

# Automatizacija testiranja postaje stvarnost

■ GORAN BURAZER

Tradicionalne metodologije razvoja softvera baziraju se na tzv. vodopad-pristupu (engl. *water-flow*) - faze prikupljanja zahtjeva, dizajna, implementacije, testiranja i održavanja softvera izvršavaju se slijedno od početka do kraja. Ovakav način za većinu modernih projekata nije dovoljno dobar, prvenstveno zbog neefikasnog odgovora na promjene u korisničkim zahtjevima. Ukratko, kada želimo napraviti nešto, ali još ne znamo dovoljno detalja, treba nam drukčiji pristup.

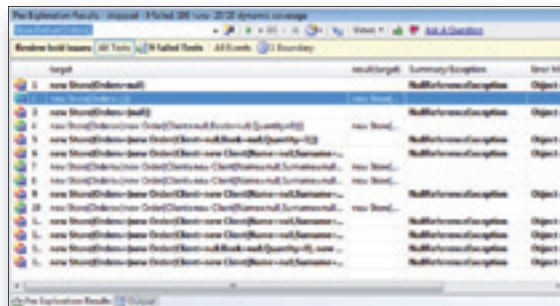
Agilne metodologije nude rješenje bazirano na postupnom razvoju s čestim preispitivanjem odluka. Korisnički zahtjevi i odgovori na njih (softver koji razvijamo) evoluiraju kroz suradnju malih interdisciplinarnih timova. Svojevremeno kontroverzne, danas su dobile svoje mjesto u razvojnim projektima diljem svijeta, kao i brojne inačice (*Extreme Programming*, *Scrum*...). Godine 2001. je objavljen manifest (<http://agilemanifesto.org/>) koji definira taj novi pristup.

Neke **prakse u razvoju softvera**, iako uglavnom otprije poznate, izbile su u prvi plan kao ključni potporni agilnih projekata. To su npr. programiranje u paru, *unit testing*, razvoj vođen testovima (engl. *TDD - test-driven development*), kontinuirana integracija, *refactoring* itd.

## Unit testing i razvoj vođen testovima

Jedna od ključnih praksi koje omogućuju agilnost procesa je *unit testing*. Programer

*Unit testing* je široko prihvaćena i vrlo korisna praksa u razvoju softvera. Alati Pex i Moles pomažu savladati neke od izazova pisanja kvalitetnog skupa testova, prvenstveno automatsko traženje graničnih slučajeva i time kompletno pokrivanje koda testovima, te izoliranje kôda koji se testira od ostatka sustava



Pex u akciji!

osim produkcijskog koda piše i testove (u obliku dodatnog koda, obično organiziranog kroz neki od popularnih *frameworka*, npr. MS Test ili NUnit) koji osiguravaju da produkcijski kôd radi ono što je programer predvidio.

Srodna agilna praksa je *test-driven development* (razvoj vođen testovima), gdje se svaki *unit-test* piše prije koda koji ga zadovoljava. Na taj način, koristeći i *refactoring* (još jedna agilna praksa, mijenjanje dizajna koda bez promjene ponašanja), dizajn koda nije unaprijed zadan, već izranja s vremenom kako se softveru dodaju mogućnosti (testovi). Produkcijski kod nastao TDD-om je očito pogodan za *unit testing*, a dobiveni skup testova osigurava maksimalnu (100%) pokrivenost koda testovima budući da se za zadovoljavanje svakog testa piše minimalna implementacija.

Razvoj vođen testovima unaprjeđuje proces, ali također ima svoje probleme i ograničenja. Može li se *unit testing* organizirati jednostavnije, bolje, automatski?

## Nov pristup - parametrični unit-testovi

*Unit-test* je metoda bez parametara koja bi trebala testirati jednu i samo jednu karakteristiku programa. Unaprjeđenje ovog pristupa je parametrični *unit test* (metoda s parametrima) koji ovisno o vrijednostima parametara odjednom testira više različitih slučajeva. Na taj način programer može napisati jedan test, „nahraniti“ ga odabranim parametrima (npr. iz baze podataka) i tako istražiti više sličnih karakteristika programa jer se zapravo isti test pokreće više puta, uz različite vrijednosti parametara. Danas praktički svaki *unit testing framework* podržava parametrične testove.

Nakon što je prepoznata prilika za stvaranje parametričnog *unit-testa*, problem „do-

brog“ testiranja koda svodi se na odabir vrijednosti parametara - konkretnih testova. Cilj je izvršavanjem testa uz zadane vrijednosti parametara postići što veću pokrivenost produkcijskog koda uz minimalan ukupan broj testova. Redundantni testovi troše vrijeme testnog računala i programera koji čeka da se izvrše, a ne dodaju nove informacije i još ih treba održavati paralelno s kodom.

