hyväksymispäivä arvosana

arvostelija

Jatkuva integraatio ja regressiotestaaminen

Piia Hartikka

Helsinki 29.10.2017

Aine

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Piia Hartikka

Työn nimi — Arbetets titel — Title
Jatkuva integraatio ja regressiotestaaminen

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Aine	29.10.2017	8 sivua

Tiivistelmä — Referat — Abstract

Aineessa tarkastellaan jatkuvaa integraatiota ja regressiotestaamista. Jatkuvaa integraatiota harjoittavat ohjelmistokehittäjät integroivat työnsä ohjelmiston pääversioon vähintään kerran päivässä. Jokainen muutos vaarantaa ohjelmiston toimivuuden, mutta jatkuvan integraation tarkoituksena on toteuttaa muutokset mahdollisimman usein ja korjata syntyneet virheet ennen kuin ne kasvavat liian suuriksi. Integraatioon kuuluu ohjelmiston kokoaminen ja regressiotestaaminen erillisellä jatkuvan integraation palvelimella. Regressiotestaaminen keskittyy testaamaan muutoksen vaikutusta ohjelmistoon ja sen oikeellisuuteen. Regressiotestit ovat testikokonaisuus, jonka muodostamiseen on erilaisia tekniikoita. Kokonaisuus keskittyy yleensä testaamaan muutoksen kannalta tärkeitä ohjelmistonosia, sillä jatkuvan integraation tiheätempoisen luonteen vuoksi kaikkia testejä ei ole mahdollista ajaa jatkuvan integraation palvelimella.

ACM Computing Classification System (CCS): A.1 [Introductory and Survey]
I.7.m [Document and Text Processing]: Miscellaneous

Avainsanat — Nyckelord — Keywords
jatkuva integraatio, regressiotestaaminen, regressiotestaus, testaaminen, automatisoitu testaaminen

Säilytyspaikka — Förvaringsställe — Where deposited
Tietojenkäsittelytieteen laitoksen kirjasto, sarjanumero C-2004-X

Muita tietoja — övriga uppgifter — Additional information

Sisältö

1 Johdanto

Jatkuva integraatio korvaa perinteisen ohjelmistokehityksen päättävän integraatiovaiheen. Se on työskentelytapa, jossa ohjelmistokehittäjä integroi työtään vähintään kerran päivässä ohjelmistokehityksen päälinjaan. Ohjelmistokehittäjä aloittaa työskentelynsä hakemalla ohjelmiston uusimman version yhteisestä koodivarastosta ja kommitoi tehdyn ja testatun työn uusimmaksi versioksi saman päivän aikana. Integraatiossa ohjelmisto kootaan ja ohjelmakoodin sisältämät automatisoidut testit ajetaan jatkuvaan integraatioon varatulla palvelimella.

Regressiotestaamisella varmistetaan ohjelmiston oikeellisuus muutoksien jälkeen. Regressiotestaaminen voidaan toteuttaa esimerkiksi valitsemalla kokemuksen perusteella sopiva joukko yksikkö-, integraatio- ja järjestelmätestejä. Kaikkien mahdollisten testien suorittaminen ei ole kannattavaa, sillä se kuluttaa palvelinresursseja ja viivyttää palautteen saamista. Ohjelmiston kokoamisen ja testien ajamisen pitäisi suosituksen mukaan kestää alle kymmenen minuuttia. Testikokonaisuutta voi rajata priorisoimalla testitapauksia tai arvioimalla testien kriittisyyttä. Regressiotestejä kehitetään ajan kuluessa varsinaisen ohjelmakoodin rinnalla.

Tässä tutkielmassa tarkastellaan jatkuvaa integraatiota ohjelmistoon kohdistuvien muutoksien näkökulmasta. Lisäksi käsitellään jatkuvien muutoksien oikeellisuuden tarkistamista regressiotestaamisella ja regressiotestaamisessa käytettävän testikokoelman kokoamista.

2 Jatkuva integraatio

Ohjelmistokehittäjien mielestä jatkuva integraatio on hyvä työskentelytapa, koska he saavat jatkuvan integraation myötä jatkuvasti palautetta työstään. Jatkuvan integraation esteiksi koetaan eniten testikokonaisuuden riittämättömyys sekä testaamisen hitaus ja vaivalloisuus.

