Niele Python

Marko Komssi, Mika Eloranta
F-Secure Corporation

Ohjelmistoteollisuudessa kehitysjohtajilta usein kuultava kommentti on, että automaattisesti suoritettavien testien määrää tulisi lisätä. He kuitenkin jättävät huomioimatta, että testausautomaatioprojektit epäonnistuvat usein (Kaner, C., Pitfalls and strategies in automated testing. Computer, 10, 4, 1997). Yksi epäonnistumisen syistä on väärin valittu työkalu. Toinen merkittävä epäonnistumisten syy on testauksen väärä kohde. Esimerkiksi käyttöliittymien kautta tehtävä testien automaattinen suoritus on yleistä, mutta harvoin kustannustehokasta, toisin kuin työkalujen myyjät väittävät.

Jos työkalu ja testauksen kohde ovat sopivia, testien automatisoinnilla on mahdollista saavuttaa merkittäviä hyötyjä. Kokemuksemme Python-ohjelmoinnista testauksen apuna ovat olleet myönteisiä erityisesti F-Securen tuotekomponenttien automaattisessa rajapintatestauksessa, jossa testikoodi kutsuu tuotekomponenttien tarjoamia rajapintafunktioita. Tässä artikkelissa kerromme meidän Python-kokemuksistamme ja sen eduista testausautomaatiossa.

”Python on ilmainen, helppo asentaa ja oppia sekä tehokas käyttää”

Mikä ihmeen Python?

Python (ks. http://www.python.org) on houkutteleva välinevaihtoehto testien automatisointiin. Se on oliopohjainen ohjelmointikieli, joka tukee monin tavoin testien automatisointia. Python on ilmainen, helppo asentaa ja oppia sekä tehokas käyttää. Sen vahvuuksia ja testausautomaatiossa hyödyllisiä ominaisuuksia ovat esimerkiksi:

- sisäänrakennettu yksikkötestaussovelluskehys testien kirjoittamiseen, raportointiin ja virheiden käsittelyyn
- interaktiivinen komentotulkki virheiden paikantamiseen ja suunnitteluideoiden nopeaan kokeiluun
- Pythonin ja testitapausten helppo asennus testiympäristöön
- käyttöjärjestelmäriippumattomuuden ansiosta samat testit voidaan ajaa useissa eri käyttöjärjestelmissä
- helposti toteutettavien C / C++ -laajennusten avulla testattavan moduuleiden rajapintoihin voidaan tehdä Python-testitapauksista kutsuja
- voimallisen syntaksin ja kattavien vakiokirjastojen ansiosta Python-ohjelman pituus on usein merkittävästi lyhyempi kuin vastaava ohjelma esim. C++- tai Java-kielillä kirjoitettuna

Microsoftin tuotteiden kolme arvoa ovat viehättävyys, käytettävyys ja hyödyllisyys. Nämä ovat erittäin tärkeitä ominaisuuksia myös testiautomaatiotyökaluille. Seuraavissa luvuissa kerromme, miten Python täyttää nämä arvot teknisen testaajan, kehitystiimin ja testausarkkitehdin näkökulmista.

Viehättävä työkalu tekniselle testaajalle

Viehättävä testaustyökalu saa teknisesti suuntautuneen testaajan innostumaan. Miltä sinusta tuntuisi testaajana pystyttää esimerkiksi web-palvelin vain kahdella rivillä koodia? Pythonilla se onnistuu seuraavasti:

```
import SimpleHTTPServer
SimpleHTTPServer.test()
```

Tämän jälkeen voisit “surffailla” koneesi tiedostoja selaimellasi osoitteesta ‘http://localhost:8000/’. Aika viileetä, eikö? Itse asiassa voisit myös hakea palvelimesi html-sivujen sisältöä Pythonin avulla esimerkiksi juurihakemistostasi seuraavasti:

```
import urllib
content = urllib.urlopen(”http://localhost:8000/”)
print content.read()
```

Eikö olekin helppoa kuin heinänteko? Testaajana voit käyttää Pythonia helposti myös moniin muihin hyödyllisiin toimintoihin, kuten sähköpostien lähettämiseen ja vastaanottamiseen sekä tietojen etsintään esimerkiksi xml-tiedostoista ja Windowsin rekisteristä.

