Programowanie ekstremalne

Bartłomiej Zieliński

Spis treści

1 Wstęp **2**

2 Programowanie ekstremalne w praktyce **3**
2.1 Ogólne zasady 3

3 Zalety i wady programowania ekstremalnego **6**
3.1 Zalety 6
3.2 Wady 6

4 Podsumowanie **8**

1 Wstęp

Programowanie ekstremalne (z ang. eXtreme Programming, XP) - zgodnie z teorią zamieszczoną na Wikipedii[1], *paradygmat i metodyka programowania mające na celu wydajne tworzenie małych i średnich "projektów wysokiego ryzyka", czyli takich, w których nie wiadomo do końca, co się tak naprawdę robi i jak to prawidłowo zrobić.*

O ile w przypadku tworzenia oprogramowania bardziej skomplikowanego i rozbudowanego taktyka tego rodzaju nie jest w stanie się sprawdzić (chyba, że uważnie pilnujemy większości jej aspektów lub, mówiąc krócej, nie przeholujemy z właściwą dla niej ze swobodą), co najwyżej może przynieść rozczarowanie wynikające z tego, że kod programu wciąż się rozrasta, a mimo to nie maleje ilość zadań, które program ma wykonywać - ciągle pojawiają się nowe, co wynika z braku bardziej teoretycznego przygotowania. Jeśli jednak chodzi o małe i średnie projekty, owa metodyka nierzadko może okazać się rozsądnym wyjściem.

Wyobraźmy sobie następującą sytuację: mamy do wykonania oprogramowanie wykonujące pewne konkretne obliczenia dla ośrodka badawczego, który jest naszym klientem. Priorytetem niewątpliwie będzie efektywność działania. Na rozwiązanie jednego problemu często mamy do wyboru wiele algorytmów, przy czym każdy z nich ma swoje wady i zalety. By dobrać odpowiedni, możemy zacząć żmudne przygotowania do pracy, opierające się głównie na zbieraniu potrzebnych informacji od pracowników oraz rozrysowywania przeróżnych schematów obrazujących strukturę kodu. Takie rozwiązanie jest o wiele bardziej stabilne. Rozpoczynamy pracę dopiero wtedy, gdy wiemy, co dokładnie ma zostać zrobione. Dużym minusem jest jednak czas potrzebny na uzyskanie konkretniejszych wyników, wartych zaprezentowania klientowi. Człowiek z natury jest niecierpliwy i chce jak najszybciej ujrzeć rezultaty, choćby miały być mizerne. Warto się więc zastanowić, czy nie dałoby rady zastosować opisywanej metodyki programowania ekstremalnego. Zamiast toczyć konwersacje z przyszłymi użytkownikami programu, bierzemy się od razu do pracy. Mając pewne zalążki dotyczące tego, co chcemy otrzymać jako wynik, piszemy kod, bez projektowania, co najwyżej umawiając się z naszymi współpracownikami w kwestii stosowanego nazewnictwa. Program jest rozwijany cyklicznie, a klient często może, a nawet musi spotykać się z nami i badać jego aktualne działanie, sugerując ewentualne poprawki (nierzadko duże). Zalety i wady paradygmatu zostaną opisane w dalszej części referatu, wobec czego nie będziemy się tym teraz zajmowali.

2 Programowanie ekstremalne w praktyce

2.1 Ogólne zasady

1. Utrzymywanie kontaktu z klientem.
   Jako, że już z samego założenia nie zbieramy wszystkich potrzebnych do napisania oprogramowania informacji, musimy znaleźć inny sposób, by mieć możliwość weryfikowania poprawności kodu. Dobrym wyjściem jest prezentowanie naszemu klientowi wyników pracy w pewnych niewielkich odstępach czasowych. Po każdej takiej sesji powinniśmy zyskać świeże spojrzenie na sprawę, a w tym świadomość, co ma zostać poprawione oraz jak ją poprowadzić dalej. Oczywiście ze względu na fakt, iż duża ilość klientów w kwestiach pisania oprogramowania to właściciele pewnych firm, musimy liczyć się z możliwością pracy z przedstawicielem klienta, a nie samym klientem - ktoś obeznany z tematem z wnętrza firmy powinien być zawsze (a przynajmniej bardzo często) dostępny w razie wszelkich wątpliwości.

