hyväksymispäivä arvosana

arvostelija

Jatkuva integraatio

Piia Hartikka

Helsinki 14.11.2017

Kandidaatintutkielma

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

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	14.11.2017	13 sivua

Tiivistelmä — Referat — Abstract

Jatkuva integraatio on työskentelytapa, jossa ohjelmistokehittäjät inegroivat työnsä ohjelmiston pääversioon vähintään kerran päivässä. Sen tarkoituksena on toteuttaa ohjelmistotuotannon riskialttiit integraatiot mahdollisimman usein, jotta virheet huomataan ja korjataan ennen kuin ne kasvavat liian suuriksi. Näin ohjelmistotuotannon projektit ovat ennustettavampia. Integraatioon kuuluu ohjelmiston kokoaminen erillisellä jatkuvan integraation palvelimessa. Kokoamisen yhteydessä suoritetaan regressiotestaaminen, joka 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. Regressiotestaamisen kustannustehokkuus onkin jatkuvan integraation ydinkysymyksiä.

ACM Computing Classification System (CCS): Software and its engineering

Software creation and management

Software verification and validation

Avainsanat — Nyckelord — Keywords
jatkuva integraatio, regressiotestaaminen

Säilytyspaikka — Förvaringsställe — Where deposited

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. [Fow06]

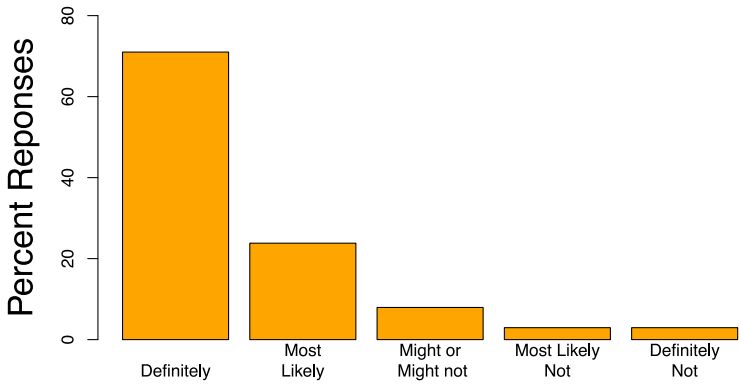
Regressiotestaamisella varmistetaan ohjelmiston oikeellisuus muutoksien jälkeen. Regressiotestaaminen voidaan toteuttaa esimerkiksi valitsemalla kokemuksen perusteella sopiva joukko yksikkö-, integraatio- ja järjestelmätestejä. LÄHDE Kaikkien mahdollisten testien suorittaminen ei ole kannattavaa, sillä 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. [LIH17]

Tämän kandidaatintutkielman luvussa yksi arvioidaan jatkuvaa integraatiota. Luvussa kaksi käsitellään regressiotestaamisen kustannustehokkuutta ja

toteutustapoja.

2 Jatkuva integraatio

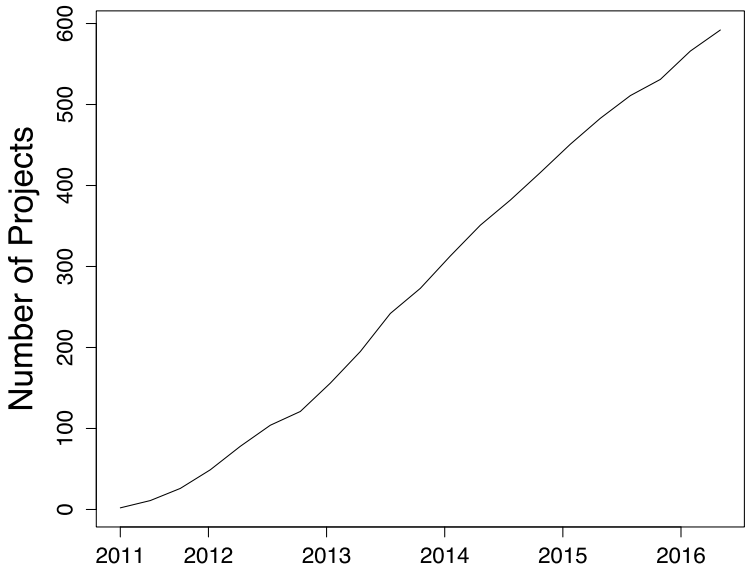
Jatkuva integraatio on suosittu työskentelytapa ohjelmistoteollisuuden [MSB17] ja avoimen lähdekoodin projekteissa. [HTH+16] Suurissa, tuhansia ohjelmistokehittäjiä työllistävissä yrityksissä työskentelevät ohjelmistokehittäjät suhtautuvat erittäin myönteisesti jatkuvaan integraatioon antaen asteikolla 1-5 arvosanaksi 4,13. [MSB17] Avoimen lähdekoodin projekteissa työskentelevät ohjelmistokehittäjät käyttäisivät mielellään jatkuvaa integraatiota seuraavassa projektissaan, mikä näkyy kuvassa 1. Mitä suositumpi projekti, sitä todennäköisemmin projektia tehdään jatkuvasti integroiden. Jatkuva integraatio myös alentaa kynnystä ottaa osaa avoimen lähdekoodin projektiin. [HTH+16] Suurin osa ohjelmistokehittäjistä integroi työnsä suositusten mukaisesti vähintään kerran päivässä. Tämä tahti nähdään sopivana sekä omassa että toisten ohjelmistokehittäjien työssä. [MSB17]



