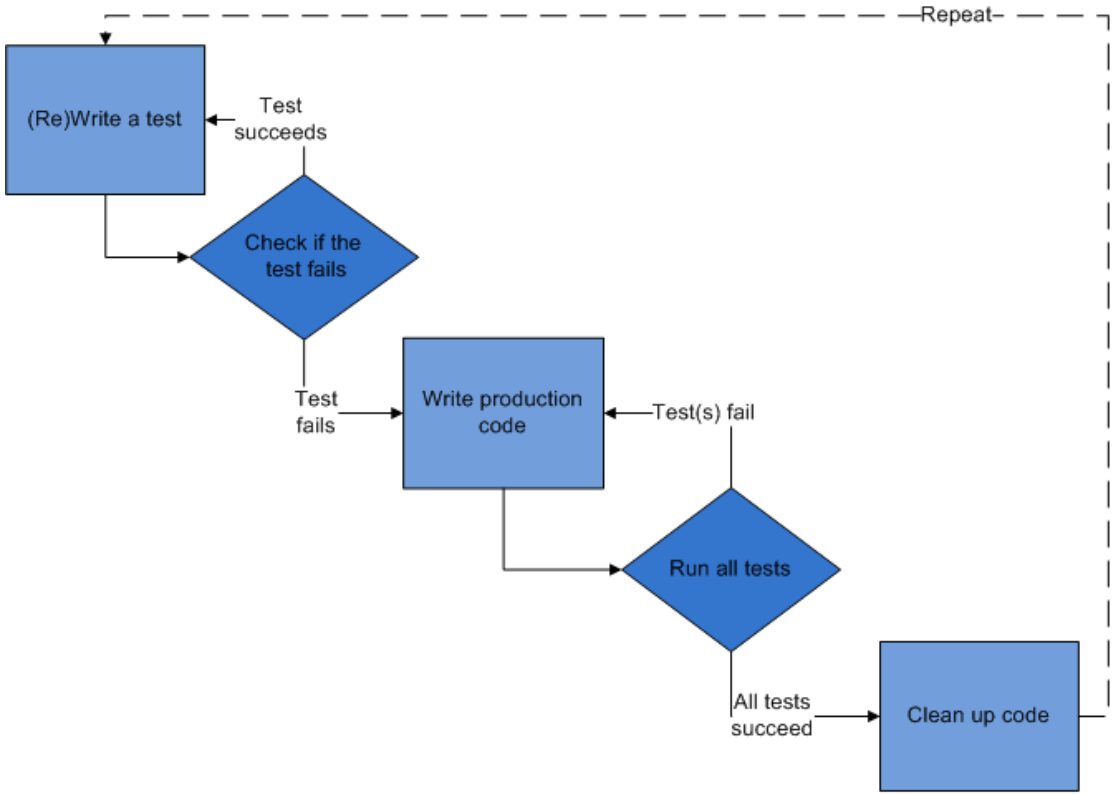
1. Dodanie nowego testu dla nowej funkcjonalności
2. Uruchomienie testów i zobaczenie, że nowy nie udaje się
3. Implementacja nowej funkcjonalności
4. Uruchomienie testów i zobaczenie, że kończą się powodzeniem
5. Refaktoryzacja kodu

Test-Driven Development

Test-Driven Development to zaliczana do metodyk Agile technika wytwarzania oprogramowania. Polega ona na wielokrotnym powtarzaniu następującej sekwencji kroków:

1. Dodanie nowego testu dla nowej funkcjonalności
2. Uruchomienie testów i zobaczenie, że nowy nie udaje się
3. Implementacja nowej funkcjonalności
4. Uruchomienie testów i zobaczenie, że kończą się powodzeniem
5. Refaktoryzacja kodu

Test-Driven Development



Cykl w TDD [7]

Continuous TDD

Continuous TDD jest wersją rozwojową tradycyjnego TDD analogiczną do usprawnienia, jakim była ciągła kompilacja w stosunku do kompilacji na żądanie. Praktyka ta polega na automatycznym uruchamianiu testów jednostkowych po każdej zmianie wprowadzonej przez programistę.

Termin Continuous TDD został po raz pierwszy użyty przez **Program Analysis Group** na uniwersytecie **MIT** w roku 2004. Przeprowadzone przez nich badania wskazały, że praktyka ta ma pozytywny efekt na efektywność i zdolność realizacji zadań u stosujących ją programistów. [1]

Zalety Continuous TDD

Do najważniejszych zalet Continuous TDD zalicza się [3]:

- Uwolnienie programisty od konieczności ręcznego uruchamiania testów
- Zwiększenie efektywności testów regresywnych poprzez zmniejszenie czasu między wprowadzeniem błędu, a jego wykryciem
- Promowanie dobrych nawyków TDD oraz pisania szybkich testów

Zalety Continuous TDD

Do najważniejszych zalet Continuous TDD zalicza się [3]:

- Uwolnienie programisty od konieczności ręcznego uruchamiania testów
- Zwiększenie efektywności testów regresywnych poprzez zmniejszenie czasu między wprowadzeniem błędu, a jego wykryciem
- Promowanie dobrych nawyków TDD oraz pisania szybkich testów

Zalety Continuous TDD

Do najważniejszych zalet Continuous TDD zalicza się [3]:

- Uwolnienie programisty od konieczności ręcznego uruchamiania testów
- Zwiększenie efektywności testów regresywnych poprzez zmniejszenie czasu między wprowadzeniem błędu, a jego wykryciem
- Promowanie dobrych nawyków TDD oraz pisania szybkich testów

Zalety Continuous TDD

Do najważniejszych zalet Continuous TDD zalicza się [3]:

- Uwolnienie programisty od konieczności ręcznego uruchamiania testów
- Zwiększenie efektywności testów regresywnych poprzez zmniejszenie czasu między wprowadzeniem błędu, a jego wykryciem
- Promowanie dobrych nawyków TDD oraz pisania szybkich testów

Narzędzia wspierające Continuous TDD

Obecnie nie ma środowiska programistycznego, które oferowałoby natywne wsparcie praktyki Continuous TDD. Istnieją jednakże wtyczki, które umożliwiają jej stosowanie. Wśród nich możemy wymienić:

- Continuous Testing Plugin - plugin stworzony przez Davida Saffa z MIT. Obecnie nierozwjiany, działa tylko w środowisku Eclipse 3.1.
- Infinitest [4]
- JUnit Max [5] - plugin nierozwijany
- ConTester [6]

Przykład

Podsumowanie

Continuous TDD jest techniką godną uwagi, ponieważ:

- zachowuje wszystkie zalety TDD
- wzbogaca TDD o kilka bardzo interesujących i przydatnych cech
- istnieje wiele narzędzi wspierających tę praktykę
- promuje wyrabianie dobrych nawyków w pisaniu testów

Bibliografia


David Saff, Michael D. Ernst: *Continuous testing in Eclipse*


http://groups.csail.mit.edu/pag/continuoustesting/


http://blog.objectmentor.com/articles/2007/09/20/


http://www.infinitest.org


http://www.threeriversinstitute.org/junitmax/subscribe.html


http://www.e-informatyka.pl/sens/


http://en.wikipedia.org/wiki/Test-driven_development