Mikko Hepola

Jatkuva integraatio ja testaaminen ohjelmistokehityksessä: systemaattinen kirjallisuuskartoitus

Tietotekniikan pro gradu -tutkielma

22. joulukuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Mikko Hepola

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Jatkuva integraatio ja testaaminen ohjelmistokehityksessä: systemaattinen kirjallisuuskartoitus

Title in English: Continuous Integration and Testing in Software Development: A Systematic Mapping Study

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmisto- ja tietoliikennetekniikka

Sivumäärä: 45+3

Tiivistelmä: Jatkuva integraatio on laajasti käytetty käytäntö, jolla pyritään nopeaan ja laadukkaaseen ohjelmistokehitykseen. Jatkuvan integraation keskeinen periaate on varmistaa ohjelmistomuutosten soveltuvuus olemassa olevan toteutuksen kanssa, jotta voidaan varmistua sovelluksen toimivuudesta myös muutosten jälkeen.

Tässä tutkimuksessa tutkittiin, millaista tutkimusta on tehty jatkuvaan integraation liittyvästä testaamisesta. Systemaattisen kirjallisuuskartoituksen avulla tutkimukseen valikoitui 52 artikkelia joiden avulla vastattiin tutkimuskysymykseen: Mikä on jatkuvaan integraatioon liittyvän testaamisen tutkimuksen nykytila?

Aineistosta havaittiin kaksi kategoriaa, joista toisessa painotettiin erityisesti jatkuvaa integraatiota ja toisessa testaamista. Tutkimuksessa mukana olevien artikkelien perusteella jatkuvaan integraatioon liittyvän testaamisen tutkimus vaikuttaa monipuoliselta ja jatkuvan integraation käytännön periaatteista näyttää olevan yksimielisyys.

Yleisesti jatkuva integraatio nähtiin hyödyllisenä, sillä se säästää aikaa ja parantaa virheiden havaitsemista. Osassa tutkimuksista pyrittiin löytämään vaihtoehtoja testien ajamiselle jatkuvassa integraatiossa, sillä haasteena nähtiin testien ajamisen kesto ja kustannukset varsinkin isommissa projekteissa.

Avainsanat: Jatkuva integraatio, testaustasot

Abstract: Continuous integration is a widely employed practice aimed at achieving rapid and high-quality software development. The core principle of continuous integration is to ensure the compatibility of software changes with the existing implementation, thereby confirming the functionality of the application even after modifications.

This study investigated the research conducted on testing related to continuous integration. Through a systematic mapping study, 52 articles were selected to address the research question: What is the current state of research on testing related to continuous integration?

Two main categories were identified from the data, with one emphasizing continuous integration and the other focusing on testing. Based on the articles included in the study, research on testing related to continuous integration appears diverse, and there seems to be consensus on the practical principles of continuous integration.

In general, continuous integration was perceived as beneficial because it saves time and improves error detection. Some studies aimed to find alternatives for running tests in continuous integration, as the challenge was seen in the duration and costs of running tests, especially in larger projects.

Keywords: Continuous integration, test levels

Kuviot

Taulukot

Sisällys

1 Johdanto

Jatkuva integraatio on laajasti ohjelmistokehityksessä käytetty laadunvarmistukseen pyrkivä käytäntö, jonka avulla pyritään nopeampaan ja laadukkaampaan ohjelmistokehitykseen (Shahin, Ali Babar ja Zhu 2017, 3910). Siinä sovelluskehittäjien tekemät muutokset integroidaan olemassa olevien muutosten kanssa, jotta voidaan varmistua niiden yhteensoveltuvuudesta mahdollisimman aikaisessa vaiheessa (Fowler 2006). Käytäntö sisältää koontiversion ajon sekä testauksen (Shahin, Ali Babar ja Zhu 2017, 3910).

Testauksen ollessa yksi tärkemmistä tekijöistä sovellusten laadunvarmistamiseksi, on myös jatkuvasta integraatiosta tullut laajalti käytetty testauksen standardi, jonka avulla kehittäjät pystyvät seuraamaan sovelluksen laatua tehokkaaasti kehitysprojektin aikana (Tronge ym. 2021, 1136). Jatkuvan integraation hyötyinä nähdään muun muassa riskien vähentäminen, projektien näkyvyyden ja ennustettavuuden parantaminen, parempi luottamus sovellukseen ja helpommin löydettävät virheet (Soares ym. 2022, 2).

Vaikka jatkuvan integraation tutkimus onkin lisääntynyt viime vuosina, Soares ym. (2022) esittävät, että jatkuva integraatio voi tuoda myös teknisiä haasteita sovelluskehittäjille ja nykyiset tutkimukset näyttävät jatkuvan integraation lähinnä positiivisessa valossa. Heidän mielestään jatkuvan integraation vaikutusten arvioinnissa olisi parannettavaa. Esimerkkinä tällaisesta haasteesta, jonka vaikutuksia voisi arvioida lisää on testien ajaminen, jonka Ali ym. (2019) näkevät vievän aikaa ja aiheuttavan kustannuksia.

Tämän tutkielman tavoitteena onkin selvittää systemaattisen kirjallisuuskartoituksen avulla, minkälaista tutkimusta jatkuvaan integraation liittyvästä testaamisesta on tehty, minkälaisia käytäntöjä jatkuvan integraation yhteydessä tehtävään testaukseen on sovellettu, sekä minkälaisia haasteita tutkimukseen valikoiduissa artikkeleissa on havaittu jatkuvaan integraatioon tai testaamiseen liittyen.

Tavoitteen mahdollistamiseksi tutkimuskysymys on asetettu seuraavasti:

- Mikä on jatkuvaan integraatioon liittyvän testaamisen tutkimuksen nykytila?

Tutkimuskysymykseen pyritään vastaamaan seuraavien alakysymysten avulla:

- Minkälaisia työkaluja jatkuvan integraation toteutuksessa on käytetty?
- Kuinka paljon tutkimuksia on julkaistu?
- Minkälaisia testaustasoja jatkuvan integraation yhteydessä on käytetty?

Johdannon lisäksi tutkielma sisältää luvut 2-5. Luvussa 2 on esitetty jatkuvan integraation ja siihen liityvän testauksen teoriaa. Tutkimusmenetelmään, tutkimuskysymyksiin, sekä aineistonhakuprosessiin keskitytään luvussa 3. Tulokset esitetään luvussa 4 ja pohdinta ja yhteenveto luvussa 5.

2 Jatkuva integraatio

2.1 Jatkuvan integraation määritelmä

Jatkuvalla integraatiolla tarkoitetaan koodimuutosten yhdistämistä säännöllisesti yhteiseen versionhallinnan kehityshaaraan. Sovellukselle muutoksineen ajetaan automaattisesti koontiversio sekä testit, jotta voidaan varmistua siitä, että muutokset eivät riko sovellusta (RedHat 2022).

