Testowanie elementów programowalnych w systemie informatycznym

Marek Żukowicz

10 października 2017

Streszczenie

W literaturze istnieje wiele modeli wytwarzania oprogramowania oraz wiele strategii testowania oprogramowania. Kilka razy w roku organizowane są konferencje, spotkania, prezentacje czy zawody, podczas których testerzy mogą wykazać się pomysłowością, przebiegłością, umiejętnością analitycznego rozwiązywania problemów itd. Mimo to istnieje pewien rodzaj testów, na który należy zwrócić uwagę, a mianowicie testowanie elementów programowalnych w systemie informatycznym.

1 Wprowadzenie do problemu

W praktyce może zaistnieć taka sytuacja, że wartość pewnego pola lub wartości pól na formularzu nie będą obliczone wzorem, nie będą ciągiem liczbowym czy ciągiem znaków wpisanym przez użytkownika, lecz będą wyznaczane za pomocą pewnych warunków logicznych lub połączonych relacji, zagnieżdżonych lub zależnych od wartości innych pól. Takim przykładem może być dowolny sklep internetowy, który posiada pola z najnowszymi produktami. W przypadku sklepu internetowego z ubraniami:

1. Jeśli użytkownikiem jest kobieta, to należy wyświetlić ubrania damskie w sekcji najnowsze produkty,
2. W przeciwnym przypadku należy pokazać ubrania męskie.

Ale kobiety oraz mężczyźni mogą być w różnym wieku, wobec tego będą zwracać uwagę na inne ubrania. Dlatego też aplikacja powinna brać pod uwagę wiek zalogowanej osoby. W przypadku ubrań ważna jest również moda albo te produkty, które są najnowsze.

Problem polega na tym, że czasami nie da się "na sztywno" zaimplementować pól w sekcji, wymagalności pól lub wartości czy opcji w polach formularza. Wobec tego muszą one być sterowane wartościami z innych pól w testowanym systemie, ale tak, jak oczekuje tego użytkownik (musi być w

aplikacji zaimplementowana możliwość definiowana własnych relacji). Jest to rodzaj funkcjonalności, na który z pewnością należy zwrócić uwagę podczas testowania. Sklep z ubraniami to prosty przykład, jednak istnieją systemy np. aplikacje finansowe, w których zaimplementowana jest silnie złożona kalkulacja, gdzie pewne wartości są sterowane za pomocą innych wartości zdefiniowanych przez użytkownika - a takie testy już wymagają cierpliwości oraz wykonania wielu przypadków testowych.

2 Testowanie sterowanych sekcji oraz pól

Podstawowymi atrybutami pola na formularzu testowanej aplikacji, którego zachowanie może być sterowane w pewien sposób, są:

- widoczność pola,
- wymagalność pola,
- wartość w polu.

Klasycznym przykładem widoczności sekcji oraz zawartych w niej pól jest zaznaczenie opcji w aplikacji, że dana osoba czy kontrahent posiada inny adres korespondencyjny niż adres zamieszkania. Jeśli dodatowo sekcja z adresem korespondencyjnym jest sterowana, czyli taka, że użytkownik (administrator) może definiować pola wraz z ich dziedziną, to podczas testów należy wykazać się dość dużą kreatywnością.

Generalnie, w przypadku testowania pól sterowanych przez użytkownika, należy zwrócić uwagę na następujące atrybuty:

1. Czy wszystkie pola zdefiniowane pojawiają się na formularzu,
2. Czy prawidłowo działa walidacja zdefiniowana przez użytkownika dla pola,
3. Czy prawidłowo działa wymagalność wartości,
4. Czy prawidłowo działają pola z listą rozwijalną (czy pojawiają się w niej wszystkie zdefiniowane opcje do wyboru),
5. Czy prawidłowo zapisują się wartości w polach z opcją wielu wartości do wyboru,
6. Czy wartości domyślne podpowiadają się tak, jak są zdefiniowane,
7. Czy zdefiniowane pola pojawiają się w dobrym miejscu,
8. Czy prawidłowo działają zdefiniowane relacje między polami na formularzach definiowanych przez użytkownika (np. jeśli wybierzemy opcję samochód używany, wówczas pole przebieg powinno stać się polem obligatoryjnym oraz większym od 0).

