

# Idealne środowisko testowe

Piotr Kałuski

## CGI

### **Piotr Kałuski**

Jest absolwentem Wydziału Elektroniki i Technik Informatycznych Politechniki Warszawskiej, kierunek informatyka. Od 1995 uczestniczył w wielu projektach informatycznych jako programista analityk, projektant i kierownik zespołu. Obecnie jest pracownikiem firmy CGI i jako konsultant jest członkiem zespołu System Test w firmie Polkomtel. Dziedzina jego zainteresowań to sposoby usprawnienia procesu testowania i wykorzystanie narzędzi Open Source w procesie wytwarzania i testowania oprogramowania. Szczegóły można znaleźć pod adresem [www.piotrkaluski.com](http://www.piotrkaluski.com).



# Idealne środowisko testowe

Piotr Kałuski

## CGI

W poniższym artykule chciałbym przedstawić cechy, jakie powinno mieć idealne środowisko testowe. Idealne z punktu widzenia testera. Z góry zaznaczam, że środowisko idealne z punktu widzenia testera może być w wielu sytuacjach niemożliwe do stworzenia ze względu na koszty sprzętu, licencji, na nakład pracy potrzebnej do utworzenia tego środowiska, zarządzania nim i jego utrzymania. Jednak czasami warto zastanowić się nad tym, co czyni ideał ideałem. Wiedza ta, nawet przy pełnej świadomości nierealności tegoż ideału, pozwala nam łatwiej wyznaczać cele i planować prace nad produktem, który ma być możliwie najlepszym ucieleśnieniem teoretycznej doskonałości.

Jest jeszcze jeden cel, który chcę osiągnąć w tym artykule. Koszty implementacji danych cech środowiska są widoczne prawie że gołym okiem. Obliczamy ile będzie potrzeba sprzętu, ilu ludzi do utworzenia i utrzymania środowiska i mamy zgrubne oszacowania. Tymczasem ocena strat spowodowanych brakiem pewnych udogodnień w środowisku testowym wymaga głębszej wiedzy na temat procesu testowania, czasami wymaga doświadczeń z „pola bitwy”. Na takie właśnie ukryte konsekwencje oszczędzania na ułatwianiu życia testerom, postaram się zwrócić uwagę. Skupię się na następujących cechach ideału:

- Niezależność i samowystarczalność środowisk poszczególnych testerów
- Łatwość w odtwarzaniu stanów środowiska
- Łatwość w debugowaniu środowiska
- Automatyzacja testów powtarzających się

## Niezależność i samowystarczalność środowisk

W idealnym świecie każdy tester powinien mieć w pełni niezależne i samowystarczalne środowisko. Myślę, że określenie „niezależne” jest zrozumiałe. Mogą się zrodzić wątpliwości, co mam na myśli pisząc „samowystarczalne”. Wszystko wyjaśnię. Generalnie chodzi o to, aby każdy tester mógł przeprowadzić swoje testy bez potrzeby koordynacji z kimkolwiek i bez potrzeby proszenia kogokolwiek o wykonanie jakiejś operacji, bez której rezultatów testy nie mogą być doprowadzone do końca.

W firmie, gdzie środowisko produkcyjne jest złożone a testerów jest 15, oznaczałoby to konieczność konfiguracji 15 pełnych środowisk. W większości firm jest to niewykonalne a jeżeli nawet wykonalne fizycznie, to na pewno niewykonalne

finansowo. Jednak poważne zaniedbanie tematu niezależności może mieć bardzo kosztowne konsekwencje.