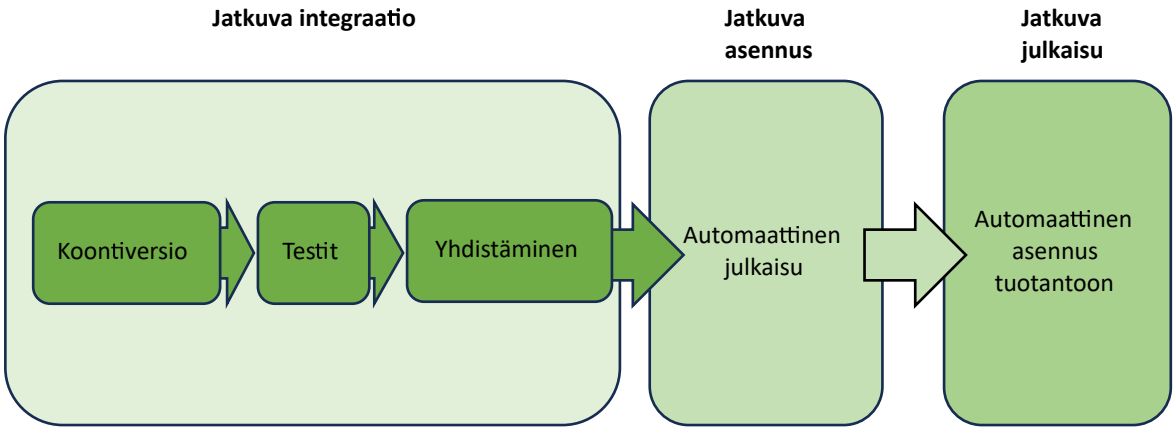
Säännöllisellä koodimuutosten yhdistämisellä pyritään välttämään yhdistämisen monimutkaisuutta myöhemmässä vaiheessa. Kehittäjä joka koittaa yhdistää tekemänsä muutokset yhteiseen versionhallintaan saa ilmoituksen onnistuiko yhdistäminen vai ei. Jos yhdistäminen ei onnistunut voi kehittäjä tehdä tarvittavat korjaukset rikkoviin muutoksiin, jotka estivät muutosten yhdistämisen. Rikkovaa koodia ei tule yhdistää yhteiseen toimivaan kehityshaaraan (Hamdan ja Alramouni 2015).

Jatkuva integraatio mahdollistaa useammin tehtävät julkaisut, parantaa sovelluksen laatua ja lisää kehitystiimin luovuutta (Shahin, Ali Babar ja Zhu 2017, 3910). Nopeaa ja tehokasta sovelluskehitystä haittaavat kuitenkin useasti pitkät koontiajat, joiden yhtenä aiheuttajana ovat automaatiotestit. Automaatiotestien hitauteen vaikuttavat sovelluksen monimutkaisuus, sovellukseen tehdyt muutokset, sekä epätehokas testaaminen. Pitkien koontiaikojen takia kehittäjät joutuvat odottamaan kauemmin palautetta muutoksistaan ja näin ollen sovelluksen tuotantoonvienti hidastuu (Marijan, Liaaen ja Sen 2018, 22–23).

Jatkuvasta integraatiosta puhuttaessa käytetään lähteestä riippuen usein samassa yhteydessä termiä *CI/CD*. Tässä *CI* viittaa jatkuvaan integraatioon (engl. continuous integration) ja *CD*, joko jatkuvaan julkaisuun (engl. continuous delivery) tai jatkuvaan asennukseen (engl. continuous deployment). Jatkuvalla julkaisulla tarkoitetaan automaattista sovelluksen julkaisua muutosten jälkeen ja jatkuvalla asennuksella sitä, että julkaistu sovellus asennetaan automaattisesti tuotantoon (RedHat 2022).

Joissain lähteissä, kuten Hilton ym. (2016) CI:n määritelmään luetaan mukaan myös sovelluksen julkaisu. Tässä tutkielmassa jatkuva integraatio määritellään, kuten Lima ja Vergilio

(2020) ja Shahin, Ali Babar ja Zhu (2017, 3910) sen määrittelevät. Näiden lähteiden mukaan jatkuva integraatio sisältää automaattisen sovelluksen koonnin ja testauksen. Kuviossa 1 on kuvattu tässä tutkimuksessa käytetty määritelmä jatkuvasta integraatiosta. On kuitenkin hyvä huomioida kirjallisuutta lukiessa, että jatkuvan integraation määritelmässä on jonkin verran vaihtelua, kuten myös (Laukkanen, Itkonen ja Lassenius 2017, 56) toteavat kirjallisuuskatsauksessaan.



Kuvio 1. CI/CD vaiheiden kuvaus (RedHat 2022).

Jatkuvan integraation käytölle on esitetty useita perusteita. Hilton ym. (2016, 435) esittävät tekemässään tutkimuksessa että jatkuvaa integraatiota käyttävät projektit julkaisivat kaksi kertaa niin paljon kuin projektit joissa ei käytetty jatkuvaa integraatiota ja koodimuutokset yhdistettiin nopeammin. Kehittäjät olivat myös vähemmän stressaantuneita koontiversion rikkomisesta. GitHubin suosituimmista projekteista 70% käyttää jatkuvaa integraatiota (Hilton ym. 2016, 435).