3 Matematyczny model reguł sterujących

Reguły sterujące mogą być w aplikacji zaimplementowane w różny sposób, ale tak naprawdę sprowadza się to do utworzenia przez użytkownika reguł wykorzystujących podstawowe funktory logiczne, znane jeszcze ze szkoły średniej, np. alternatywa, koniunkcja, negacja, implikacja. Takie wyrażenia mogą być definiowane w systemie za pomocą wzoru. Przykładowo, jeżeli wartość pola P jest sterowana innymi polami, to ogólna postać takiego problemu ma postać:

$$P = F(x_1, x_2, ..., x_n),$$

gdzie $x_1 \in X_1, x_2 \in X_2, ..., x_n \in X_n, n \in \mathbb{N}$. Podczas definiowania reguł sterujących wartością pewnego pola zbiory $X_1, X_2, ..., X_n$ najczęściej można przedstawić w postaci:

- $X_1 = \bigcup_{i=1}^{n} X_{1i},\ X_{1p} \cap X_{1s} = \emptyset,$
- $X_2 \bigcup_{i=1}^{k} X_{2i},\ X_{2p} \cap X_{2s} = \emptyset,$
- ...,
- $X_m = \bigcup_{i=1}^{m} X_{mi},\ X_{mp} \cap X_{ms} = \emptyset,$

gdzie $n, p, s, m, k \in \mathbb{N}, p \neq s$.

Taki zapis jak wyżej wynika z tego, że pewna ilość elementów (zmiennych) danego zbioru analogicznie oddziałuje na pole, którego wartość jest sterowana (tak samo jak znana ze szkoły średniej funkcja, która ma kilka wzorów na różnych przedziałach dla różnych argumentów).

Przykład 1 (model abstrakcyjny)

Załóżmy, że istnieje aplikacja, która posiada taką funkcjonalność, że użytkownik może sterować zachowaniem pól na formularzu lub na sekcji formularza. Załóżmy również, że na potrzeby testów chcemy zdefiniować dla pewnego formularza pola A, B, C powiązane następującymi relacjami:

1. Dziedziną pola A są liczby całkowite z przedziału $[-1000, 1000]$, pole A jest polem obowiązkowym.
2. Jeśli pole A ma wartości **niedodatnie**, to pole B jest polem **nieobowiązkowym**, dziedziną pola B są liczby całkowite z przedziału $[0, 9]$.
3. Jeśli pole A ma wartości **dodatnie**, to pole B jest polem **obowiązkowym**, wówczas dziedziną pola B są liczby całkowite z przedziału $[0, 20]$.
4. Jeśli pole B ma wartości dodatnie, które:

- są większe od 10,
- są mniejsze od 16,
- jest polem obowiązkowym,

to pojawia się pole C, które nie jest obligatoryjne. Dziedziną pola C jest tekst zmiennej długości do 25 znaków.

5. Jeśli pole B ma następujące własności:

   - wartości pola są większe lub równe 16,
   - jest polem obowiązkowym,

   to pojawia się pole C, które jest obligatoryjne. Wartość pola C należy wybrać z listy rozwijalnej, dodatkowo pole C umożliwia wyszukiwanie wartości po początkowych znakach.