Jeżeli mamy w zespole 5 testerów, i wszyscy testują tę samą aplikację typu klient-serwer to istnieją różne możliwe konfiguracje. Rozważmy sytuacje skrajne - Wszyscy testerzy są podłączeni do jednego serwera, bądź też każdy ma swoje środowisko i swój serwer. Pierwsze rozwiązanie ma nad drugim tę przewagę, że jest prostsze w konfiguracji. Drugie ma jednak pewną zaletę, która czasami ma kapitalne znaczenie - środowiska testerów są **niezależne**. Każdy może przeprowadzić swoje testy bez potrzeby koordynacji z innymi. Każdy może prowadzić testy bez ryzyka, że niechcący popsuje coś innemu testerowi i bez strachu, że ktoś zniszczy mu jego dane, które przygotowywał przez ostatni tydzień. Możliwe straty wynikające z takich pomyłek są różne - od żadnej po zniweczenie tygodnia pracy kilku osób. Jeżeli wykonujemy tylko proste testy typu nasza akcja - reakcja systemu, to ryzyko kolizji i ewentualne straty są niewielkie. Jednak przy testowaniu aplikacji biznesowych takie proste testy stanowią tylko część wszystkich zaplanowanych scenariuszy. Aby przetestować np. nowy sposób naliczania należności za usługę, gdzie w grę wchodzi zniżka i staż klienta, wykonanie testu wymaga wykonania kilku akcji, często w określonej kolejności ze względu na testowane zależności czasowe. Inny przykład to testowanie funkcjonalności odpowiedzialnej za raporty. Jeżeli chcemy sprawdzić czy odpowiednie pola w raporcie odpowiednio się sumują, to musimy wprowadzić odpowiednie dane szczegółowe będące bazą dla raportu.

Jak może dochodzić do „*tragicznych*” pomyłek? Na przykład system pozwala poszczególnym testerom wykonywać testy prawie niezależnie w jednym środowisku. Każdy może wykonać większość kroków bez obawy, że coś komuś popsuje. Jednak wszyscy muszą uważać, żeby nie wykonać ostatniego kroku - procesu, który np. bierze dane wygenerowane przez wszystkich testerów, po czym zmienia zawartość bazy danych w taki sposób, że poprzednie dane testowe stają się nieaktualne, tzn. nie można ich już użyć do powtórzenia testów. Oczywiście zmiany w bazie danych są odwracalne, ale jeżeli tym ostatnim krokiem był proces uaktualniający, globalnie, dane związane z należnościami za cały miesiąc, to złożoność tych zmian może czynić je nieodwracalnymi z praktycznego punktu widzenia. Jeżeli coś takiego się wydarzyło i został zniweczony tydzień pracy 5 ludzi, to kierownik zespołu testerów, któremu się to komunikuje, powinien w pierwszej kolejności dziękować Bogu, że się wogóle o tym dowiaduje. Jeżeli szkoda powstała w wyniku akcji niedoświadczonego testera, który bardzo boi się utraty pracy, to może on spróbować ukryć ten fakt przed innymi i próbować zamazać ślady. Kierownikowi zespołu zaraportuje, że wszystko jest w najlepszym porządku i system pójdzie na produkcję nieprzetestowany. A o tym, jak kosztowne jest wprowadzenie wadliwego oprogramowania na produkcję, wie każdy doświadczony informatyk.

Oczywiście, niektórzy czytelnicy mogą mieć wrażenie, że wyolbrzymiam problem, lub też, że problem ma proste rozwiązania:

1. ***Skoro ten nieodwracalny krok jest tak niebezpieczny to najlepiej go nie wykonywać.***

To nie jest takie proste. Jeżeli jest to proces standardowo wykonywany na produkcji, to koniec końców, musi on być też wykonany w testach

## 2. *Testerzy mogą koordynować między sobą swe poczynania*

Tak, to prawda. Co więcej, testerzy tak właśnie robią. Ale w pośpiechu łatwo o pomyłkę. A nawet, jeżeli nie ma pośpiechu, to przy wykonywaniu skomplikowanych scenariuszy ze złożonymi danymi każdy może się pomylić.

Sprawa się pogarsza, jeżeli takich nieodwracalnych procesów jest więcej i to one są akurat testowane. Jeżeli nawet nie dochodzi do pomyłek i nawet jeżeli testerzy sobie nawzajem nic nie popsują to pojawia się następny problem - testy jednych wstrzymują testy drugich. Jedna grupa testerów nie chce uruchamiać nieodwracalnych procesów w swoich testach. Druga musi to zrobić. Jeżeli chce się to pogodzić, to grupa druga musi poczekać, aż pierwsza grupa zakończy swoje testy. A w informatyce czekanie kosztuje. Kosztuje tym więcej im więcej jest osób w dziale testów i im wyższe są kwalifikacje testerów (czyli ich pensje).