Monia testaajia viehättävät palkankorotukset ja kollegoiden arvostus. Siksi on hyvä tietää, että Pythonin käyttö antaa uutta pontta palkkaneuvotteluihin. Software Development -lehden (marraskuu 2003) mukaan Pythonia käyttävät ohjelmistoteollisuuden ammattilaiset tienaavat enemmän kuin muita ohjelmointikieliä käyttävät ammattilaiset. Toistaiseksi emme ole huomanneet tällaista ilmiötä palkkapusseissamme, mutta olemme kokeneet saavamme erityistä arvostusta kollegoiltamme. Itse asissa, meidän onnistuneiden kokemuksiemme myötä Pythonin käyttö on jatkuvasti lisääntynyt F-Securella.

Johtajien ja päälliköiden on vaikeampi ymmärtää testausautomaation hyötyjä, jos he eivät pysty näkemään testejä. Vaikka käyttöliittymien kautta tehtävä testien automaattinen suoritus ei yleensä ole optimaalisin tapa, käyttöliittymien testien automatisointia kannattaa pienissä määrin harkita juuri demonstroimista varten. Lisätäksemme työmme näyttävyyttä olemme automatisoineet Pythonilla muutaman käyttöliittymätestitapauksen. Teknisen testaajan on hyvä muistaa, että pomojen hämmästyttämiseen riittää pari testitapausta. Tämä erityisesti siksi, että Python ei ole parhaimmillaan käyttöliittymään kohdistuvassa automaattisessa testauksessa.

Hyödyllinen testityökalu ohjelmistokehitystiimille

Kehitystiimit tuottavat projektin aikana ohjelmistokomponenttien uusia versioita, joiden muutokset mahdollisine sivuvaikutuksineen tuotteessa tulisi varmistaa. Valitettavasti käsin tehtävä regressiotestaus on kallista ja hidasta. Pythonilla toteutettu ohjelmistokomponenttien rajapintatestaus voi nopeuttaa regressiotestaukseen kuluvaa aikaa merkittävästi.

Automaattisella testauksellalöydettyjen vikojen korjaamiseen kuluva aika vähenee, koska Pythonilla kirjoitettujen testitapausten koodia voidaan käyttää virheiden paikallistamiseen. Koska Python-koodi ei vaadi erillistä kääntämistä, testikoodia voidaan muokata suoraan testikoneella ja ajaa testi uudestaan. Lisäksi Pythonin interaktiivista komentotulkkia voidaan käyttää myös käsin tehdyllä testauksella löydettyjen vikojen paikallistamiseen.

Testitapausten ohjelmointi Pythonilla on tehokasta. Kirjoitamme sillä esimerkiksi automaattiset testit lähes kaikkiin tiimimme vastuualueella oleviin muutoksiin uudessa F-Securen Client Security -tuotteessa, koska uskomme voittavamme käytetyn ajan hyvin nopeasti takaisin. Mielestämme Pythonin avulla voidaan rajapintafunktioiden toiminnallisuus, raja-arvot ja poikkeustilanteet kattaa paljon vaivattomammin kuin esimerkiksi Javalla, Perlillä tai C++:lla.