2. Pierwsze wydanie daje obraz planowanej całości.
   Na pierwszym spotkaniu po zawarciu umowy z klientem powinniśmy pokazać mu pewien całościowy "szkic" oprogramowania. Działanie jest mało rozwinięte - do usprawniania kolejnych części dążyć będziemy w trakcie przyszłej pracy, powinny jednak istnieć już pewne najistotniejsze dla klienta funkcje.

3. Rozwój kodu poprzez dodawanie pojedynczych funkcjonalności.
   Zajmujemy się tylko jednym zagadnieniem naraz. Program jest rozwijany poprzez dodawanie kolejnych, w pełni działających funkcji, nie zaś stopniowo i całościowo.

4. Specyficzny cykl programowania.
   Najpierw piszemy program testowy dla konkretnej funkcjonalności, dopiero potem implementujemy ją, ale tylko na tyle, by udało się skompilować nasz test i go uruchomić. Ostatecznie dążymy do tego, by test nie tylko zadziałał, tj. by był poprawny składniowo, lecz by jego przejście dało poprawne wyniki. Na koniec martwimy się posprzątaniem kodu -

jego refaktoryzacją[1], podziałem na odpowiednie struktury danych etc. Testy powinny być przeprowadzane także w celu sprawdzenia poprawności wprowadzanych zmian.

5. Praca w parach.
   Opracowywanie pojedynczej funkcjonalności zajmować powinny się dwie osoby - jedna pisze kod, druga kontroluje pierwszą - zadaje pytania, sugeruje zmiany. Po określonym czasie następuje zamiana.

6. Kontakt między członkami zespołu.
   Ze względu na dość nietypową technikę tworzenia kodu należy w miarę często zwoływać narady pracującego nad nim zespołu, by ustalić pewne kwestie mogące wzbudzać wątpliwości.

7. Częste wydawanie nowych wersji i integracja.
   Odstęp pomiędzy kolejnymi wydaniami programu nie powinien przekraczać kilku miesięcy. Integracja nowo powstających funkcji z resztą kodu powinna być przeprowadzana codziennie, nawet po kilka razy.

8. Proste rozwiązania.
   Funkcje powinny spełniać swoje zadania w możliwie jak najprostszy sposób - bardzo prawdopodobne, że będzie trzeba je zmienić w najbliższym czasie, więc nie ma sensu pisać perfekcyjnego kodu. To samo tyczy się klas - staramy się tworzyć ich jak najmniej.

9. Stosowanie metaforyki w opisywaniu kodu.
   Zamiast suchych informacji możemy korzystać ze słownictwa lepiej oddającego tworzoną funkcję i zrozumiałego dla każdego człowieka, który je czyta, np. pisząc program będący elektronicznym dziennikiem dla szkoły możemy się odnosić do jego części tak, jakby był papierowym dziennikiem, nie musimy nazywać tabelki reprezentującej oceny jako "obiekt typu DataGridView", a po prostu "tabelka z ocenami".

10. "Gra w planowanie".
    Klient decyduje o tym, co program ma robić, które z tych rzeczy są najważniejsze oraz kiedy powinny pojawiać się nowe wersje. Zadaniem

[1] Refaktoryzacja - podział kodu na mniejsze fragmenty w celu zmniejszenia objętości, dopasowania go do ogólnie przyjętych obecnie standardów oraz ograniczenia powtarzalności.

twórcy jest natomiast oszacowanie, co będzie potrzebne do napisania aplikacji od strony technicznej ze świadomością, dlaczego wybór padł na to a nie coś innego oraz opracowanie harmonogramu jego tworzenia dla kolejnego wydania.

