Ze świata USOS. Część 8 – Leniwy programista, czyli co może za nas zrobić komputer

*Michał KURZYDŁOWSKI**

*Instytut Informatyki, Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski

Nikt nie lubi sprawdzania swojej pracy, prawda? Udało nam się rozwiązać zadanie, bo wpadliśmy na pomysł i potrafiliśmy go zrealizować. Podobnie programista często potrafi napisać cały kod potrzebny do wykonania zadania, nim go choć raz uruchomi, by sprawdzić, czy program robi to, co było zamierzone. W taki „wir pracy" każdy z nas nieraz wpadł. W końcu właśnie w tym czujemy się najlepiej – w rozwiązywaniu problemów.

Z przykrością jednak zauważamy, że nazbyt często taki tryb pracy doprowadza nas do złych rozwiązań. Stawia nas przy tym w sytuacji, w której ciężko jest znaleźć odpowiednią drogę. Nie jest jasne, co z obecnego rozwiązania jest prawidłowe, a co nie. Nie pozostawiliśmy za sobą bowiem żadnych wskazówek ani innego rodzaju kamieni milowych wyznaczających małe sukcesy na drodze do celu.

Chciałbym w tym artykule podzielić się z Czytelnikiem doświadczeniem, jakiego nabyliśmy my, programiści USOS, szukając lepszego sposobu rozwiązywania powierzonych nam zadań. Mam nadzieję pokazać, że warto być leniwym i odwlekać wykonanie pracy na później.

Zdefiniuj problem testami. Najczęściej działamy pod wpływem impulsu. Jeśli ktoś zada nam pytanie lub postawi przed nami problem, to większość z nas zapewne podsunie mu niemal natychmiast odpowiedź. Często będzie to pierwszy pomysł, jaki przyjdzie nam do głowy. Czasem będzie to wynik głębszego przemyślenia, ale pewno przeoczymy jakieś szczegóły. Wydaje się nam, że rozwiązaliśmy zadanie, ale to właśnie te szczegóły, które w pośpiechu pominęliśmy, powodują, iż nasza odpowiedź jest błędna. Nie inaczej postępuje nieuważny programista, który także woli przejść od razu do swojego ulubionego zajęcia – programowania.

Spróbujmy jednak zatrzymać się na chwilę i skupić na dokładnym zdefiniowaniu problemu. Podczas rozwiązywania zadania matematycznego użyjemy w tym celu kartki lub tablicy. Zapiszemy to, co wiemy, i oznaczymy to, czego szukamy. Będziemy przekształcali opis słowny problemu na zapis z użyciem symboli matematycznych. Taka postać jest dla nas wygodniejsza, bo jest pozbawiona zbędnego „szumu", a przy tym nie pomija żadnego istotnego szczegółu.

Jeśli, przykładowo, dostaniemy zadanie, by znaleźć funkcję liniową, która przechodzi przez dwa określone punkty, to możemy sformułować je następująco:

$$f(x_1) = y_1,$$
$$f(x_2) = y_2,$$
$$f(x) = ax + b.$$

Powyżej zapisaliśmy trzy „ograniczenia", jakim musi podlegać szukana funkcja f. Spójrzmy na nie jak na testy, które dla zadanej funkcji f dostarczą nam informację o tym, czy owa funkcja spełnia przyjęte założenie. Sprowadziliśmy zatem nasze zadanie do poszukiwania funkcji, która spełnia wszystkie te warunki. Podobny formalizm możemy zastosować podczas programowania, ale o tym za chwilę.

Pracowity czy leniwy? Załóżmy, iż został nam zgłoszony błąd na stronie, który uniemożliwia nauczycielowi edycję ocen studentów z pewnego przedmiotu. Pierwszym krokiem jest próba powtórzenia opisanego problemu. Warunki początkowe, takie jak oceny studentów, informacje o tychże studentach, dane o przedmiocie, wraz z listą działań, które podejmuje nauczyciel, stanowią opis testu. Test jest naszym narzędziem, które pozwala nam wyznaczyć moment, w którym kod został już naprawiony.

Pracowity programista nie potrzebuje nic więcej. Jest w stanie modyfikować kolejne fragmenty kodu, powtarzając co pewien czas test, aż do momentu, kiedy

zostanie on spełniony. Szybko jednak odkrywa, że najnudniejszym i dającym najmniej satysfakcji zajęciem jest owo powtarzanie testu, gdyż sprowadza się do wykonywania po wielekroć tych samych czynności. Rozwiązanie, które najczęściej znajduje w tym przypadku, to odwlekanie testowania albo zawężanie jego zakresu do absolutnego minimum. Pierwszy wybór powoduje, że błędy w kodzie odkrywamy dopiero po czasie. Drugi doprowadza do kodu, który tylko pozornie wydaje się poprawny, nie spełnia jednak bardziej szczegółowych testów.

Lenistwo w tym przypadku może popłacać, bo komputer okazuje się doskonałym narzędziem do wykonywania za nas wszelkich czynności, które potrafimy opisać czy, ujmując inaczej, zautomatyzować. Taką właśnie czynnością jest testowanie. Przy odrobinie pracy test zapisany na kartce można przekształcić w kod, który będzie mógł być uruchamiany wedle potrzeby programisty. Nie potrzeba tym samym kompromisu między testowaniem a programowaniem. Programista może zająć się tym drugim, gdy komputer wyręcza go w tym pierwszym.