## Pex, Program Explorer

Dobra vijest je da se optimalan odabir parametara može automatizirati i upravo tome služi Pex, vrući novi alat iz Microsoft Researcha koji bi mogao donijeti revoluciju u sam koncept *unit-testinga*.

Nakon što mu programer zada koje *assemblyje* smije istraživati, Pex je u stanju generirati kostur parametričnog *unit-testa* za odabranu metodu ili više njih (za cijelu klasu), pronaći minimalan skup vrijednosti ulaznih parametara tako da se testiraju sve grane u kodu (postići potpuni *code coverage*), detektirati vrijednosti ulaznih parametara koje „ruše“ program (izazivaju neobrađene iznimke) te snimiti konkretan skup klasičnih *unit-testova* (pomoću pronađenih parametara) koji se mogu priložiti produkcijskom kodu (*checkinati* u *version control* sustav) za kasnije regresijsko testiranje.

Pex je moguće na razne načine konfigurirati i usmjeriti u istraživanju dijela koda koji mu je zadan. Mogu se pripremiti *factoryji* za objekte (kada parametar test-metode nije vrijednost elementarnog tipa, već objekt). Može se forsirati odabir nekih vrijednosti parametara ili zabraniti odabir nekih drugih. Pex je dobro integriran s *Code Contracts* sustavom, još jednim dodatkom za Visual Studio iz Microsoft Researcha koji omogućuje zadavanje uvjeta koji moraju biti ispunjeni prije i nakon metode (što ograničava skup vrijednosti njenih parametara).

Također, Pex ne mora nužno generirati *unit-testove* za *defaultni* MS Test *framework*, već je ekstenzibilan i postoje *pluginovi* za skoro svaki *framework* u upotrebi danas.

**Oprez!** Pex će izvršiti svaku moguću putanju kroz kod koji se testira. Ako se pritom koriste vanjski resursi (npr. datotečni sustav ili baza podataka) ili kôd kontrolira fizičke uređaje, osigurajte da su odspojeni prije pokretanja Pexa. Inače izvršavanje svih putanja testiranog koda može nanijeti ozbiljnu

### Pex and Moles

Proizvođač	Microsoft
Tip	Dodatak razvojnom alatu
Minimalna konfiguracija	Kao i za VS2010, 1,6 GHz procesor, 1 GB (32 bit OS) ili 2 GB (64 bit OS) RAM
Preporučena konfiguracija	Kao i za VS2010, 3 GHz quad-core procesor, 4 GB DDR3 RAM
Cijena	Besplatan (inačica <i>Academic</i> ), besplatan uz MSDN pretplatu



Automatizacija rutinskog, a važnog posla izrade testova, konfigurabilnost, kontrola (Moles)



Problemi s nekim tipovima parametara, kompleksnost, rizik nestručnog rukovanja

### DOJAM

Sjajni alati za povećanje kvalitete testiranja i razvoja aplikacija



Pex for fun - „Lite“ verzija na webu za istraživanje komadića koda



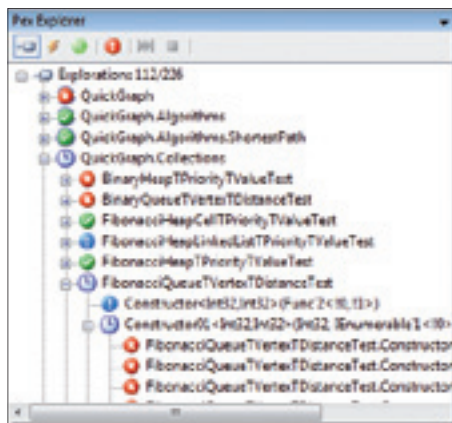
Kako radi Pex? Shematski prikaz glavne petlje za otkrivanje zanimljivih ulaznih parametara

štetu vanjskim resursima (brisanje datoteka ili baza, oštećivanje uređaja).

### Moles framework za izolaciju koda

Jedan od problema kod pisanja *unit*-testova je odvajanje produkcijskog koda koji se testira od ostatka sustava. Budući da test treba biti ponovljiv i što jednostavniji (atomaran), on ne bi smio ovisiti o okruženju, a ta izolacija se standardno postiže korištenjem tzv. oponašajućih (engl. *mock*) objekata. Umjesto dijela sustava o kojem ovisi kod koji se testira, programer prilikom pisanja testa „podmeće“ unaprijed pripremljen objekt koji ima isti *interface*, ali s ponašanjem programiranim za potrebe testa (umjesto pozivanja stvarnog vanjskog objekta). Naravno, produkcijski kod treba biti strukturiran tako da je *mock*-objekt moguće podmetnuti umjesto stvarnog pod-sustava (indirekcija na odgovarajućem mjestu u kodu), što se najlakše postiže upravo TDD-om.

**Moles** (hrv. krtice) je *framework* koji je izrastao za vrijeme izrade Pexa, a cilj mu je sistematski omogućiti odvajanje produkcijskog koda koji se testira od ostatka sustava.