11. 40-godzinny tydzień pracy i brak nadgodzin.
    Programiści nie mogą narzekać na to, że nie mają czasu, by coś zrobić. Kod tworzony zbyt szybko najczęściej bywa absolutnie nieoptymalny i nierzadko zawiera więcej błędów. Nie stosuje się też nadgodzin, dzięki czemu pracownicy rzadziej ulegają przemęczeniu.

12. Pożywienie.
    Jako, iż programiści pracują w zasadzie bez przerwy, jedzenie oraz picie powinny znajdować się w pomieszczeniu, w którym pracują. Dzięki temu mogą dłużej siedzieć przy stanowisku i stworzyć więcej kodu (na szczęście nikt nie wpadł na pomysł, by podstawiać im także wiaderka, by nie musieli tracić czasu na korzystanie z toalety).

3 Zalety i wady programowania ekstremalnego

3.1 Zalety

- możliwość szybkiego reagowania na nowe technologie, a co za tym idzie, wprowadzania ich na bieżąco,
- duża swoboda dla programistów, dzięki czemu mają oni okazję wykazać się własną twórczością i doświadczeniem, zamiast zapisywać w formie kodu już opracowane plany i wcześniej ustalone schematy,
- błędy są na ogół wypatrywane wcześniej, o wiele łatwiej dzięki temu je naprawić bez wpływu na resztę kodu,
- możliwość sprawdzania efektywności różnych pomysłów od razu na poziomie praktycznym,
- nie trzeba tworzyć specyfikacji systemu, liczą się tylko kod i testy,
- nie ma potrzeby pisania dokumentacji - zamiast tego umieszczamy komentarze w kodzie oraz opisujemy testy.

3.2 Wady

- ciągła ewolucja kodu oraz sama technika prowadzona ze zbyt dużą swobodą może prowadzić do mało optymalnego kodu, tak pod względem działania, jak i estetyki,
- mogą wystąpić mniej lub bardziej poważne komplikacje wywołane nieporozumieniami w kwestii oczekiwań klienta, a tym, co właściwie powstaje - zagrożenie to jest większe, jeśli programiści nie posiadają zbyt dużej wiedzy w tematyce tworzonej aplikacji,
- programiści są różnymi ludźmi, niektórzy mogą sobie nie radzić z pisaniem bez uprzedniego przygotowania, lub po prostu woleć poczynić najpierw jakieś plany, problemem może być też brak zaangażowania nawet jednej osoby,
- kod jest wspólny - oznacza to, że każdy może wprowadzić zmiany w dowolnym jego fragmencie.

4 Podsumowanie

Programowanie ekstremalne niewątpliwie znajduje zastosowanie w wielu przypadkach - nie wymaga większego planowania, klient ma okazję widzieć na własne oczy postępy w tworzeniu aplikacji, a programiści mogą pisać "po swojemu". Z reguły jest o wiele bardziej ryzykowne od schematów tradycyjnych, gdzie planowanie stoi na pierwszym miejscu. Niebezpieczeństwo problemów w trakcie tworzenia maleje tym bardziej, im biorący w projekcie udział programiści wiedzą więcej o swoim fachu. Paradygmat programowania ekstremalnego jest częściowo stosowany przez większość początkujących programistów, choć nie zdają sobie z tego z reguły sprawy - to już moje osobiste spostrzeżenie, także z własnych pierwszych kroków. Głównym mankamentem w takich przypadkach jest błędne przekonanie, że funkcje w programie tworzone są od podstaw do końca i dopiero potem kompilowane. Nie odzwierciedla to wprawdzie założeń opisywanej techniki, w której testowanie jest rzeczą ogromnej wagi, ale tym samym pokazuje, że niewłaściwe jego stosowanie nie prowadzi w zasadzie dokądkolwiek. Tworzenie kodu bez planowania wymaga większego wysiłku i koncentracji oraz doświadczenia w jego pisaniu.

Literatura

[1] http://pl.wikipedia.org/wiki/Programowanie_ekstremalne

[2] http://zasoby.open.agh.edu.pl/~10sdczerner/page/programowanie_ekstremalne_XP

[3] http://www.jera.com/techinfo/xpfaq.html

[4] http://www.governica.com/Programowanie_ekstremalne