Kuva 1: Käyttäisivätkö avoimen lähdekoodin projekteissa työskentelevät ohjelmistokehittäjät jatkuvaa integraatiota seuraavassa projektissaan. [HTH+16]

Avoimen lähdekoodin projektien keskuudessa jatkuvan integraation käyttö on kasvanut jyrkästi viime vuosina. Ilmiö on nähtävissä kuvassa 2, josta käy ilmi jatkuvan integraation huimasti kasvanut suosio. Trendikkyydestä huolimatta jatkuvan integraation suosion ei odoteta laskevan tulevaisuudessa. [HTH+16] Ohjelmistokehittä-

jät itse näkevät jatkuvan integraation tärkeänä ja hyödyllisenä työskentelytapana. Toisaalta toteutus koetaan hankalana, kuten erityisesti testikokonaisuuden kokoaminen regressiotestaamista ja testiympäristöjä silmälläpitäen. [MSB17] Kaikki ohjelmistokehittäjät eivät tunne jatkuvaa integraatiota tarpeeksi ja yliopistot voisivatkin opettaa sitä enemmän. [HTH+16]



Kuva 2: Avoimen lähdekoodin projektit, joissa käytetään jatkuvaa integraatiota. [HTH+16]

Jatkuvasta integraatiosta on kiistatta hyötyä [HTH+16], mutta sen saama suitsutus on ylimitoitettua. [SB13] Hyvä esimerkki tästä on ohjelmistokehittäjien taipumus vältellä koontiversion (build) testien epäonnistumista. Huolellisuus on hyve, mutta ei aina kustannustehokasta, mikäli työaikaa kuluu turhaan. [HTH+16] Ohjelmistokehittäjät noudattavat jatkuvan integraation tarkkoja käytänteitä mielellään, jos se on järkevää. Esimerkiksi modulaarisen arkkitehtuurin parissa työskentelevät ohjelmistokehittäjät eivät koe tarvetta integroida omaa työtään ohjelmiston versionhallinnan päälinjaan, vaan he perustavat sinne oman sivuhaaran (branch). [MSB17]

2.1 Koontiversio

Integraatiossa ohjelmisto kootaan, jolloin ohjelmakoodin sisältämät automatisoidut testit ajetaan jatkuvan integraation palvelimella. Palvelin antaa palautetta koonnin ja testaamisen onnistumisesta. [Fow06] Koontiversion epäonnistuessa ohjelmistokehitystiimin kuuluu tehdä välittömästi toimenpiteitä, joilla koontiversion status muuttuu onnistuneeksi. [LGL+16] Tämä on tosin ristiriidassa käytännön havaintojen kanssa, joiden mukaan koontiversion statuksesta ja laadusta ei viestitä tarpeeksi. [SB13]