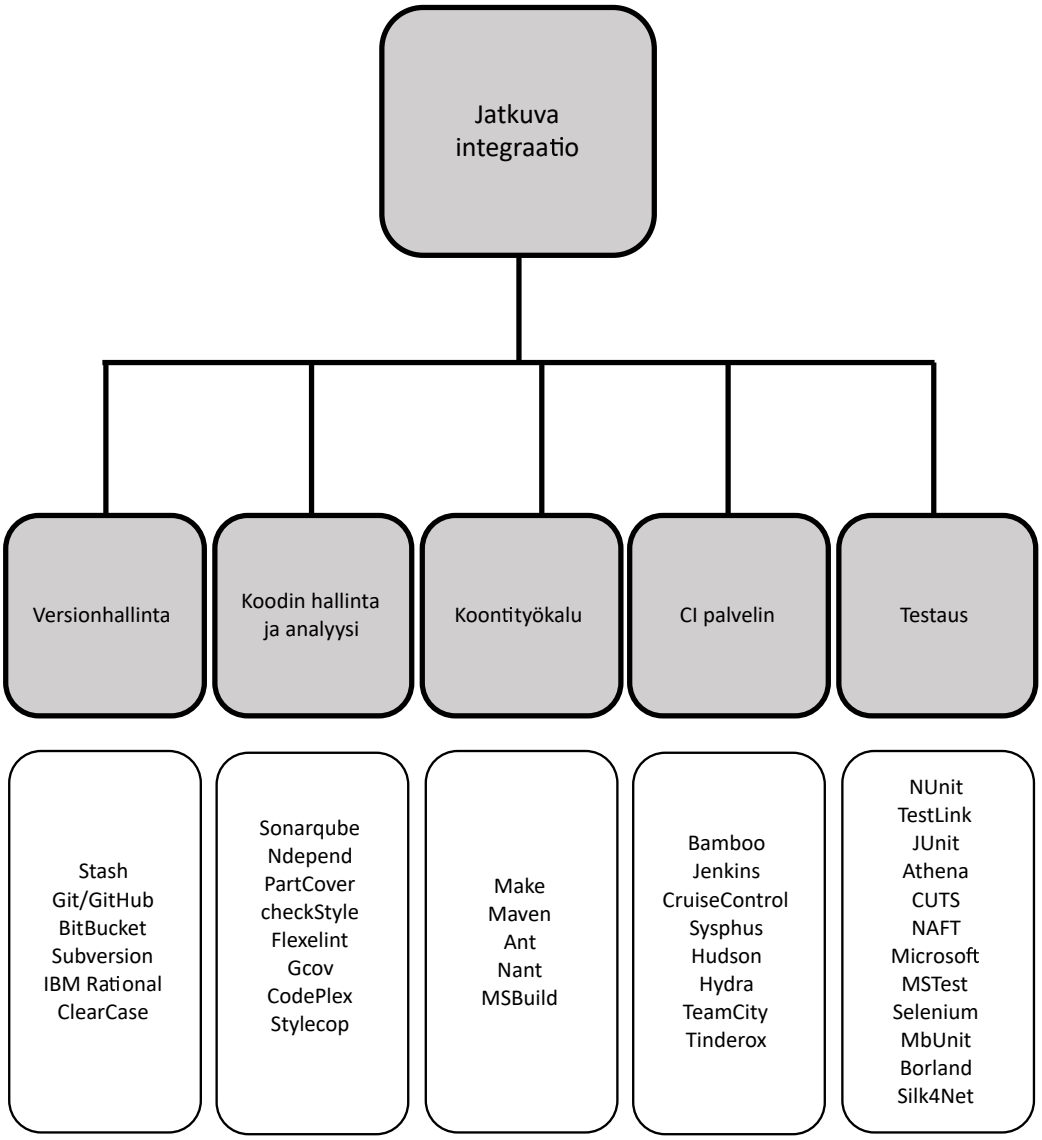
Vasilescu ym. (2015) puolestaan tutkivat 246 GitHub projektia joissa oli toteutettu jatkuva integraatio Travis CI:n avulla. Tutkimuksessa todettiin, että jatkuvan integraation toteuttaminen lisäsi koodimuutosten yhdistämispyyntöjä (engl. pull request) pääkehittäjiltä. Nämä pyynnöt myös hyväksyttiin aiempaa useammin. Tämä nähtiin merkkinä siitä, että jatkuvan integraation käyttöönotto paransi koodin yhdistämisten hallintaa. Pyyntöjen lisääntymisen ei nähty lisäävän käyttäjäkokemukseen liittyvien virheiden määrää. Kehittäjien raportoimien virheiden määrässä sen sijaan nähtiin kasvua, jonka nähtiin johtuvan siitä että jatkuvan integraation käytön jälkeen kehittäjien oli helpompi havaita virheet.

Alla on kuvattu esimerkki nykyaikaisesta jatkuvan integraation prosessista, jossa kehittäjä tekee muutoksen sovelluksen koodiin ja yhdistää sen versionhallintaan (Penson ym. 2017):

- Versionhallinnasta otetaan sen hetken tilanne omalle koneelle
- Luodaan oma haara muutoksille
- Suoritetaan muutokset
- Lähetetään muutokset versionhallintaan
- CI-palvelimella ajetaan testit ja analyysit ja infotaan virheistä ja testien epäonnistumisista sekä varoituksista
- Jos testit epäonnistuvat kehittäjän täytyy muokata koodia. Jos testit menevät läpi muutokset katsoo toinen kehittäjä ja mikäli hän hyväksyy muutokset ne voidaan yhdistää kehityshaaraan

2.2 Jatkuvan integraation työkalut

Jatkuvan integraation eri vaiheissa voidaan käyttää erilaisia työkaluja automaation helpottamiseksi. Työkalujen valinnalla on merkitystä siihen kuinka helppoa jatkuvan integraation käyttöönotto on yrityksissä (Shahin, Ali Babar ja Zhu 2017). Shahin, Ali Babar ja Zhu (2017) jaottelevat työkalut eri jatkuvan integraation vaiheiden mukaan, kuten kuviossa 2 on esitetty. Kuviossa 2 on eri vaiheiden kohdalla työkaluja joita käytettiin heidän kirjallisuuskartoituksessaan tutkimissa artikkeleissa.



Kuvio 2. Jatkuvan integraation työkalut vaiheittain (Shahin, Ali Babar ja Zhu 2017).

Kuviossa 2 esitetään jatkuvan integraation vaiheet suoritusjärjestyksessä. (Shahin, Ali Babar ja Zhu 2017) huomauttavat kuitenkin että ei ole olemassa standardia jatkuvan integraation toteutuksesta, eikä kaikkien vaiheiden toteuttaminen ole tarpeellista. Heidän esimerkissään kehittäjät yhdistävät ensin tekemänsä muutokset versionhallintaan, jonka jälkeen koodinhallinta ja analyysi -työkalulla voidaan analysoida esimerkiksi koodikattavuutta ja koodauskäytänteiden noudattamista. Koodinhallinta ja analyysi -työkalu voidaan integroida käytettyyn CI-palvelimeen, kuten myös esimerkiksi testityökalut. CI-palvelimen tehtävänä on toimia

jatkuvan integraation keskiössä ja suorittaa versionhallinnan muutosten yhteydessä esimerkiksi koontiprosessi, koodianalyysit sekä yksikkötestit.

2.3 Jatkuva integraatio ja testaaminen

Automaattiset testit nopeuttavat sekä tehostavat virheiden löytymistä sovelluksen koontiversiota tehtäessä (Fowler 2006). Jatkuvaa integraatiota pidetään usein yhtenä tärkeimmistä ketterää sovelluskehitystä ja testausta tukevista käytännöistä (Stolberg 2009, 369).

Tyypillisesti testaus aloitetaan jo kehittäjien paikallisesti ajamilla yksikkötesteillä, joilla vahvistetaan muutosten toimivuus paikallisessa kehitysympäristössä. Testien mennessä läpi kehittäjät yhdistävät muutokset yhteiseen versionhallintaan jatkuvan integraation prosessin mukaisesti. Tässä vaiheessa CI-palvelimella ajetaan sovelluksen koontiversio ja testit, sekä koodianalyysit käyttäen koodianalyysityökaluja (Kato, Shimizu ja Ishikawa 2022)

Automaattisilla koodianalyysityökaluilla voidaan analysoida esimerkiksi, metodien nimeämiskäytäntöjä, koodin kompleksisuutta, koodin haavoittuvuuksia tai sovittujen koodikäytänteiden noudattamista (Vassallo ym. 2019).