Testaajan tuntemus tuotteesta saattaa rajoittua pitkälti käyttöliittymiin, sillä sisäisten rajapintojen käyttö vaatii esimerkiksi C++-koodin pointtereiden ja kielen monimutkaisen syntaksin tuntemusta. Pelkästään käyttöliittymien kautta tapahtuvalla testauksella ei voi saada tarkkaa kuvaa ohjelman sisäisestä toiminnasta. Pythonilla toteutettu rajapintatestaus kaventaa testaajien ja ohjelmoijien välistä kielimuuria sallimalla testaajan helpomman pääsyn tuotteen sisäisiin rajapintoihin. Esimerkiksi ohjelmoija voi kirjoittaa Pythonilla minimaaliset testitapaukset komponentin tärkeimpiin rajapintafunktioihin ja tämän jälkeen testaaja laajentaa testitapauksia. Tämä toimintatapa lisää mahdollisuuksia löytää ohjelmasta vikoja: testaaja ei tee koodista niin paljon oletuksia kuin ohjelmoija tekee (Kaner, C., Bach, J., Pettichord, B., Lessons Learned from Software Testing - A Context-driven Approach. Wiley Computer Publishing, 2002). Ohjelmoijahan hellii tekemäänsä koodia, toisin kuin (ilkeä) testaaja, joka haluaa rikkoa kaiken käsiin saamansa. Kun testauksella on katettu rajapintafunktioiden tärkeimmät toiminnallisuudet, voi ohjelmoija tehdä melko turvallisesti tarvittavia muutoksia ohjelmakoodiin. Testikoodi toimii lisäksi hyvänä dokumentaationa tuotteen komponentin rajapintafunktioiden eri käyttötavoista ja niiden reunaehdoista.

”Johtajien ja päälliköiden on vaikeampi ymmärtää testausautomaation hyötyjä, jos he eivät pysty näkemään testejä”

Olemme automatisoineet kahden C++-kielellä toteutetun F-Securen ohjelmakomponentin testausta Pythonilla. Ohjelmakomponentit ovat osa laajaa tuoteperhekokonaisuutta. Emme ole ensisijaisesti pyrkineet löytämään uusia vikoja tuotekomponenteistamme, vaan lyhentämään regressiotestaukseen käytettävää aikaa. Tavoitteemme on ollut kattaa tärkeimpiä skenaarioita, joita komponentit tarjovat toisille ohjelmakomponenteille. Voimme varmistaa komponenttien tärkeimmät toiminnallisuudet nopeasti rakentaessamme uuden version. Lisäksi olemme testiympäristöissä korvanneet ulkoisia järjestelmiä simuloimalla niiden testeissä tarvittavat toiminnallisuudet Python-koodilla. Tämä toimintatapa on vähentänyt testiympäristön asentamiseen kuluvaa aikaa ja kohdistanut testauksen pelkästään haluamaamme komponenttiin.

Meillä ei vielä ole käytännön kokemusta Java-koodin testauksesta Pythonin avulla. Pythonista on kuitenkin olemassa Java-kielellä toteutettu sisarversio Jython (ks. http://www.jython.org), jonka avulla Java-koodin testaus on suoraviivaisempaa kuin C++-koodin testaaminen. Jythonilla näet pääsee suoraan käsiksi Java-kirjastojen rajapintoihin ilman ylimääräisten laajennusten kirjoittamista.

```
import urllib2, unittest
class WebServerTest(unittest.TestCase):
    def testNonExistingPage(self):
        try:
            urllib2.urlopen(”http://www.company.com/nonexisting.html”)
            self.fail(”Web page received when expecting error code”)
        except urllib2.HTTPError, error:
            assert error.code == 404, ”Error code received, but a wrong one”

    def testPageTitle(self):
        content = urllib2.urlopen(”http://www.company.com/”)
        for line in content.readlines():
            if ”<TITLE>” in line.upper():
                return
        self.fail(”Title not found in index.html!”)

unittest.main(defaultTest=”WebServerTest”)
```