Pex Explorer - prikaz napretka Pexovog istraživanja

Moles je *framework* za *mock*-objekte. Iako se može koristiti i samostalno, pravu snagu pokazuje u sprezi s Pexom. U krajnjem slučaju je moguće funkcijski poziv na nivou IL-a preusmjeriti na vlastiti delegat (funkciju), čime je moguće odvajanje bilo kojeg komada .NET koda.

### Kako to sve upogoniti?

Pex i Moles su *power tools* za Visual Studio 2010 (koji postoji i u sasvim besplatnoj, Express inačici). Moguće ih je besplatno preuzeti s Microsoft Research stranica posvećenih Pexu (<http://research.microsoft.com/pex/>), koje su usput krzate resursima (*tutoriali*, videoprezentacije, *whitepaperi*). Dostupni su i u besplatnoj inačici (*Academic*), naravno za nekomercijalnu uporabu, a u punoj verziji za MSDN pretplatnike. Ako se koristi s Visual Studio Expressom, dostupna je samo komandnolinjska verzija, inače je na raspolaganju i vrlo funkcionalan GUI (slike).

S Pexom se možete zaigrati i *online*, na stranici <http://www.pexforium.com/>. Zadatak je jednostavan: u odabranom .NET programskom jeziku (C#, Visual Basic, F#) napisati implementaciju („tajni zadatak“) koja je zadana Pexovim testovima. Probajte, zabavno je!

Pex se pokazao odličnim za detaljno ispitivanje i pronalaženje *bugova* u algoritamski zahtjevnom i kompleksnom kodu. Npr. Microsoft je pokrenuo Pex analizu nad nekim dijelovima .NET *frameworka* i pronašao nove *bugove* u kodu koji se koristi godinama i do sada je smatran dobro testiranim, što znači da je Pex otvorio vrata nove razine kvalitete.

Već sada TDD prelazi u BDD (*behavior-driven development*, razvoj vođen ponašanjem), što znači da se sustavi dizajniraju tako da se umjesto testiranja koda (previše detaljan nivo) **provjerava ponašanje** sustava. Scenariji korištenja sustava odgovaraju korisničkim zahtjevima i propisuju kakvo treba biti ponašanje sustava uz određene ulazne parametre. Prilikom razvoja, umjesto pokretanja skupa testova, pokreće se verifikacija scenarija korištenja čime je cijeli TDD koncept podignut na viši nivo. Postoji i nekoliko *frameworka* za .NET (npr. xBehave, xSpec). Ali što je s *unit*-testovima? I dalje ih možemo generirati i koristiti, pomoću Pexa!

Ovakav spoj dizajniranja aplikacija s pogledom s veće visine (ponašanje aplikacije nasuprot atomarnim *unit*-testovima za aplikaciju), uz zadržavanje detaljnog uvida u točnost izvršavanja koda pomoću *unit*-testova automatski generiranih alatom poput Pexa sada izgleda kao dio budućnosti razvoja aplikacija. @

## Unit-testing: prije i nakon Pexa

Ako ne pišete *unit*-testove uz produkcijski kod, trebali biste. Njima se postiže dokumentiranje korisničkih zahtjeva, demonstriranje dizajnerskih odluka, zaštita od uvođenja neželjenih promjena (uz provođenje regresijskog testiranja) i, najvažnije, samopouzdanje da je kod dobro testiran i robusan ako je postignuta dobra pokrivenost koda testovima (engl. *code coverage*).

Iako je o vještini pisanja dobrih *unit*-testova napisana gomila materijala, to je i dalje mukotrpan i pogreškama podložan proces za koji programeri često nemaju dovoljno strpljenja, a ni vremena. Zato dolazi do problema:

- Nedovoljna kvaliteta testova
- Ovisna o vještini programera i vremenu koje je spreman uložiti u pisanje testova
- Nedovoljan broj testova - novi testovi ne znače nužno povećanje pokrivenosti koda
- Novi kod, stari testovi - kod se obično održava ažurnije nego testovi, pa se pojavljuju netestirani dijelovi
- Integracijski umjesto *unit*-testova - ako test ovisi o okruženju u kojem se izvršava (npr. datotečni sustav na disku ili baza podataka s određenim sadržajem), a ne samo o kodu koji testira, kasnije ga možda neće biti moguće reproducirati i postaje beskoristan

Proces *test-driven developmenta* uz korištenje Pexa i parametričnih *unit*-testova izgleda ovako:

1. Napisati ili promijeniti parametrični *unit*-test, ili kod koji implementira ponašanje opisano postojećim parametričnim *unit*-testovima
2. Pokrenuti Pex nad parametričnim *unit*-testom
3. Ako je Pex pronašao pogreške, vratiti se na korak 1 i popraviti test ili kod, možda ograničavanjem vrijednosti ulaznih parametara koje se razmatraju
4. Zadržati generirane *unit*-testove za buduće regresijsko testiranje