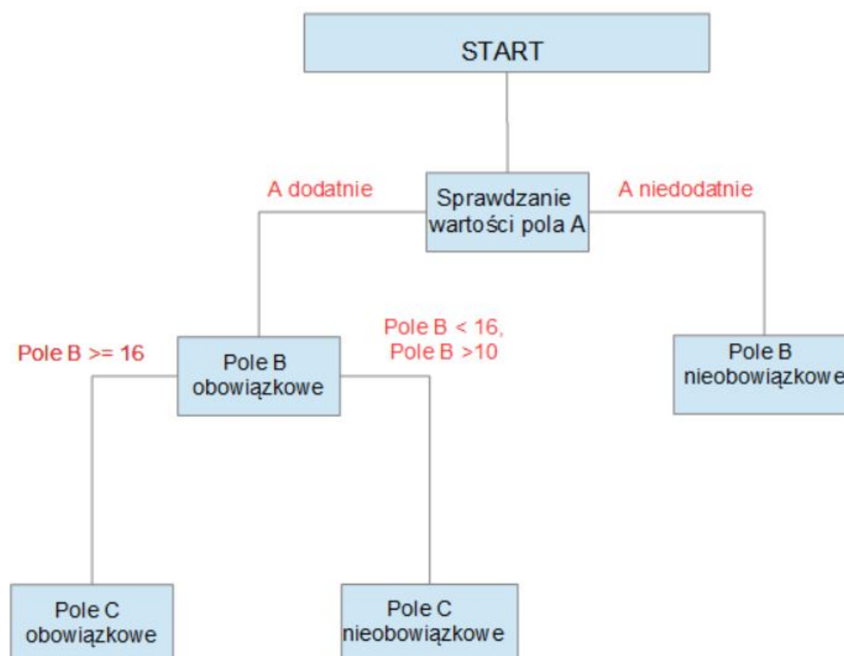
Po utworzeniu takich relacji pomiędzy polami należy sprawdzić poprawność ich działania, a co za tym idzie, poprawność mechanizmu pozwalającego tworzyć zależności pomiędzy obiektami. Ciekawą cechą takiej funkcjonalności jest to, że najpierw należy ją po prostu ***"zaprogramować"*** i jest to rola testera. Programista w rzeczywistości tworzy pewnego rodzaju ***"framework"*** umożliwiający ustawienie zachowania się wartości w polach.

Testowanie wyżej przedstawionej funkcjonalności wymaga wielu testów i należy wybrać pewną optymalną drogę. Punkty od 1-5 opisują tylko jedną konkretną sytuację, którą ma zaprogramować tester oraz sprawdzić działanie tego, co zaprogramował. Takich sytuacji można oczywiście utworzyć nieskończenie wiele, jeśli aplikacja posiada mechanizm umożliwiający definiowanie reguł (czyli sterowanie zachowaniem wartości w obiektach). W takim przypadku sprawdzą się metody: *pairwise testing* (ewentualnie *n-wise testing*), analiza wartości brzegowych. W literaturze istnieją artykuły prezentujące te dwie wspomniane strategie testów, dlatego w artykule nie będziemy się na nich skupiać. Testy wyżej opisanego problemu można zaprojektować na kilka sposobów, np.:

- narysować graf z warunkami logicznymi, rozgałęzienia oraz ścieżki główne,
- w formie tabelki,
- analizować krok po kroku.

Do tego rodzaju problemu można zastosować graf, a na jego podstawie zrobić analizę testów, do której można wykorzystać *analizę wartości brzegowych.* Przykładowo, graf oraz analiza krok po kroku mogą mieć postać:

Przypadek 1



Rysunek 1: Logika działania pól - opracowanie własne

Krok 1:
Pole A ma wartość większą nie większą niż 0*.*
Rezultat:

1. *Pole B nie jest obowiązkowe,*
2. *Dziedziną pola B są liczby całkowite z przedziału* $[0, 9]$.

Przypadek 2

Krok 1:
Pole A ma wartość większą od 0*.*
Rezultat:

1. *Pole B jest obowiązkowe,*
2. *Dziedziną pola B są liczby całkowite z przedziału* $[0, 20]$.