Rozwiązaniem tego problemu jest danie każdemu testerowi niezależnego środowiska. W zależności od sprzętu, na którym testy są wykonywane, cena zasobów potrzebnych na powielenie środowiska (procesory, pamięć RAM i dyskowa) może się różnić. Jednak ceny zasobów komputerowych systematycznie tanieją (licencje na oprogramowanie niestety nie). To jednak niestety nie wszystko. Ktoś musi środowiskami testowymi zarządzać. Jeżeli 10 testerów pracowało na jednym środowisku i teraz chcemy każdemu z nich dać osobne środowisko, to osoba odpowiedzialna za konfigurację tych środowisk, będzie miała 10 razy więcej pracy. Jej czas musi być uwzględniony w naszych kalkulacjach. Podejmując decyzje w kwestii środowisk testowych, musimy więc uwzględnić następujące elementy:

- jak bardzo poszczególni testerzy są zależni od innych, jak długie mogą być zastoje niektórych grup,
- ile czasu zostanie zmarnowane przez ewentualne pomyłki,
- koszt osobodnia testera,
- koszt zasobów sprzętowych i licencji potrzebnych do dania każdemu testerowi osobnego środowiska,
- koszt konfiguracji i utrzymania tych środowisk,

Te parametry na pewno różnią się w zależności od projektu, i może się zdarzyć, że operowanie testerów na jednym środowisku, nawet po uwzględnieniu kolizji, jest finansowo bardziej opłacalne niż dawanie testerom osobnych środowisk.

A teraz coś o samowystarczalności. Najłatwiej będzie mi to zilustrować przykładem.

Wyobraźmy sobie, że w banku testuje się system windykacyjny. Raz na jakiś czas, system łączy się z systemem obsługi kart kredytowych, aby np. zablokować kartę dłużnika. Jeżeli środowisko jest samowystarczalne to tester ma do dyspozycji obydwie

systemy – windykacyjny i do obsługi kart. Kiedy testuje scenariusz, w którym karta kredytowa dłużnika jest blokowana, może natychmiast sprawdzić czy system zablokował kartę czy też nie. Jeżeli tak się nie stało, to może łatwo powtórzyć test i zobaczyć, co nie działa.

A teraz wyobraźmy sobie inną sytuację. Każdy tester ma swoją kopię systemu windykacyjnego. Ale system obsługi kart kredytowych jest tylko 1 – ten używany na produkcji. Dzieje się tak - na przykład - dlatego, że pojedyncza licencja jest bardzo droga. Testujemy znów scenariusz z dezaktywacją karty dłużnika. Od obsługi technicznej systemu obsługi kart dostajemy pulę numerów kart, które na pewno nie będą użyte na produkcji, więc mogą być wykorzystane do testów.

**Jest 9 rano.** Tworzymy w systemie windykacyjnym scenariusz, w którym saldo i historia klienta powodują wysłanie żądania dezaktywacji karty. Żądanie jest wysyłane do systemu obsługi kart. Ponieważ, jest to system produkcyjny, zawierający informacje poufne, nie możemy ot, tak sobie sprawdzić czy karta jest dezaktywowana. Musimy zadzwonić do obsługi systemu kart i zapytać się jaki jest stan karty o numerze taki a takim.

**Jest 9:30.** Dzwonimy. Nikt nie odbiera.

**O 10** udaje nam się kogoś złapać:

*Tester czyli my: Słuchaj, jaki jest stan karty XXX? ”.*

*Ktoś z obsługi systemu kart: Wiesz co, musisz z tym poczekać. Coś się u nas sypie i mamy urwanie głowy. Zadzwoń o 11.*

Cóż robić. Zabieramy się za coś innego (jeżeli mamy co innego do roboty) a przynajmniej udajemy, że coś robimy.

**O 10:45** mamy zebranie zespołu.

Wracamy do biurka o **11:30**. Dzwonimy:

*Tcm: No i jak?*

*Kzosc: Podaj mi jeszcze raz numer karty. Achaaaa, tiaaa... Jest aktywna.*

Do diabła! A miała być dezaktywowana.