Testaaminen ennen integraatiota on tärkeää, jotta koontiversion status pysyy onnistuneena. [ERP14] Näin ohjelmistokehittäjät voivat luottaa työskentelevänsä toimivaksi testatun ohjelmiston kanssa. [MSB17] Turvallisuuskriittisissä ohjelmistoissa testaaminen on erityisen tärkeää ja tällaiseen ohjemistoon kohdistuvat muutokset ovat erilaisia kuin esimerkiksi web-applikaatioissa. [LGL+16] Ketterän testaamisen harjoittaminen korreloi jatkuvan integraation kanssa, mutta kausalisaatiota tuskin esiintyy [SB13], vaikka niin väitetäänkin. [LGL+16] Varmaa on, että jatkuva integraatio tukee virheiden löytämistä ja helpottaa versioiden yhteenliittämistä (merge). [SB13]

Koontiversion epäonnistumiseen johtava tärkein ja hyödyllisin syy on virhe ohjelmistossa. Kun koontiversio epäonnistuu, virhe löydetään ja se voidaan korjata. Muut epäonnistumiseen johtavat syyt sen sijaan hidastavat jatkuvaa integraatiota ja aiheuttavat kustannuksia ohjelmistotuotannossa. [LIH17] Nämä ovat sekä teknisiä että sosiaalisia syitä, kuten riittämättömät ja virheelliset testit tai liian tiukka aikataulu. Ohjelmistokehittäjät myös luottavat liikaa testien virheettömyyteen. [PRC17] Siispä testaaminen ja erityisesti integraatiossa suoritettava regressiotestaaminen on isoin ja tärkein jatkuvan integraation osa-alue, jonka tarkastelua jatkuvan integraation yhteydessä ei voi jättää tekemättä. [ERP14]

<Tähän KOONTIVERSION EPÄONNISTUMINEN>

2.2 Hyvät ja huonot puolet

Jatkuvan integraation tärkein hyöty on virheiden löytyminen aikaisin [PRC17] [HTH+16], minkä vuoksi projektien aikataulu on helpommin ennustettavissa. [SB13] Sen omaksuminen nopeuttaa ohjelmistotuotantoprosessia ja mahdollistaa ripeämmän julkaisusyklin. [PRC17] Lisäksi avoimen lähdekoodin projekteissa puoltopyynnöt (pull request) hyväksytään nopeammin. [HTH+16] Jatkuva integraatio lisää sekä sisäistä että ulkoista viestintää, mikä mahdollistaa rinnakkaisen ohjelmistokehityksen. [SB13]

Testaaminen on olennainen osa jatkuvaa integraatiota [LGL+16], mutta testien kirjoittamiseen ja ylläpitämiseen menee jopa yhtä paljon työaikaa kuin itse ohjelmakoodin kirjoittamiseen. [LIH17] Lisäkustannuksia voi tulla projektista riippuen ostopalveluista ja välineistä. Testeihin myös luotetaan liikaa, vaikka ne voivat olla epädeterministisiä tai kattavuudeltaan huonoja. [PRC17]

Jatkuva integraatio vaatii ohjelmistokehittäjiltä itsekuria, ainakin aluksi. Työyhteisöllä tulee olla hyvä testaamiskulttuuri ja käytänteet on tehtävä selväksi myös uusille työntekijöille. [PRC17] Kuten aiemmin käytiin läpi, ohjelmistokehittäjät myös välttelevät liikaa koontiversion testien epäonnistumista. [HTH+16] Ohjelmistokehittäjät toteuttaisivatkin integraatiot useammin, mikäli työ olisi pilkottavissa ja järkevästi ajoitettu. Ohjelmiston arkkitehtuurilta toivotaan modulaarisuutta ja testaamisessa työkalujen ja prosessien olisi hyvä olla nopeita ja helppoja. Haasteena olisi siis tehdä ohjelmistokehittäjille ympäristö ja olosuhteet, joissa he voisivat helpommin toteuttaa jatkuvaa integraatiota. Taulukossa 1 näkyy teemakartoitus ohjelmistokehittäjien esteistä kommitoida päivittäin ohjelmiston versionhallinnan päälinjaan. [MSB17]

Jatkuva integraatio on työskentelytapana sellainen, että se rytmittää ohjelmisto-