Jatkuvan integraation testitoiminnot voidaan jakaa karkeasti kahteen luokkaan: testaaminen ennen integraatiota ohjelmistokehittäjän omalla työkoneella ja jatkuvan integraation palvelimella suoritettava ohjelmiston testaaminen. Ohjelmistokehittäjän on testattava työnsä ennen integraatiota välttääkseen turhaan rikkomasta yhteistä ohjelmakoodia.

2.1 Yksi yhteinen koodivarasto

Jatkuvan integraation myötä samaa ohjelmakoodia muokkaavat kymmenet, sadat tai jopa tuhannet ohjelmistokehittäjät ja heidän kaikkien tulee kommitoida tekemänsä muutokset vähintään kerran päivässä. Ohjelmisto on alati muutoksien kohteena ja ohjelmiston toimivuus ja oikeellisuus vaarantuvat joka muutoksessa. Paras tapa ehkäistä virheiden kasaantumista ja isoja rakenteellisia ongelmia on toteuttaa muutokset ja korjata virheet mahdollisimman tiuhaan.

Testitoimintojen toimivuus on iso osa jatkuvan integraation toimimista ja ohjelmistokehittäjien kokemaa tyytyväisyyttä jatkuvaan integraatioon. Ohjelmistokehitystiimi ei voi työskennellä luotettavasti rikkinäisen ohjelmiston parissa. Siispä jatkuvan integraation palvelimen antama palaute testien läpäisystä toimii vakuutena koko ohjelmistokehitystiimille, että työtä voi jatkaa rauhassa ja se rakentuu toimivan kokonaisuuden päälle. Uudet virheet voidaan paikantaa muutoksen tarkkuudella, eikä kenenkään työtä sotketa turhaan.

2.2 Regressiotestaaminen validoi muutokset

Regressiotestaamista käytetään koko ohjelmiston toimivuuden ja oikeellisuuden testaamiseen muutoksien jälkeen. Ohjelmiston koko ja testien määrä on yleensä suuri, joten ei ole missään määrin järkevää, että kaikki ohjelmiston parissa työskentelevät ohjelmistokehittäjät ajaisivat kaikki olemassaolevat testit päivittäin. Tämän vuoksi regressiotestaamista varten kootaan erikseen testikokonaisuus.

Integraatiotesteillä testataan sisäisesti ohjelmiston eri osien yhteensopivuus. Järjestelmätesteillä ajetaan ulkoisesti testitapauksia, jotka on tehty ohjelmistolle asetettujen vaatimuksien pohjalta. Järjestelmätesteillä saadaan selville ohjelmiston oikeellisuus. Regressiotestien ei tarvitse testata yksikkötesteillä uudelleen yksittäisiä koodinpätkiä, sillä ohjelmistokehittäjät testaavat työnsä huolellisesti työkoneellaan ennen integraatiota. Siitä huolimatta testikokonaisuuteen voidaan lisätä ohjelmiston toimivuuden kannalta tärkeitä yksikkötestejä.

Regressiotestaaminen suoritetaan jatkuvan integraation palvelimella neutraalissa testiympäristössä. Jatkuvan integraation palvelimen on tarkoitus tuottaa puolueeton ja luotettava testitulos, joka vertautuu ohjelmiston käyttämiseen oikeissa tuotantoolosuhteissa.

2.3 Kustannukset

Jatkuva integraatio on kustannustehokkaampi kuin integraatio erillisenä tuotantovaiheena. Jatkuva integraatio lisää ohjelmistotuotantoprosessin läpinäkyvyyttä, kun ohjelmistoon syntyneet virheet tulevat heti koko ohjelmistokehitystiimin tietoon. Pienet virheet ovat nopeampia korjata kuin suuret, mikä vähentää työtunteja ja ehkäisee aikataulun venymistä. Virheiden huomaaminen ennen niiden paisumista isoiksi ongelmiksi ehkäisee rikkonaisen ikkunan syndroomaa, eli ohjelmistokehittäjien taipumusta olla tarttumatta liian suuriin ongelmiin.