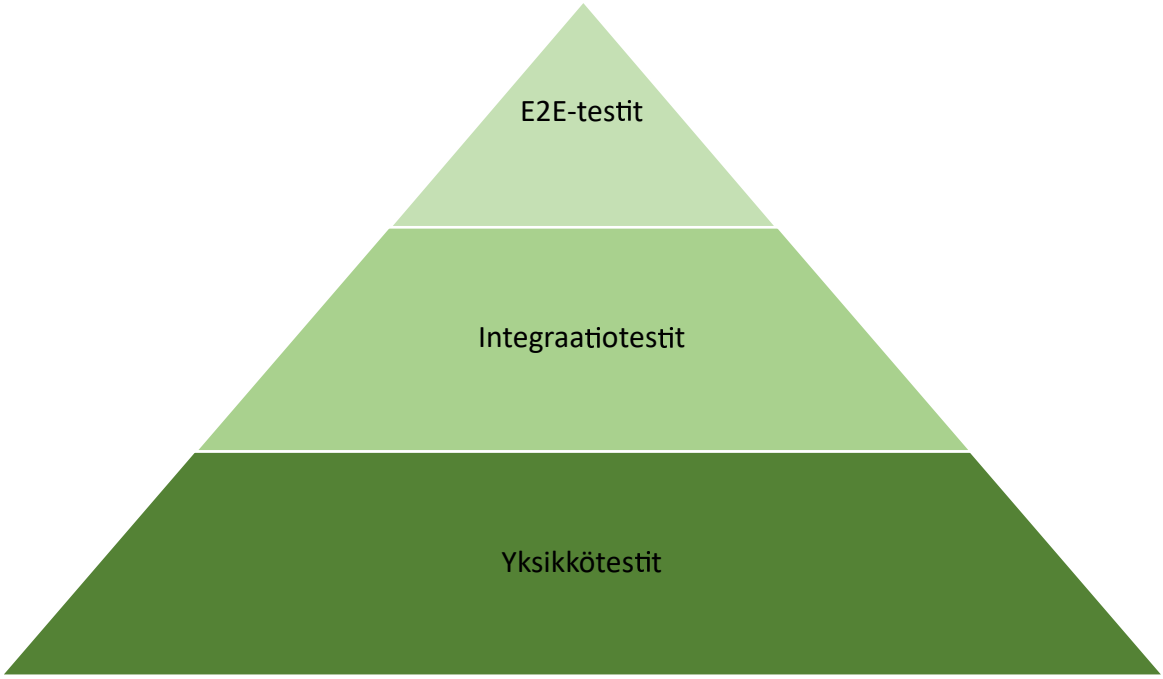
Eräitä jatkuvan integraation toteutuksessa mahdollisesti käytettäviä testaustasoja kuvataan taulukossa 1.

Testaustaso	Testauksen kohde
Staattinen analyysi	Sovittujen koodikäytänteiden noudattaminen, koodin kompleksisuus, koodin haavoittuvuus (Vassallo ym. 2019)
Yksikkötesti	Yksittäiset yksiköt
Integraatiotesti	Yksittäisten moduulien yhteensopivuus
Regressiotesti	Toiminnallisuus muutosten jälkeen
E2E-testi	Järjestelmän kokonaisuus

Taulukko 1. Testaustasot ja niiden kohteet (Kato, Shimizu ja Ishikawa 2022, 157)

Tyypillisesti sovellukselle ajetaan yksikkö -ja integraatio testejä (RedHat 2022). Testaustasojen priorisointiin voidaan käyttää esimerkkinä kuviota 3, jossa nopeasti ja helposti suo-

ritettavia yksikkötestejä on määrällisesti eniten. Ne luovat vahvan pohjan testaukselle jonka jälkeen voidaan siirtyä monimutkaisempiin testeihin (Kato, Shimizu ja Ishikawa 2022, 161). Yksikkötesteillä testataan ohjelman yksittäisiä yksikköjä erillään muusta järjestelmästä (Runeson 2006). Ne ovat myös halvempia kuin kuviossa 3 esitetyt muut testaustasot (Wongkampoo ja Kiattisin 2017).



Kuvio 3. Testipyramidi (Kato, Shimizu ja Ishikawa 2022; Wongkampoo ja Kiattisin 2017).

Integraatiotesteillä testataan järjestelmän yksittäisten moduulien yhteensopivuutta (Leung ja White 1990, 290) ja E2E-testeillä puolestaan keskitytään testaamaan järjestelmän kokonaisuutta keskittyen käyttäjän näkökulmaan (Bai ym. 2001, 141).

Guşeilă, Bratu ja Moraru (2019) esittävät jatkuvan integraation tutkimuksessaan, että koonnin ja yksikkötestien jälkeen ajetaan staattinen koodianalyysi, jonka avulla voidaan tunnistaa koodivirheitä ja turvallisuuspuutteita, sekä vahvistua siitä että koodauskäytänteitä noudatetaan. Samassa tutkimuksessa todetaan regressiotestien olevan testejä joilla vahvistetaan, että olemassa olevat toiminnallisuudet toimivat myös muutosten jälkeen.

Testaaminen on kuitenkin kallista ja aikaa vievää ja jokaisen muutoksen testaaminen ei ole välttämättä mahdollista. Tämän takia onkin kehitetty erilaisia testien valintaan liittyviä tek-

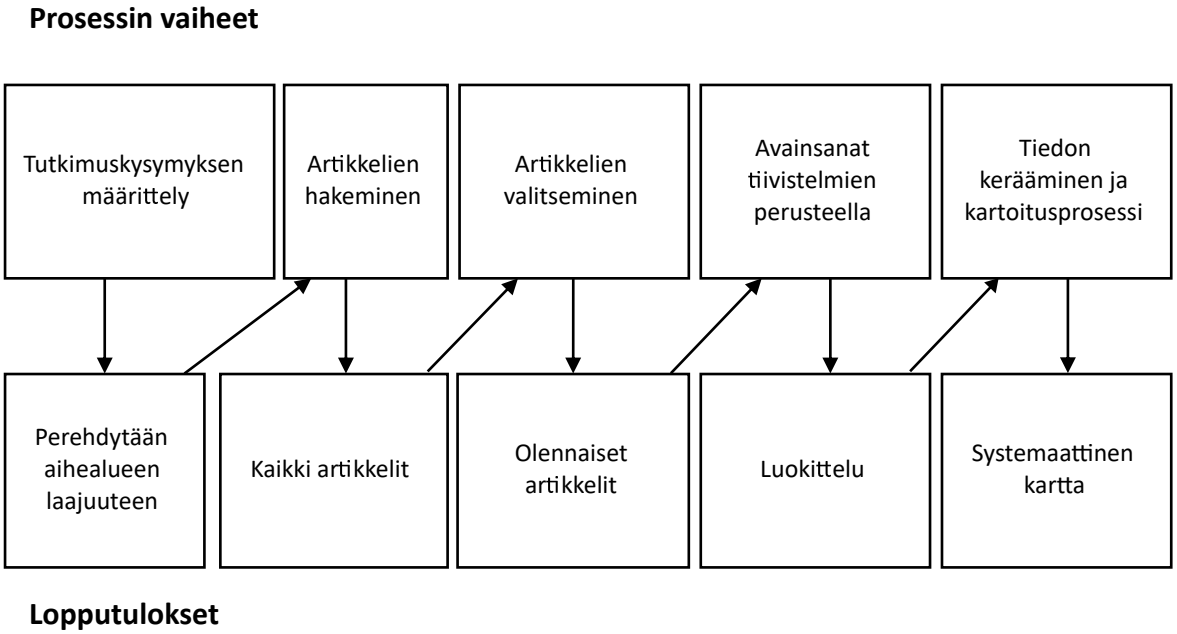
niikoita, joilla pyritään vähentämään kustannuksia (Bavand ja Rigby 2021, 217).