	Total	Case study A	Case study B
Activity planning and execution	19	10	9
- Work breakdown	15	7	8
- Teams and responsibilities	6	1	5
- Activity sequencing	13	6	7
System thinking	17	8	9
- Modular and loosely coupled architecture	12	5	7
- Developers must think about the complete system	8	5	3
Speed	19	9	10
- Tools and processes that are fast and simple	15	8	7
- Availability of test environments	9	2	7
- Test selection	7	1	6
- Fast feedback from the integration pipeline	9	5	4
Confidence through test activities	16	9	7
- Test before commit	9	6	3
- Regression tests on the mainline	9	5	4
- Reliability of test environments	6	3	3

Taulukko 1: Teemakartoitus ohjelmistokehittäjien esteistä kommitoida päivittäin ohjelmiston versionhallinnan päälinjaan. [MSB17]

kehittäjän arkea. [Fow06] Tämän vuoksi jatkuvan integraation prosessien vaivattomuudella on iso vaikutus ohjelmistokehittäjän tyytyväisyyteen ja halukkuuteen toteuttaa jatkuvaa integraatiota. [MSB17] Esimerkiksi automatisoitu testaaminen ja neutraalin jatkuvan integraation palvelimen käyttäminen testaamiseen helpottaa virheiden löytymistä [PRC17], mutta silti ohjelmiston kehitys- ja testaamisvälineet eivät aina ole yhteensopivia [LGL+16] ja jatkuvan integraation palvelimen configuroiminen on hankalaa. [PRC17] Jatkuva integraatio voi myös hämärtää eri testitoimintojen rajoja. [LGL+16]

2.3 Case: Jatkuva integraatio Googlella

Googlella jatkuvan integraation palvelimen asemassa toimii testialusta TAP, joka tulee sanoista Test Automation Platform. TAP suorittaa päivittäin 800 000 koontiversiota, mikä on suuruusluokkaa yksi per sekunti. Googlella on huomattu, että

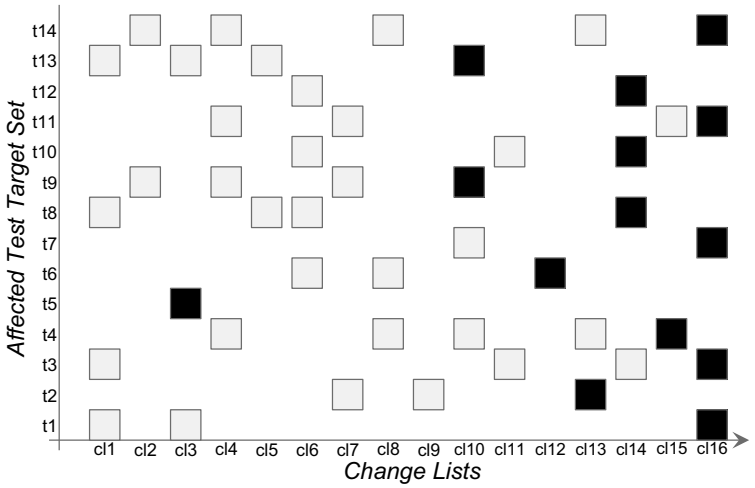
jatkuvan integraation vaatimat resurssit kasvavat nelinkertaisesti suhteessa aikaan, sillä sekä lähetysmäärä että testipohjan koko kasvavat lineaarisesti. [MGN+17]

Kuten nimestä voikin päätellä, jatkuva integraatio on Googlella testikeskeistä.[MGN+17] TAP:n vastuulla on sekä pre- että post-testaaminen ja post-testaamiseen pääsee vain jos pre-testaaminen onnistui. [ERP14] Kommitien määrä ja koodipohjan koko ovat valtavia, jonka vuoksi kaikkea ei voida testata ja toisaalta koodipohjan analysointiin perustuvien työkalujen käyttöäkään ei voi tehdä. [MGN+17] Ohjelmistokehittäjät kirjoittavat itse testit ja kokoavat testikokonaisuuden, mutta testaamisessa ja testi-infran ylläpidossa on oma tiiminsä. Kaikissa testikohteissa käytetään XUnit-tyyppistä testikehystä, mikä mahdollistaa testaamisen keskittämisen, kun kaikkia testikohteita voidaan käsitellä samalla tavalla. [ERP14]