Toisaalta testitoimintojen ylläpitämiseenkin kuluu työaikaa. Pelkästään regressiotestien kehittämiseen ja ylläpitoon voi kulua yhtä paljon työaikaa kuin itse ohjelmakoodin kirjoittamiseen ja regressiotestaaminen on vain yksi osa ohjelmiston testaamista. Ohjelmiston lisäksi myös testeissä itsessään voi olla virheitä, joiden tunnistaminen ja korjaaminen voi olla hankalaa. Testitoiminnot voi myös rakentaa osin ulkopuolisen palveluntarjoajan palveluiden varaan, mutta tällöin testejä voi joutua muuttamaan jokaisen palveluntarjoajan tekemän muutoksen myötä.

Jatkuva ohjelmiston kokoaminen ja testien ajaminen voi johtaa palvelinresurssien riittämättömyyteen. Esimerkiksi Googlen testipalvelimella ajetaan päivittäin 800 000 integraatiota. Siellä resurssien riittämättömyys on ratkaistu niin, että palvelin kokoaa ja testaa 45 minuutin välein kertyneet kommitit. Iso osa ajetuista testeistä meni aina läpi, joten niitä päätettin ajaa harvemmin.

3 Testikokoelman muodostaminen

Regressiotestaamisella pyritään ohjelmiston laadun kasvuun ja ohjelmistokehityksen kustannusten vähentämiseen. Regressiotestaamista rajoittavat palvelin- ja henkilöresurssit. Kuten edellisessä luvussa todettiin, kaikkia mahdollisia testejä ei ole yleensä mahdollista tai järkevää ajaa jokaisessa integraatiossa, minkä vuoksi regressiotestaamista varten muodostetaan oma testikokonaisuus. Yleisimmät tekniikat tällaisen testikokonaisuuden muodostamisessa ovat kokonaisuuden minimointi, sopivien testien valinta ja testien ajojärjestyksen priorisointi. Testikokonaisuuden muodostamisessa voi myös käyttää useampia tekniikoita ja tekniikoita voi myös vaihdella testiajojen mukaan.

3.1 Tekniikat

Testikokonaisuuden minimointi edellyttää varmuutta siitä, että minimoitu testikokonaisuus todella riittää ohjelmiston testaamiseen ja virheiden löytämiseen. Eräs minimointimenetelmä on tarkastella muutoksen vaikuttavuutta ohjelmistossa ja testata vain näitä osia.

Testikokonaisuuden muodostaminen valinnalla voidaan toteuttaa käsin tai ohjelmallisesti. Kummallakin menetelmällä on tarkoitus valita asiantuntemuksen perusteella relevanteimmat testit. Esimerkiksi käyttäytymiseen perustuva regressiotestaaminen (behavioral regression testing) on menetelmä, jolla ohjelmistokehittäjien valitsemien regressiotestien lisäksi käytetään itsestään generoituvia testejä. Niillä pyritään varmistamaan ohjelmiston oikeellisuuden lisäksi myös sitä, että oikeellisuus tuotetaan tarkoitetulla tavalla.

Testien priorisoiminen tarkoittaa sitä, että valitaan mitkä testitapaukset ajetaan ensin. Näin testaamisesta syntyvä palaute saadaan nopeammin tärkeistä testeistä ja

palaute saadaan silloinkin, kun ohjelmiston testaaminen keskeytyy esimerkiksi virheen tai muistin loppumisen myötä. Testitapausten priorisointijärjestys tulee suoraan ohjelmiston vaatimusten priorisointijärjestyksestä.

3.2 Historiatietojen kerääminen

Erilaisia historiatietoja voi käyttää testikokonaisuuden muodostamisen tukena tai pohjana. Esimerkiksi toistuvasti hylkäävien testien voidaan olettaa testaavan ohjelmiston hauraita osia, jolloin jatkossa niitä testejä voidaan ajaa useammin. Samoin jatkuvasti läpi menevät testit voivat olla turhia ja niitä ei tarvitse ajaa joka integraatiossa.