Esimerkiksi Ali ym. (2019) esittävät mallin, jossa testitapausten priorisointi ja valinta toteutetaan perustuen usein vaihtuviin, sekä usein epäonnistuviin testeihin. Mallin perusteella valitaan ajettavat testit sekä niiden ajojärjestys, jotta virheitä pystyttäisiin havaitsemaan aiemmin, sekä parantamaan niiden havaitsemisastetta. Samaan kustannusten karsimiseen pyrkivät Bavand ja Rigby (2021) tutkimuksessa, jossa pyrittiin vähentämään testiajojen määrää yhdistämällä koodimuutoksia yhteen ennen testien ajoa.

3 Tutkimusmenetelmä ja tutkimuskysymys

3.1 Tutkimusmenetelmä

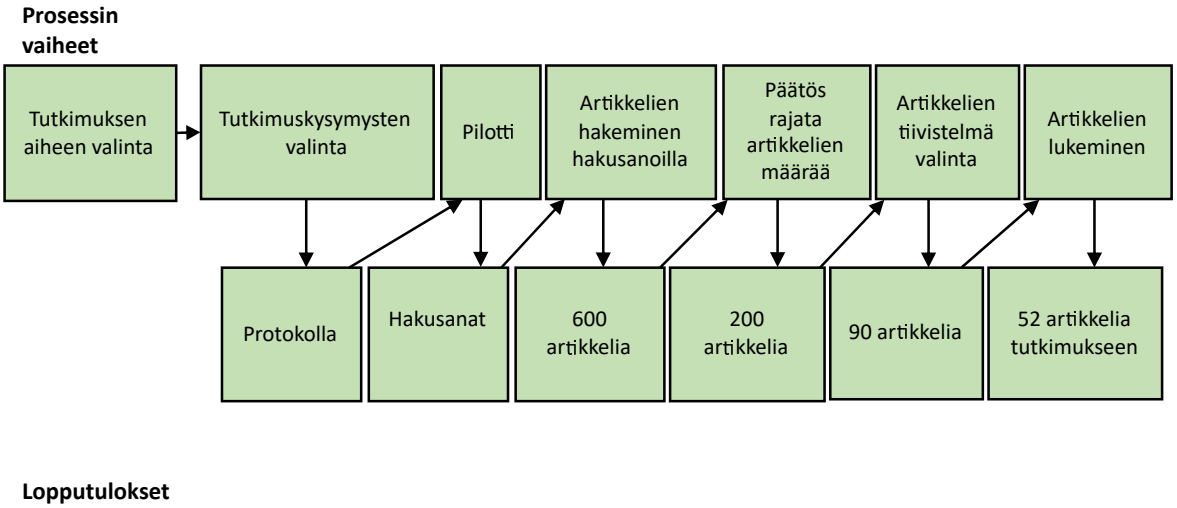
Tämän tutkielman tutkimusmenetelmänä käytetään systemaattista kirjallisuuskartoitusta. Kirjallisuuskartoituksessa käydään läpi olemassa olevia aihealueen primääritutkimuksia sekä vastataan asetettuihin kartoituskysymyksiin, joita tyypillisesti ovat käytettyjen menetelmien ja saavutettujen tulosten kuvaaminen. Tavoitteena on luoda yleiskatsaus aihealueella tehtyyn tutkimukseen. Usein halutaan esimerkiksi kartoittaa milloin ja minkätyyppisiä artikkeleita on julkaistu (Petersen ym. 2008). Petersen ym. (2008) ovat kuvanneet kirjallisuuskartoituksen prosessia kuvion 4 mukaisesti:



Kuvio 4. Kirjallisuuskartoituksen prosessi (Petersen ym. 2008).

Näiden vaiheiden avulla pyritään löytämään tutkimuksen kannalta relevantit artikkelit, jotka vastaavat asetettuihin tutkimuskysymyksiin (Petersen ym. 2008).

Tämän tutkimuksen prosessi on kuvattu kuviossa 5:



Kuvio 5. Tutkimuksen prosessi.

Kuviossa 5 kuvattua tutkimuksen prosessia esitellään seuraavissa luvuissa.

3.2 Tutkimuskysymykset

Tutkimuksessa vastataan kysymykseen:

- Mikä on jatkuvaan integraatioon liittyvän testaamisen tutkimuksen nykytila?

Tutkimuskysymykseen pyritään vastaamaan seuraavien alakysymysten avulla:

- Minkälaisia työkaluja jatkuvan integraation toteutuksessa on käytetty?
- Kuinka paljon tutkimuksia on julkaistu?
- Minkälaisia testaustasoja jatkuvan integraation yhteydessä on käytetty?

3.3 Protokolla

Tutkimusta varten tehtiin protokolla, jossa määriteltiin tutkimuksen lähtöasetelma sekä suunniteltu toteutus.

Protokollassa määritettiin tutkimuksen aihe, tutkimuskysymykset, kartoituskysymykset, hyväksyntä -ja hylkäyskriteerit artikkeleille, tutkimusmenetelmä, sekä kuvattiin artikkeleiden

hakua varten tehdyn pilotin suunnittelu. Protokollaa päivitettiin pilotin jälkeen valittujen käytettävien hakusanojen osalta.

3.4 Pilotti

Tämän tutkimuksen aineistonhakua varten tehtiin pilotti, jonka avulla valittiin mistä tietokannoista artikkeleita haettiin, sekä mitä hakusanoja käytettiin. Pilottia lähdettiin tekemään sen jälkeen kun tutkimuksen aihe, sekä tutkimuskysymys ja tutkimuskysymyksen alakysymykset oli valittu. Tässä vaiheessa myös alustavat hyväksymis- ja hylkäyskriteerit, tietokannat joista artikkeleita haetaan, sekä hakusanat oli määritelty.

Pilotti päätettiin tehdä niin, että etsittiin alustavilla hakusanoilla kahdesta hakukoneesta, Scopuksesta ja IEEE:stä, artikkeleita ja katsottiin montako artikkelia hakusanoilla löytyy. Tarkoituksena oli löytää tällä tavalla hakusanat, joilla löytyy tutkimuksen kannalta oleellisia artikkeleita, sekä rajoittaa artikkelien määrää tutkimuksen laajuudelle sopivaksi.

Eri hakuja pilotissa oli seitsemän kappaletta, joista kuusi ensimmäistä hakua suoritettiin 03.03.2023 ja seitsemäs 10.03.2023. Seitsemännessä haussa päädyttiin 600 artikkeliin jolloin päätettiin, että suoritetaan pilotti kyseisillä hakusanoilla.

Pilotti hakusanoilla tehtiin Scopukseen siten, että valituilla hakusanoilla haettiin Scopuksesta artikkeleita Relevance järjestyksessä ja katsottiin kuinka monta artikkelia 20 ensimmäisestä täyttää tiivistelmän perusteella tutkimuksen hyväksymiskriteerit, mutta ei hylkäyskriteeriä. Tiivistelmien lukemisen perusteella päädyttiin siihen että 14 artikkelia täytti asetetut kriteerit. Tässä vaiheessa todettiin, että hakusanoilla löytyy relevantteja artikkeleita ja voidaan aloittaa aineistonhaku, mutta että kriteerit täyttävien artikkelien määrä on kuitenkin todennäköisesti suuri ottaen huomioon tutkielman laajuuden. Päätettiin että luetaan tiivistelmät 100:sta ensimmäisestä artikkelista molemmista tietokannoista Relevance järjestyksessä.