Käytettävä ja helposti ylläpidettävä työkalu testausarkkitehdille

Kokemuksemme mukaan Python-koodin ylläpito on vaivattomampaa kuin esimerkiksi Perl-kielellä tehdyn koodin. Isohkotkin Python-projektit säilyvät ylläpitokelpoisina, eikä muutosten tekemistä tarvitse välttää koodin rikkomisen pelossa. Oliosuuntautuneisuuden ja selkeän syntaksin lisäksi Pythonissa on mukana unittest-moduuli, jota voidaan käyttää automaattisten testitapausten järkevään ylläpitoon, jaotteluun ja suoritukseen sekä virheiden raportointiin. Alla on yksinkertainen esimerkki unittest-moduulin käytöstä. Esimerkissä testataan aluksi, että yrityksen internet-palvelin lähettää tietyn virhekoodin, kun etsittyä sivua ei löydy. Toisessa testissä varmistetaan, että sivuston pääsivu on olemassa ja että sieltä löytyy haluttu html-elementti. Enempää Python-koodia ei tarvita näiden kahden testitapauksen testaamiseen ja testitulosten raportoimiseen. Koodiesimerkki on muutoksitta ajettavissa esimerkiksi Linuxissa ja Windowsissa.

Python on yleiskäyttöinen työkalu. Sitä kehitetään jatkuvasti ja uusia versioita julkistetaan usein. Sen monipuoliset kirjastot antavat testien kirjoittajalle mahdollisuuden ratkaista mitä erilaisempia testaukseen automatisointiin liittyviä ongelmia. Tässä lyhyessä artikkelissa rajasimme esimerkkitapauksemme yksinkertaisuuden vuoksi http-protokollaan.

Pythonin valmiita kirjastoja on helppo laajentaa omien tavoitteiden mukaisesti. Olemme esimerkiksi rakentaneet unittest-moduulin ympärille oman sovellustestauskehyksen, joka tarjoaa monipuolisemmat testitapausten hallinnointityökalut ja helppokäyttöiset versiot tuotteidemme perusrajapinnoista ja näin soveltuu myös muiden tuotekomponenttitiimien käyttöön.

> ”Pythonilla toteutettu rajapintatestaus kaventaa testaajien ja ohjelmien välistä kielimuuria”

Python-versioiden ylläpito aiheuttaa testausarkkitehdille vain vähän ylläpitotoimenpiteitä. Testausarkkitehdin pitää uutta versiota päivittäessä kääntää uudestaan C++-kielellä kirjoitetut Python-laajennukset. Tämä on kerrallaan vaatinut meiltä vain noin tunnin verran työtä. Python-versioiden välinen yhteensopivuus on hyvä ja niiden välisten muutosten takia olemme harvoin joutuneet tekemään muutoksia itse Python-koodiin.

Houkutteleva Python pähkinänkuoressa

Python on huokutteleva ohjelmointikieli testausautomaatioprojekteihin, koska sillä on laajat käyttömahdollisuudet, alkuunpääseminen on edullista ja testikoodin ylläpito ei yleensä vaadi paljon resursseja. Valmiiden luokkakirjastojen avulla voidaan kirjoittaa nopeasti koodia. Lisäksi kokemustemme mukaan rajapintatestaus Pythonilla lähentää testaajia ja ohjelmoijia, lisää testaajien tuntemusta testattavasta tuotteesta ja nopeuttaa testaussyklejä. Testaustyökalun valintavaiheessa on tärkeää tunnistaa sen menestystekijät ja sopivuus haluttuun käyttöympäristöön. Emme halunneet lähteä tässä artikkelissa kattavasti vertailemaan Pythonia muihin ohjelmointikieliin ja testausautomaatiotyökaluihin. Python on sopivin testaustyökalu käyttötarkoituksiimme. Koska työkalun arviointi on osin tunneasia, jätämme Pythonin arvioinnin ja vertailun muihin työkaluihin omaksi tehtäväksesi.

Marko Komssilla on noin kahdeksan vuoden työkokemus skriptikielien (esim. Pythonin ja Perlin) käytöstä. Hänen intohimoinaan ovat testauksen ja konfiguroinnin automatisointi sekä ohjelmistojen tarkastukset (engl. software inspections).

Mika Elorannalla on yli kymmenen vuoden työkokemus Windows-ohjelmistokehityksestä sekä automaatio- ja testausohjelmistojen kehityksestä. Hänen erikoisosaamisensa on luontaisen laiskuuden ansiosta rutiinityön välttely automaation avulla, jossa Python on ollut pääasiallisena työkaluna kuutisen vuotta.