hyväksymispäivä arvosana

arvostelija

Jatkuva integraatio ja regressiotestaaminen

Piia Hartikka

Helsinki 8.10.2017

Aine

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

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

2 Jatkuva integraatio

2.1 Yksi yhteinen koodivarasto

Jatkuvan integraation myötä samaa ohjelmakoodia muokkaavat kymmenet, sadat tai jopa tuhannet ohjelmistokehittäjät. Heidän kaikkien tulisi kommitoida tekemänsä muutokset vähintään kerran päivässä, joten ohjelmisto muuttuu alati. Ohjelmiston toimivuus ja oikeellisuus vaarantuvat muutoksessa. On huomattu, että paras tapa ehkäistä virheiden kasaantumista ja isoja rakenteellisia ongelmia on toteuttaa muutokset ja korjata virheet mahdollisimman tiuhaan. Kun kaikki ohjelmiston kehittämiseen osallistuvat ohjelmistokehittäjät työskentelevät yhteisen ohjelmakoodin parissa, ohjelmiston uusimman version toimimattomuudesta tulee yhteinen ongelma.

Ohjelmistokehittäjät kokevat, että jatkuva integraatio on hyvä työskentelytapa. He saavat jatkuvan integraation myötä jatkuvaa palautetta työnsä toimivuudesta, eikä parempaakaan tapaa ole. Jatkuvan integraation esteiksi koetaan eniten ohjelmiston testaamisen osa-alueita, kuten testikokonaisuuden riittämättömyys tai testaamisen nopeus ja vaivattomuus.

Ohjelmistokehitystiimi ei voi työskennellä luotettavasti rikkinäisen ohjelmiston parissa. Jatkuvan integraation palvelimen antama palaute testien läpäisystä toimii vakuutena koko ohjelmistokehitystiimille. Näin he voivat luottaa rakentavansa työnsä toimivan kokonaisuuden päälle ja uudet virheet voidaan paikantaa kommitin tarkkuudella.

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 yh-

teistä ohjelmakoodia.

2.2 Regressiotestaaminen validoi muutokset

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

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

Käyttäytymiseen perustuva regressiotestaaminen (behavioral regression testing) on eräs menetelmä, jossa regressiotestien lisäksi käytetään itsestään generoituvia testejä. Niillä pyritään testaamaan, käyttäytyykö ohjelmisto muutoksien jälkeen toivotulla tavalla. <sisäisesti, ulkoisesti vai molempia???>

<Tarkista, onko muita menetelmiä. Varmasti on. Tilaa on vielä lyhyesti selittää muutamia juttuja ja tästä aliluvusta voi hyvin supistaa tarvittaessa.>

2.3 Kustannustehokkuus

Jatkuva integraatio on kustannustehokkaampi kuin integraatio erillisenä tuotantovaiheena. Jatkuva integraatio lisää ohjelmistotuotantoprosessin läpinäkyvyyttä, eli

ohjelmistoon syntyneet virheet tulevat heti koko ohjelmistokehitystiimin tietoon. Pienet virheet on suuria nopeampi korjata, mikä vähentää työtunteja ja ennen kaikkea 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. Testitoiminnot voi myös rakentaa osin ulkopuolisen palveluntarjoajan palveluiden varaan. Niiden toiminta ja rajapinnat voivat muuttua, jolloin testejä pitää vastaavasti muuttaa.

Jatkuva ohjelmiston kokoaminen ja testien ajaminen voi johtaa palvelinresurssien riittämättömyyteen. Esimerkiksi Googlen testipalvelimella ajetaan päivittäin kaksi miljoonaa integraatiota. Siellä resurssien riittämättömyys on ratkaistu niin, että palvelin ajaa kokoamiset ja testaamiset muutaman kerran tunnissa, jolloin osa jonoon kertyneistä testeistä on samoja ja ne voidaan jättää ajamatta. <Ratkaisu oli laajempi, etsi se ja kirjoita tähän.>

Regressiotestaamisen testitulokset voi jakaa kahteen luokkaan: hyödyllisiin ja haitallisiin. <Nämä liittyivät jotenkin siihen, että on hyödyllistä ja kustannustehokasta löytää virheitä jatkuvan integraation myötä, mutta tietoisuus arkkitehtuurin pieleen menosta tms. lisäävät kustannuksia. Kuulostaa omituiselta, joten etsi lähde.>

3 Testikokoelman priorisointi ja rajaaminen

3.1 Rajalliset resurssit ja laatutekijät

Regressiotestaamisella pyritään ohjelmiston laadun lisäämiseen ja ohjelmistokehityksen kustannusten vähenemiseen. Regressiotestaamista rajoittavat resurssien rajallisuus, kuten palvelinresurssit ja aikataulu. Miten löytää pienin ohjelmaa testaava testikokonaisuus.

<Miten resurssit ja laatutekijät ohjaavat testikokoelman priorisointia ja rajaamista, esittele lyhyesti eri tapoja.> <software change impact analysis (= analyysi siitä, mihin osiin muutokset vaikuttavat) on ilmeisesti priorisointiin auttava väline, mistä oli muistaakseni paperi siellä konferenssissa> <On hyvä pitää listaa testeistä, jotka eivät ole menneet läpi. Listaa voidaan käyttää myöhemmin herkästi särkyvän(fragile) koodin ennustamiseen.>

3.2 Testidatan seuraaminen

Testidatan liikkeistä voidaan regressiotestien suorituksen yhteydessä kerätä metadataa.

<Mitä reittiä testidata kulkee testin suorituksen aikana. Onko reitti odotettu. Metadataa voi myös analysoida ja analyysin perusteella tehdä testikokonaisuutta koskevia valintoja>

<Käytettävän testidatan valinta on tärkeässä roolissa, kuten testaamisessa yleensäkin. Onko se tämän aineen kannalta oleellista?>

4 Yhteenveto

Tekstiä.

Lähteet

dSCJdPB+17 de S. Campos Junior, H., de Paiva, C. A., Braga, R., AraÃºjo, M. A. P., David, J. M. N. ja Campos, F., Regression tests provenance data in the continuous software engineering context. New York, NY, USA, 2017, ACM, sivut 10:1–10:6, URL http://doi.acm.org/10.1145/3128473.3128483.

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.