Pilotin 6 ensimmäistä hakua on koottu liitteiksi ja 7. haku jonka hakusanoja käytettiin tutkimuksen artikkelien hakuun on kuvattu kappaleessa Hakusanat.

3.5 Hakusanat

Hakusanojen avulla pyrittiin löytämään tutkimuksen kannalta oleellisia artikkeleita. Tutkimuksen hakusanoiksi valikoituivat hakusanat, jotka vaikuttivat pilotin perusteella parhailta. Näitä hakusanoja käytettiin pilotin haussa 7. Tähän hakuun otettiin hakusanoiksi aiemmin käytettyjä hakusanoja, mutta poistettiin *methods* sana, jonka koettiin rajoittavan hakutuloksia liikaa. Hakutulosten määrä oli näilläkin hakusanoilla suuri, mutta pilotti päädyttiin tekemään näillä hakusanoilla sen takia että kokonaisuutena hakusanat vaikuttivat parhailta. Tässä vaiheessa todettiin kuitenkin, että mikäli pilotissa löytyy relevantteja artikkeleita, niin haetaan näillä hakusanoilla molemmista tietokannoista 100 ensimmäistä artikkelia Relevance järjestyksessä, jotta artikkelien määrä pysyy sopivana tutkimuksen laajuuteen nähden.

Tutkimuksessa käytetyt hakusanat:

Haettu 10.03.2023

Tietokanta	Hakusanat	Artikkelien hakujärjestys	Tuloksia
Scopus	TITLE-ABS-KEY ("continuous integration"AND testing AND "software development")	Relevance	236
IEEE	("All Metadata":continuous integration) AND ("All Metadata":testing) AND ("All Metadata":software development)	Relevance	364

Taulukko 2. Tutkimuksessa käytetyt hakusanat

3.6 Hyväksyntä- ja hylkäyskriteerit

Artikkeleiden valinnassa käytettiin hyväksyntä- ja hylkäyskriteereitä, joiden avulla artikkeleita karsittiin.

Hyväksyntäkriteerit:

- Tieteellinen alkuperäisartikkeli
- Englanninkielinen
- Internetistä luettavissa
- Empiirinen tutkimus
- Vähintään julkaisufoorumin luokka 1 artikkeli
- Artikkeli käsittelee jatkuvaa integraatiota ja siihen liittyvää testausta

Hylkäyskriteerit:

- Artikkeli on kirjallisuuskartoitus tai kirjallisuuskatsaus

3.7 Kartoitusprosessi ja kartoituskysymykset

Aineisto tutkimukseen haettiin kahdesta eri tietokannasta, jotka olivat Scopus ja IEEE. Molemmista tietokannoista haettiin samoilla hakusanoilla 100 artikkelia Relevance järjestyksessä. Artikkeleista luettiin tiivistelmä, jonka perusteella katsottiin täyttyvätkö tutkimuksen hyväksymiskriteerit. Mikäli ne täyttyivät, eikä tutkimuksen hylkäyskriteeri täyttynyt, artikkeli otettiin mukaan seuraavaan vaiheeseen, jossa se luettiin kokonaan. Joidenkin artikkelien kohdalla pelkän tiivistelmän lukeminen ei antanut riittävää kuvaa siitä täyttyvätkö hyväksymiskriteerit. Tällaisten artikkelien tekstiä luettiin sen verran että saatiin kuva täyttyvätkö ne.

Näin löydettiin Scopuksesta 53 artikkelia ja IEE:stä 37 artikkelia. IEEE:stä löytyi samoja artikkeleita kuin Scopuksesta 26 kappaletta, mutta niitä ei otettu tukimuksessa huomioon. Yhteensä artikkeleita löydettiin 90 kappaletta.

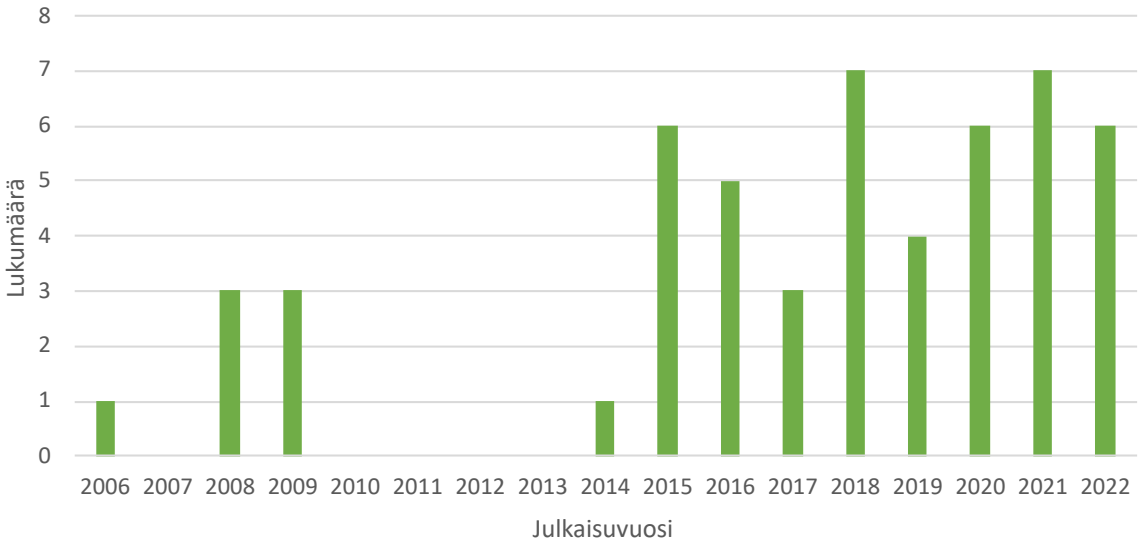
Nämä artikkelit luettiin kokonaisuudessaan keräten kartoituskysymyksiin vastauksia. Kartoituskysymyksiä olivat:

- Minä vuonna artikkeli on julkaistu?
- Minkälaisia työkaluja jatkuvan integraation toteutuksessa on käytetty?
- Millaisia testaustasoja on käytetty?

Artikkelien lukuvaiheessa karsittiin 38 artikkelia, joita ei otettu mukaan tutkimukseen ja jotka täyttivät alkuperäiset hyväksyntäkriteerit. Artikkelit karsittiin pääasiallisesti siitä syystä, että niissä ei oltu tehty omaa empiiristä tutkimusta jatkuvasta integraatiosta ja siihen liittyvästä testauksesta tai se vaikutti vähäiseltä. Artikkeleissa oli tutkittu esimerkiksi useita muiden projekteja. Myös sellaiset artikkelit joissa tutkimus näytti jäävän vain teorian tasolle karsittiin. Näissä artikkeleissa esitettiin esimerkiksi teoria jatkuvaan integraatioon liittyvän testauksen toteutuksesta, mutta artikkelista ei ilmennyt oliko teoriaa testattu. Lukemisen jälkeen tutkimukseen otettiin mukaan 52 artikkelia.