Testidatan kulkureitistä ja läpikäymistä prosesseista kerättävä metadata (provenance data) on monikäyttöistä. Metadatan keräämisestä ja analysoimisesta on hyötyä regressiotestaamisen optimoinnissa ja lisäksi tekniikkaa käytetään tieteellisissä kokeissa. Metadataa kerätään yleensä joko tarkoituksena tuottaa tietoa tulevaa testaamista varten tai yksityiskohtia edellisistä testiajoista.

Sekä testiajojen historiasta että metadatan analysoimisesta voi muodostaa parempaa palautetta ohjelmistokehittäjille. Esimerkiksi ohjelmistokehittäjä voisi saada tilastotietoja omista integraatioistaan ja yksityiskohtaisia kehitysehdotuksia parantamaan integraatioiden onnistumisprosenttia. Lisäksi integraation onnistumiseen ja epäonnistumiseen liittyvien tekijöiden tunnistaminen tehostaa ohjelmistokehitystä.

4 Yhteenveto

Eräs jatkuvan integraation harjoittamisen haasteista on ohjelmistoon kohdistuvien muutosten hallinta. Ohjelmiston kehitystiimin jokainen jäsen integroi työnsä vähintään päivittäin ohjelmistokehitystiimin yhteiseen ohjelmiston pääversioon, mutta samalla jokainen muutos on uhka ohjelmiston toimivuudelle.

Regressiotestaaminen validoi integroinnissa lisätyt muutokset. Regressiotestaamisessa käytetään omaa testikokonaisuutta, jolla pyritään tehokkaasti varmistamaan, että ohjelmisto toimii muutoksen jälkeen kuten pitääkin. Testikokonaisuus muodostetaan minimoimalla, valinnalla tai rajaamalla ja näitä voi myös yhdistellä. Menetelmiin kuuluu niin ohjelmistokehittäjän kokemukseen perustuva arviointi tärkeistä testeistä kuin ohjelmallisesti toteutetut testigeneraattoritkin. Testikokonaisuuden kokoa ja ajoaikaa rajoittavat palvelinresurssit sekä testikokonaisuuden ylläpitoon kuluvat henkilöresurssit.

Regressiotestaamisesta kannattaa myös kerätä historiatietoja. Esimerkiksi testien läpimenotilastot voivat paljastaa hauraita ohjelmistonosia ja toisaalta aina läpi meneviä testejä, joita ei välttämättä tarvitse ajaa joka kerta. Testidatan kulkua ohjelmiston läpi voidaan seurata keräämällä metadataa ja sen perusteella voidaan sekä kehittää testikokonaisuutta että saada arvokasta tietoa ohjelmiston toiminnasta.

Jatkuva integraatio vähentää ohjelmistokehityksen riskejä, kun ohjelmisto altistetaan riskialttiille muutoksille monta kertaa päivässä. Riskit realisoituvat ja niihin reagoidaan nopeasti, jolloin lopputuloksena on ajantasainen ja läpinäkyvä ohjelmisto. Regressiotestaamisen rooli jatkuvassa integraatiossa on tärkeä, sillä se validoi integraatiot ja sen seurauksena virheet on helppo jäljittää yksittäiseen integraatioon.

Lähteet

ERP14 Elbaum, S., Rothermel, G. ja Penix, J., Techniques for improving regression testing in continuous integration development environments. New York, NY, USA, 2014, ACM, sivut 235−245, URL http://doi.acm.org/10.1145/2635868.2635910.

LIH17 Labuschagne, A., Inozemtseva, L. ja Holmes, R., Measuring the cost of regression testing in practice: A study of java projects using continuous integration. New York, NY, USA, 2017, ACM, sivut 821−830, URL http://doi.acm.org/10.1145/3106237.3106288.

MGN+17 Memon, A., Gao, Z., Nguyen, B., Dhanda, S., Nickell, E., Siemborski, R. ja Micco, J., Taming googlescale continuous testing. Piscataway, NJ, USA, 2017, IEEE Press, sivut 233−242, URL https://doi.org/10.1109/ICSE$\-$SEIP.2017.16.

VW16 Vöst, S. ja Wagner, S., Tracebased test selection to support continuous integration in the automotive industry. New York, NY, USA, 2016, ACM, sivut 34−40, URL http://doi.acm.org/10.1145/2896941.2896951.