Krok 2:
Pole B ma wartość całkowitą z przedziału $[11, 15]$.
Rezultat:

1. *Pojawia się pole C,*
2. *Pole C nie jest obowiązkowe,*
3. *Dziedziną pola C jest tekst o maksymalnej długości* 25 *znaków.*

Przypadek 3

Krok 1:
Pole A ma wartość większą od 0*.*
Rezultat:

1. *Pole B jest obowiązkowe,*
2. *Dziedziną pola B są liczby całkowite z przedziału* $[0, 20]$.

Krok 2:
Pole B ma wartość całkowitą z przedziału $[16, 20]$.
Rezultat:

1. *Pojawia się pole C,*
2. *Pole C jest obowiązkowe,*
3. *Pole C pobiera wartości z listy rozwijalnej,*
4. *Pole C umożliwia wyszukiwanie wartości po początkowych znakach.*

W taki sposób można wypisać wszystkie główne ścieżki. Jednak ilość testów może znacznie przekroczyć liczbę głównych ścieżek, ponieważ można testować negatywnie, stosować analizę wartości brzegowych, wpisywać tekst zamiast liczb, ułamki itp.

Przykład 2

Kolejny przykład dotyczy branży finansowej. Załóżmy, że pewien bank udziela kredyty dla firm, a w niektórych przypadkach wymaga ubezpieczenia kredytu. Warunkami wstępnymi, które należy uwzględnić podczas podejmowania decyzji o konieczności ubezpieczenia kredytu, są:

1. Forma prawna,
2. Dochód z ostatnich 12 miesięcy,
3. Wiek właściciela firmy.

Zadaniem programisty jest utworzenie takiej funkcjonalności, która umożliwi użytkownikowi tworzyć dowolną ilość pól na formularzu, które muszą posiadać następujące własności:

- Pola **Forma prawna**, **Dochód z ostatnich 12 miesięcy** są obowiązkowe,
- Pole Forma prawna ma do wyboru zdefiniowany przez użytkownika zbiór nazw,
- Pole Dochód musi być nieujemne.
- Pole **Wiek właściciela firmy** ma wartości całkowite z przedziału $[18, 70]$, nie musi być obowiązkowe i widoczne w każdym przypadku.
- Zdefiniowane pola powinny mieć taką własność, która umożliwi uzależnienie ich wymagalności oraz wartości od pól Forma prawna, Dochód z ostatnich 12 miesięcy, Wiek właściciela firmy oraz innych zdefiniowanych pól.

Takie postępowanie ma sens, ponieważ istnieje wiele form prawnych, wobec czego napisanie warunków na "sztywno" w kodzie, mogłoby być słabo przydatne w przypadku potrzeby zmian.

Załóżmy, że osoba testująca chce użyć w celu sprawdzenia aplikacji dwie formy prawne: działalność jednoosobowa, spółka z o.o. Jeśli formą prawną jest działalność jednoosobowa, to:

- wymagany jest wiek właściciela,
- w zależności od wieku wymagane jest wyższe lub niższe ubezpieczenie.

Jeśli formą prawną jest działalność spółka z o.o, to:

- opcjonalny jest wiek właściciela,
- w zależności od dochodu wymagane jest wyższe lub niższe ubezpieczenie.

Jeśli formą prawną jest spółka z o.o. oraz dana osoba prowadzi własną jednoosobową działalność, to wyznaczane jest ubezpieczenie dla obu form prawnych osobno, a następnie brane niższe.

Na potrzeby testów tworzymy przykładowe reguły:

- w przypadku, gdy klient prowadzący jednoosobową działalność ma nie mniej niż 18 lat i nie więcej niż 35 lat, to ubezpieczenie jest najniższe (5% wartości kapitału),

- w przypadku, gdy klient prowadzący jednoosobową działalność ma więcej niż 35 lat i mniej niż 50, to ubezpieczenie jest na średnim poziomie 8% wartości kapitału,
- w przypadku, gdy klient prowadzący jednoosobową działalność ma 50 lub więcej lat, to ubezpieczenie jest na najwyższym poziomie 12% wartości kapitału,
- w przypadku spółki z o.o, jeżeli dochód jest większy lub równy 500 000 zł, ubezpieczenie kredytu nie jest wymagane,
- w przypadku spółki z o.o jeżeli dochód jest mniejszy niż 500 000 zł i większy lub równy 200 000 zł, ubezpieczenie kredytu wymagane i jest to 5% wartości kredytu,
- w przypadku spółki z o.o, jeżeli dochód jest mniejszy niż 200 000 zł, ubezpieczenie kredytu wymagane, jest to 10% wartości kredytu.