4 Tulokset

4.1 Artikkelien julkaisuvuodet



Kuvio 6. Artikkelien lukumäärä vuosittain.

Artikkelit painottuvat 2010-luvulle ja yhtään artikkelia ei ole julkaistu ennen 2000-lukua. Jatkuvan integraation käytännön lähteestä on kirjallisuudessa jonkin verran eriäviä näkökulmia, mutta vaikuttaa että se on kehitetty 1990-luvulla, joten on loogista että tutkimuksiakin aiheesta näyttää alkaneen tulla vasta 2000-luvulla. Myös painotus ajalle 2015-2022 vaikuttaa loogiselta, sillä teknologian kehittyessä, myös jatkuvan integraation ja sen tutkimuksen tarve on kasvanut.

On hyvä huomioida, että ensimmäisen julkaistun artikkelin ja uusimman välissä teknologia on kehittynyt huomattavasti ja jatkuvan integraation käyttö on laajempaa ja tutkimus kattavampaa nykyään kuin ensimmäisen artikkelin julkaisuvuonna 2006. Esimerkiksi työkalut ovat uudistuneet ja monipuolistuneet ja testaaminen kehittynyt. Näin ollen tässä tutkimuksessa esitetyissä tuloksissa esimerkiksi jotkin työkalut voivat olla nykyajan jatkuvan integraation työkaluihin tottuneille vieraita.

Artikkelit on pääasiassa julkaissut IEEE, sillä tutkimuksien valintavaiheessa puolet tarkastelluista artikkeleista haettiin IEEE:stä ja myös Scopuksesta haetuista artikkeleista moni oli

IEEE:n julkaisemia. Yhteensä IEEE:n julkaisemia artikkeleita oli 45. Muissa lähteissä julkaistuja artikkeleita olivat ACM:n 4 julkaisua, Springerin 2 julkaisua, ja Elsevierin 1 julkaisu. Näistä ACM ja IEEE ovat julkaisufoorumin luokitukseltaan 1 ja Springer ja Elsevier ovat luokitukseltaan 2. Luokitus 1 kuvaa julkaisufoorumin mukaan perustasoa ja 2 johtavaa tasoa ("Julkaisufoorumi" 2023). Alle 1 luokituksen saaneet artikkelit hylättiin hylkäyskriteerin mukaisesti.

4.2 Artikkelien jaottelu

Tutkimuksien aihepiiri oli rajattu hakusanoilla ja kriteereillä sellaiseksi, että kaikki artikkelit käsittelivät jatkuvaa integraatiota ja siihen liittyvää testaamista jollain tavalla. Tutkimuksien näkökulma vaihteli kuitenkin sen verran, että näissä tuloksissa päädyttiin tekemään tarkempi jaottelu havainnollistamaan sitä mikä on ollut tutkimusten pääasiallinen tutkimusaihe. Jaottelussa on kaksi kategoriaa, joista toisessa kategoriassa ovat ne tutkimukset, joissa jatkuvan integraation toteutuksen katsottiin olevan pääasiallinen tutkimuksen aihe ja toisessa ne joissa testaamisen katsottiin olevan pääasiallinen tutkimusaihe. Molempien kategorioiden tutkimuksissa on kuitenkin käsitelty kumpaakin aihetta. Kategoriat on kuvattu seuraavaksi:

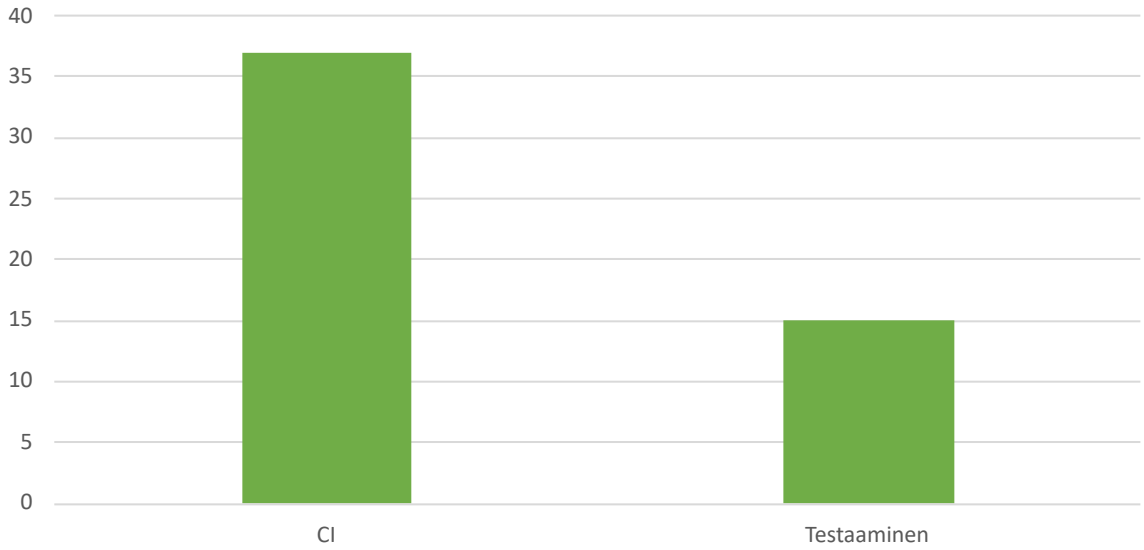
CI

CI kategoriaan valikoituivat ne artikkelit, joissa esimerkiksi toteutettiin jatkuva integraatio itse tai integroitiin jokin sovellus osaksi jatkuvaa integraatiota. Näissä artikkeleissa oli myös yleensä jokin sovellus tai järjestelmä jota varten tutkimusta jatkuvan integraation käytöstä tehtiin. Testien osuus näissä tutkimuksissa oli enemmän pintapuolista, mutta esimerkiksi testaustasoja joita käytettiin nimettiin useammin. Myöskin käytetyt työkalut mainittiin näissä tutkimuksissa useammin.

Testaaminen

Testaaminen kategoriassa tutkittiin pääasiassa testaamista ja aihealueina olivat muun muassa testien priorisointi, testien valinta ja koodimuutosten yhdistäminen, jotta testien ajokertoja saatiin vähennettyä. Näissä artikkeleissa jatkuvaa integraatiota ei oltu esimerkiksi itse toteutettu ja työkalujen maininta oli harvinaisempaa. Tutkimukset painottuivat tutkimaan miten

testaamisesta saa tehokkaampaa ja testaustasoja ei välttämättä nimetty vaan niitä käsiteltiin yleisellä tasolla esimerkiksi testijoukkoina. Kuviossa on 7 on esitetty tutkimukset kategorioittain.