Testowania różne smaki. Wiemy już, że pierwszą czynnością programisty będzie powtórzenie błędu. Kolejną zaś jego zapisanie w postaci testu, który będzie potrafił wykonać za niego komputer. Nie jest jednak oczywiste, jak ów test zapisać. Wróćmy do przykładu z niezapisującymi się ocenami. Odtworzyliśmy problem, używając przeglądarki. Wystarczyłoby zatem zapamiętać stan bazy danych, konfiguracji aplikacji i nagrać czynności, które wykonaliśmy w przeglądarce. To jednak cała masa informacji, które muszą być zapisane i odtworzone przed każdym wykonaniem testu.

Opłaca się wykonać więcej analiz, nim przejdziemy do zapisania testu. W końcu i tak będziemy analizować problem, by znaleźć rozwiązanie. Po zastanowieniu możemy dojść do wniosku, że istotne są tylko niektóre informacje związane z przedmiotem i studentami. Możemy też, przykładowo, zauważyć, że błąd występuje jedynie dla ocen oznaczających niezaliczenie. Tego typu spostrzeżenia zmniejszą opis problemu i sprawią, że będzie czytelniejszy.

Taki test jest jednak nadal „ciężki", choć także „pewny". Ciężki ze względu na ilość danych i pracy, która musi zostać wykonana przy każdym uruchomieniu testu – musimy bowiem załadować dane do bazy, uruchomić przeglądarkę i wykonać w niej zapisane akcje. Pewny, gdyż jego spełnienie daje spore gwarancje rozwiązania problemu. Trzeba się jednak zawsze liczyć z tym, że nasze dane lub sposób testowania nie zawierają jakiegoś istotnego szczegółu.

Istnieje jeszcze jeden prócz czasochłonności mankament tego rodzaju testu. Jego działanie często zależy od danych niepowiązanych bezpośrednio z problemem, czy od struktury stron internetowych i interakcji między nimi. Źle zapisany test mógłby np. wymagać tego, by guzik do edycji ocen był w określonym miejscu na stronie. Trzeba mieć na uwadze, że aplikacja będzie musiała ulegać modyfikacjom i to nie tylko ze względu na zmieniające się wymogi, czy też nowe funkcjonalności, ale i trendy w internecie. Nie chcemy, by oznaczało to także konieczność edycji testów niezwiązanych bezpośrednio z tymi zmianami.

Lżej znaczy lepiej? Na etapie pisania testu często wiemy już, co wymaga naprawy. Ta wiedza pomaga nam doprowadzić test do postaci, która najczytelniej definiuje problem. Możemy jednak jeszcze bardziej „odchudzić" nasze testy, zmieniając technikę testowania. Zapomnijmy zatem o przeglądarce i potraktujmy ją jako narzędzie tłumaczące operacje użytkownika na żądania realizowane przez aplikację.

Opis żądania i format odpowiedzi na nie stanowią pewnego rodzaju API (*Application Programming Interface*), które w dobrze zaprojektowanej aplikacji rzadko ulega zmianie. Korzystając z takiego API, możemy zdefiniować nasz problem, opisując żądanie i spodziewaną odpowiedź. Taki test ma mniejszy opis i może zostać szybciej wykonany przez komputer, gdyż nie wymaga od niego użycia przeglądarki lub też symulacji jej działania.

Czy da się zejść jeszcze „niżej", tj. w głąb naszej aplikacji? API specyfikuje format danych wejściowych, wyjściowych i zachowanie. Sposób implementacji takiego API jest w gestii programisty, który zazwyczaj ułatwia sobie pracę i nadaje kodowi czytelną strukturę, używając do tego zrębów aplikacji (ang. *framework*). Wspominam o nich, gdyż stanowią one szkielet, który powoduje, że łatwiej znaleźć w kodzie pewną powtarzającą się strukturę, która nie ulega zbyt częstym zmianom. Takim schematem może być np. zapisywanie logiki aplikacji w postaci metod przypisanych pewnym obiektom. Brzmi abstrakcyjnie? Spójrzmy na przykład.

Ocena jest pewnym obiektem w aplikacji, dla którego może istnieć metoda pozwalająca na jej edycję. Przy takiej implementacji możemy sprowadzić nasz problem do wywołania owej metody i oczekiwania, że zakończy się ona poprawnie. Zmniejszyliśmy tym samym poziom abstrakcji definicji naszego problemu, redukując tym samym opis testu. Uzależniliśmy się jednak równocześnie od struktury aplikacji. Podejmowanie decyzji o wyborze konkretnego rodzaju testu wymaga zatem wzięcia pod uwagę wielu czynników. Trzeba zbadać, które narzędzie jest najbardziej odpowiednie do danego zadania.

Lenistwo popłaca? Widzimy zatem, że analiza problemu, dobór testu i jego implementacja pochłaniają dużą część czasu programisty. Moglibyśmy o takim pracowniku powiedzieć, że jest leniwy. Jego powinnością jest pisanie kodu, z którego będzie się składać aplikacja, a on tymczasem odkłada ten obowiązek na później. Owszem, pisze kod, ale buduje z niego testy. One zaś nie dostarczają bezpośredniej wartości. Programista będzie twierdził, że poprawia w ten sposób jakość aplikacji, którą pisze, ale czy będzie w stanie potwierdzić swoje zapewnienia? Sami oceńcie, czy warto być leniwym. Może i wy, podejmując się kolejnego zadania, spróbujecie odłożyć „pracę" na później?