W celu testowania sterowalności pól dodatkowo należy zdefiniować na formularzu pole z kwotą Kapitału, aby wiedzieć, czy tworzenie reguł jest prawidłowo obsłużone. Formularz utworzony przez osobę testującą powinien mieć takie pola jak niżej na obrazku:

Forma prawna*

Dochód z ostatnich 12 miesięcy

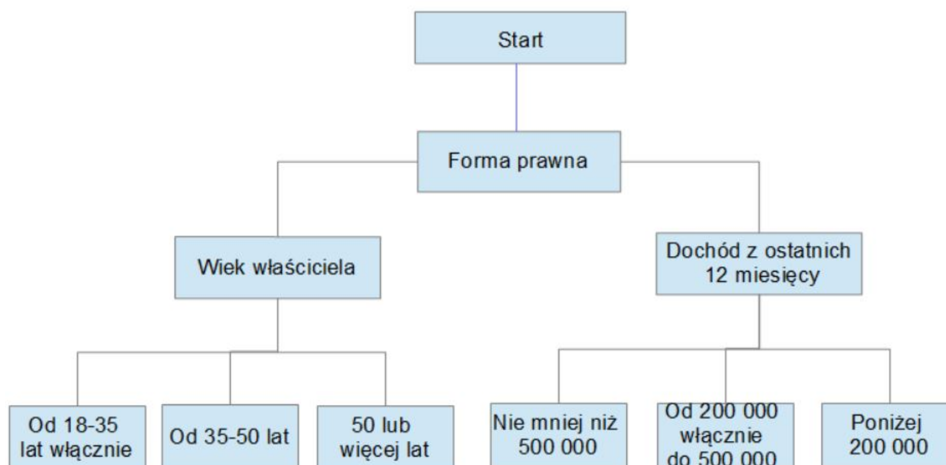
Wiek właściciela firmy

Kwota kapitału*

Kwota ubezpieczenia*

Rysunek 2: Formularz utworzony na potrzeby testów - opracowanie własne

Pola **Dochód z ostatnich 12 miesięcy** oraz **Wiek właściciela firmy** nie są domyślnie obowiązkowe, ponieważ pole **Forma prawna** domyślnie jest puste - to pierwsza z utworzonych przez osobę testującą reguł. Kroki jakie należy wykonać, aby przetestować podstawowe ścieżki pokazuje niżej przedstawiony graf:



Rysunek 3: Podstawowe ścieżki testów wyżej wymienionej funckjonalności - opracowanie własne

Analiza i opis są analogiczne jak w przykładzie 1. Testy mogą również być przeprowadzone w sposób negatywny, aby sprawdzić, czy framework umożliwiający starowanie polami działa tak, jak należy. Przedziały wiekowe lub dochody również można zmienić.

4 Podsumowanie

Celem napisania artykułu było podkreślenie problemu testowania obiektów, które są "programowane" przez osobę testującą, a nie przez programistę. Jest to dość ciekawa klasa problemów z punktu widzenia testowania oprogramowania, ponieważ wymaga od testera dodatkowego zaangażowania podczas projektowania testów, jakim jest pewien schemat "programowania", nie związany wprost z samym językiem kodu programu. Taka klasa problemów może pojawić się np. w aplikacjach finansowych, jeśli użytkownik ma możliwość ofertowania pewnych produktów finansowych (kredyt, leasing) oraz prezentowania kilku wariantów produktu finansowego dla klienta.