*Tcm: A czemu? Wysłałem żądanie dezaktywacji*

*Kzosc: Tak? To poczekaj sprawdzę logi. Zadzwonię do Ciebie.*

**O 11:45** dzwoni:

*Kzosc: No nie wiem co jest nie tak. Muszę poszperać głębiej. Mam teraz coś pilnego do zrobienia. Zajmę się tym po lunchu, OK?*

*Tcm: Dobra.*

**Jest 12:00**, idziemy na lunch, bo nie mamy co robić, musimy czekać.

**O 13** dzwoniemy:

*Kzosc: Dopiero wróciłem. Już patrzę.*

Dzwoni o **13:20**:

*Kzosc: No nie wiem, wygląda jakby żadne żądanie nie przyszło.*

Sprawdzamy połączenie – powinno działać. Próbujemy jeszcze raz – dostajemy komunikat „Nie można dezaktywować karty nieaktywnej”. Czyli już gdzieś w systemie jest zapamiętane, że takie żądanie poszło. Dzwonimy. Słyszymy:

*Kzosc: Teraz nie mogę, jestem na spotkaniu. Zadzwoń o 14:30.*

Dzwonimy, mówimy co się dzieje.

*Kzosc: To może żądania nie przechodzą przez interfejsy. Zadzwoń do interfejsów.*

Dzwonimy. Ktoś odbiera i ma dla nas czas (a to dopiero zbieg okoliczności!). Okazuje się, że interfejs łączący środowisko testowe z systemem kart jest wyłączony. Włączają go. Żądanie przechodzi do systemu kart.

**O 15** dzwoniemy i dowiadujemy się, że żądanie zostało odrzucone z błędem „Zła suma kontrolna karty”. Sprawdzamy – rzeczywiście. Pomyliliśmy się przy jednej cyferce.

**Jest 15:30**. Tworzymy nowy scenariusz.

**O 16:00** idzie nowe żądanie. Dzwonimy, nikt nie odbiera. No tak, tam pracują od 8 do 16. Musimy poczekać z tym do jutra.

Po 2 dniach okazuje się, że żądania są wysyłane w niepoprawnym formacie. A gdybyśmy mieli system obsługi kart do własnej dyspozycji, wykrylibyśmy to w 2 godziny. I niech mi nikt nie mówi, że przesadzam. Byłem świadkiem takich historii nie raz i nie dwa razy.

## **Jak to wdrożyć**

No cóż, najprościej jest kupić odpowiednią ilość sprzętu i licencji. Jeżeli nie da się każdemu dać pełnego środowiska, to przynajmniej spróbujmy dać tym środowiskom tyle niezależności ile tylko jest możliwe.

Warto zastanowić się nad automatyzacją procesu tworzenia środowiska. To może pozwolić osobom utrzymującym system na spędzanie mniej czasu na ustawianiu pojedynczego środowiska a co za tym idzie będą mogli ich tworzyć więcej.

Przy negocjacjach zakupu oprogramowania specjalistycznego, warto mieć tę kwestię na uwadze. Jeżeli jesteśmy dużym klientem i płacimy ciężkie pieniądze, spróbujmy wymusić na dostawcy oprogramowania darmowe licencje do testów. Jeżeli damy się „wpuścić” w jedną licencję, na dodatek z kluczem sprzętowym, tak że produkcja będzie musiała dzielić klucz z testerami, bo nie będzie można korzystać jednocześnie, to od razu możemy czas testów wykorzystujących to oprogramowanie pomnożyć przez 5. Będzie to klasyczny brak samowystarczalności, o którym pisałem powyżej.

## **Łatwość w odtwarzaniu różnych stanów środowiska.**

Jeżeli test przebiega pomyślnie to po wykonaniu scenariusza temat się właściwie kończy. Jeżeli jednak test nie przejdzie to możemy mieć do czynienia z następującymi sytuacjami:

1. **Błąd jest ewidentny, nie ma wątpliwości co poszło źle.** Dzwonimy do programisty i mówimy co się stało. On patrzy w kod i widzi to, co jest nie tak albo powtarza nasz scenariusz u siebie i dostaje ten sam błąd. Ma wszystkie informacje potrzebne do naprawienia błędu

Powyższy scenariusz był najbardziej pozytywny z możliwych. Może jednak nie być tak łatwo

2. **Błąd nie jest ewidentny.** Rezultaty nieznacznie różnią się od oczekiwań, ale jeżeli wykonujemy akurat skomplikowany scenariusz to nie jesteśmy pewni czy może nie popełniliśmy jakiejś pomyłki w scenariuszu. Najlepiej by było powtórzyć test jeszcze raz
3. **Błąd jest ewidentny. Scenariusz skomplikowany i trudno odwracalny.** Dzwonimy do programisty. Mówimy co jest nie tak. Ten próbuje u siebie. „Słuchaj” – mówi – nie udało mi się tego odtworzyć. Mógłbyś powtórzyć ten test jeszcze raz?”
4. **Jak punkt 3, z tym że programista może debugować w naszym środowisku.** Po wykonaniu scenariusza mówi – Już chyba wiem o co chodzi. Możemy ten test wykonać jeszcze raz?