Ohjelmistokehittäjät liittävät TAP:lle lähetettävään kommitiin muutoslistan (changelist), joka sisältää tiedon muutoksien sijainnista. Googlen koodipohja on jaettu paketteihin (package), joita vastaavat itsenäisesti koottavat testikohteet (test-target). TAP aloittaa kommitin käsittelyn muutoslistasta, jonka perusteella etsitään myös muut testikohteet, joihin on riippuvuuksia tai jotka riippuvat muuttuneista kohteista. [MGN+17] [ERP14] Näitä riippuvuuskohteita tosin approksimoidaan rankalla kädellä. [ERP14]

TAP ei aja testikohteita heti, vaan se odottaa seuraavan virstanpylvään (milestone) kulumista. Yksi virstanpylväs on kolme varttia. Kuvassa 3 on esitelty kommitien kerääntyminen yhden virstanpylvään aikana. Kun koontiversiot suoritetaan, jokaisesta testikohteesta ajetaan ainoastaan viimeisin kommit. Tämä säästää aikaa ja resursseja, vaikka toisaalta ohjelmistokehittäjä voi joutua odottamaan testituloksia täyden virstanpylvään verran. Lisäviipeitä aiheuttavat esimerkiksi muistin loppuminen, ohjelman kaatuminen tai laitteisto-ongelmat. [MGN+17] Testitulokset ilmoitetaan testikohteen tarkkuudella. [MGN+17] [ERP14] Ne ilmoitetaan kommitin lähettäneelle ohjelmistokehittäjälle ja hänen tulee korjata epäonnistuneet kohteet välittömästi.

[ERP14]



Kuva 3: Kommitien kerääntyminen yhden virstanpylvään aikana. Joka testikohteesta ajetaan vain viimeisimmät kommitit, jotka näkyvät mustina neliöinä. [MGN+17]

3 Regressiotestaaminen jatkuvassa integraatiossa

3.1 Kustannustehokkuus

4 Yhteenveto

Jatkuva integraatio on kiistatta hyvä ja tehokas tapa tuottaa laadukasta ohjelmakoodia. Kuitenkin jatkuvan integraation sujuva toteutus vaatii henkilö-, palvelin- ja teknologiaresursseja. Hankaluudet testaamisessa, integraatiossa tai työn pilkkomisessa heikentävät jatkuvan integraation onnistumista. Ohjelmistokehittäjät arvostavat jatkuvaa integraatiota, joten se kannattaa tehdä heille helpoksi.

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 ohjelmis-

to. 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. *FSE 2014 Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, 2014, ACM, sivut 235−245, URL http://doi.acm.org/10.1145/2635868.2635910.

Fow06 Fowler, M., Continuous integration, May 01 2006.

HTH+16 Hilton, M., Tunnell, T., Huang, K., Marinov, D. ja Dig, D., Usage, costs, and benefits of continuous integration in opensource projects. *TITLE*, New York, NY, USA, 2016, ACM, sivut 426−437, URL http://doi.acm.org/10.1145/2970276.2970358.

LGL+16 Li, N., Guo, J., Lei, J., Li, Y., Rao, C. ja Cao, Y., Towards agile testing for railway safetycritical software. *XP16 Workshops Proceedings of the Scientific Workshop Proceedings of XP2016*, New York, NY, USA, 2016, ACM, URL http://doi.acm.org/10.1145/2962695.2962713.

LIH17 Labuschagne, A., Inozemtseva, L. ja Holmes, R., Measuring the cost of regression testing in practice: A study of java projects using continuous integration. *TITLE*, 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. *ICSESEIP '17*, Piscataway, NJ, USA, May 2028 2017, IEEE Press, sivut 233−242, URL https://doi.org/10.1109/ICSE$\-$SEIP.2017.16.

MSB17 Mårtensson, T., Ståhl, D. ja Bosch, J., Continuous integration impediments in largescale industry projects. *TITLE*. IEEE, April 2017, sivut 169−178.

PRC17 Pinto, G., Rebouças, M. ja Castor, F., Inadequate testing, time pressure, and (over) confidence: A tale of continuous integration users. *TITLE*, Piscataway, NJ, USA, 2017, IEEE Press, sivut 74–77, URL `https://doi.org/10.1109/CHASE.2017.13`.

SB13 Ståhl, D. ja Bosch, J., Experienced benefits of continuous integration in industry software product development: A case study. *JOURNAL*.