We wszystkich tych sytuacjach możliwość szybkiego zapamiętania i odtworzenia danego stanu środowiska może nam oszczędzić godziny na każdym powtórzeniu scenariusza. To w sumie potrafi dać kilka dni oszczędności.

## **Jak to wdrożyć**

Najprostszym sposobem zapamiętywania stanu środowiska jest robienia jego kopii. Może to być długotrwałe i miejsce-żerne dla dużych środowisk.

Inną drogą jest utworzenie skryptów, które szybko odtworzą stan środowiska z przed wykonania danego scenariusza.

## **Możliwość debugowania<sup>1</sup> w środowisku testowym**

Ta cecha środowiska jest z obopólną korzyścią dla testera i programisty a także dla całego projektu.

Środowisko, na którym pracuje tester powinno być zorganizowane w taki sposób, aby na wypadek błędu, który jest trudny do odtworzenia w środowisku deweloperskim, programista mógł uruchomić debugger w środowisku testera.

Podobnie jak w innych przypadkach, dostępność tego ułatwienia może zaoszczędzić kilka dni pracy testera i programisty. Niejednokrotnie, środowisko aplikacji ma tyle stopni swobody, że ręczne odtworzenie środowiska testera w środowisku programisty może być albo praktycznie niemożliwe albo bardzo pracochłonne. Jeżeli w środowisku testowym jest debugger, to nie ma tematu odtwarzania środowisk. Tester odtwarza błąd, woła programistę a ten może prześledzić działanie programu w środowisku w którym ten błąd występuje.

### **Jak to wdrożyć**

Trzeba w środowisku testowym zainstalować większość niezbędnych narzędzi, z których korzystają programiści przy szukaniu błędów. Czy jest to proste czy trudne – zależy od projektu. Ale w zasadzie, nie powinno być bardzo trudne ani bardzo kosztowne.

## **Automatyzacja testów powtarzających się**

Automatyzacja to wśród testerów ostatnimi czasy bardzo popularne słowo. I na pewno warto się do automatyzacji przymierzyć. Jest kolosalnym błędem próbować zautomatyzować wszystko (gorszym od nieautomatyzowania niczego). Ale automatyzacja rozważnie i mądrze wybranych zadań, może być źródłem wielotygodniowych (sic!) oszczędności.

### **Jak to wdrożyć**

Po prostu zainteresować się tematem.

Przepraszam, że ten temat traktuję tak skrótowo. To nie przez lenistwo. Automatyzacja testów jest procesem złożonym i nawet jej pobieżne potraktowanie

---

<sup>1</sup> Debugowanie – Proces uruchamiania programu krok po kroku, z możliwością śledzenia zmian będących konsekwencją każdego kroku.

wymaga osobnego artykułu. Dość ciekawą i wyważoną dyskusję na temat automatyzacji testów można znaleźć na przykład na [www.qaforums.com](http://www.qaforums.com). Ale nie tylko. Temat automatyzacji jest dość popularny w środowisku testerów.

## **Podsumowanie**

Próbowałem pokazać jakie mogą być konsekwencje pewnych braków w środowiskach testowych. Czy to znaczy, że uważam narzędzia i ułatwienia opisane powyżej za rzecz niezbędną i uważam managerów, którzy z nich rezygnują za ignorantów? Absolutnie nie. Nie jest celem tego artykułu zdiagnozować, co jest najważniejsze i wskazać jedynie słuszną drogę. Chciałem tylko zasignalizować istnienie zjawisk, które mogą umknąć uwadze osób na stanowiskach kierowniczych. Koszty generowane przez te zjawiska warto wziąć pod uwagę obok cen sprzętu i licencji. Każdy manager na ich podstawie będzie musiał sam podjąć decyzję, gdzie, na tej wadze szalkowej - jaką jest jego projekt, przesunąć odważnik tak aby waga pozostała w równowadze. Czy bliżej bieguna oszczędności, czy bliżej bieguna usprawniania procesów.