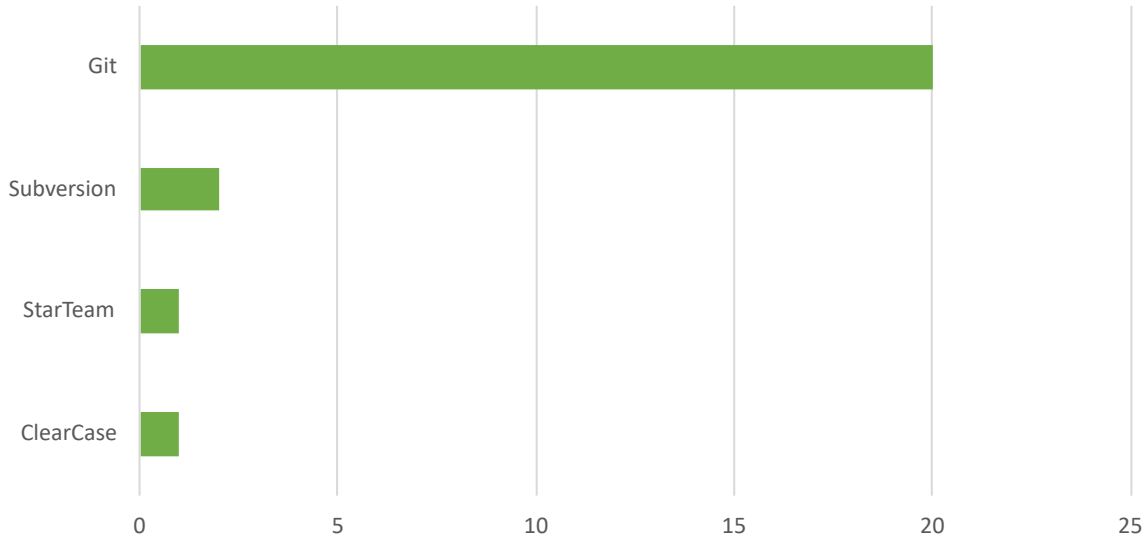
Kuvio 7. Tutkimukset kategorioittain.

37 tutkimuksessa käsiteltiin pääasiallisesti jatkuvaa integraatiota ja 15 testaamista.

4.3 Versionhallinta

Versionhallinta on ohjelmistokehitysprojekteissa käytetty järjestelmä lähdekoodin hallinnointiin. Käytännössä nykyaikaisissa ohjelmistokehitysprojekteissa versionhallinta on jollain tavalla käytössä. Usein versionhallinta yhdistetään CI-palvelimeen, jolloin koodimuutoksista lähetetään automaattisesti tieto CI-palvelimelle. Teixeira, Arrais ja Veiga (2021) mukaan versionhallinta toimii jatkuvaa integraatiota käyttävien projektien perustana.

Kuviossa 8 kuvataan tutkimuksissa käytettyjä versionhallintajärjestelmiä.



Kuvio 8. Käytetyt versionhallintajärjestelmät.

Tutkimuksen tuloksissa näkyy, että käytetty versionhallintajärjestelmä mainitaan vain noin 42 % tutkimuksista, vaikka sen voidaankin laskea kuuluvan jatkuvan integraation toteutukseen. Vaikka versionhallintajärjestelmä onkin tärkeä lähdekoodin hallintaan, käytetyn versionhallintajärjestelmän mainintaa ei kuitenkaan välttämättä pidetä oleellisena jatkuvan integraation toteutuksen kannalta.

Git oli käytetyin versionhallintajärjestelmä ja eniten käytetyt Git-repositorioiden hallintapalvelut olivat GitHub ja GitLab. Hallintapalvelu mainittiin kuitenkin harvoin ja monessa tutkimuksessa sitä ei mainittu vaikka versionhallintajärjestelmä mainittiin. Luvuissa on mukana vain tutkimusten teksteissä selkeästi mainitut käytetyt versionhallintajärjestelmät. Esimerkiksi vain kuvassa olevaa hallintapalvelun logoa ei laskettu, eikä käytetyn CI-palvelimen perusteella tulkittu versionhallintajärjestelmää. Jos nämäkin olisi laskettu niin Git olisi mainittu muutaman kerran useammin.

Integraatio versionhallintajärjestelmän ja CI-palvelimen välillä on jatkuvassa integraatiossa oleellinen, joten varsinkin hallintapalveluita, joissa on usein toteutettu valmiiksi integraatiomahdollisuus yleisiin CI-palvelimiin, olisi voinut olettaa mainittavan useammin kuin mitä tutkimuksissa mainittiin.

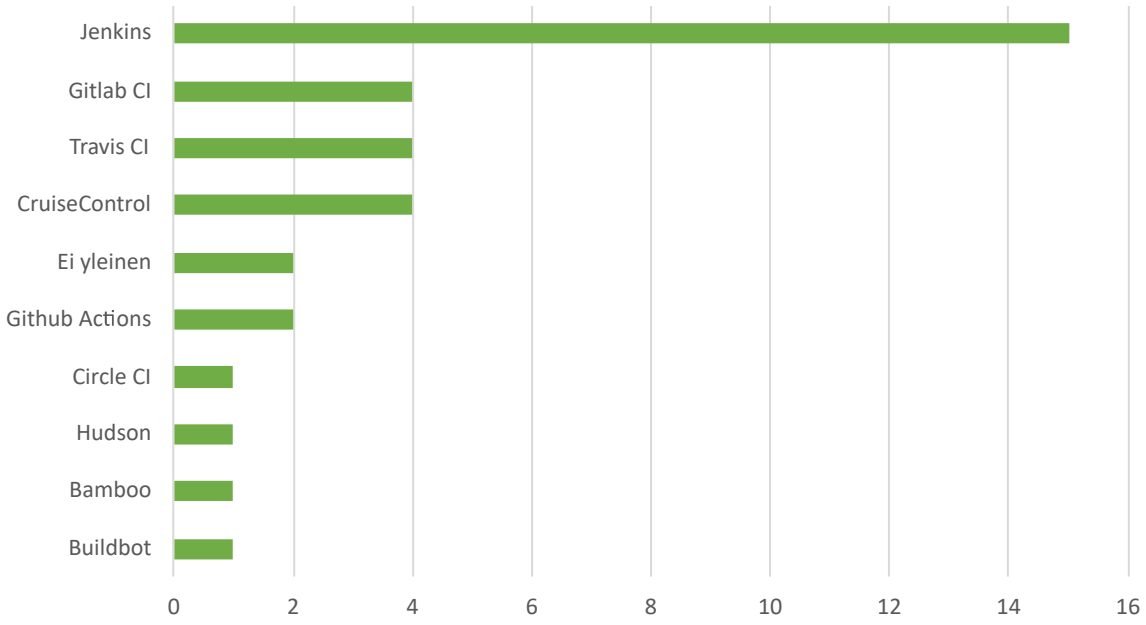
Testaamiseen keskittyvissä tutkimuksissa vain kolmessa mainittiin käytetty versionhallinta-

järjestelmä.

4.4 Jatkuvan integraation palvelin

CI-palvelimen nimi mainittiin 32 tutkimuksessa. Useiten käytetty CI-palvelin oli Jenkins, joka mainittiin 15 tutkimuksessa. Esimerkiksi Niranjan ja Mohana (2022) mainitsevat Jenkinsin olevan yhden eniten mainituista CI-palvelimista kirjallisuudessa. Sen suosioon vaikuttavat avoin lähdekoodi, helppo asennettavuus, soveltuvuus usealla käyttöjärjestelmälle, useat eri liitännäiset ja vahva yhteisön tuki.

Suurimmassa osassa tutkimuksen artikkeleista, joissa CI-palvelin oli mainittu käytettiin yleisesti saatavilla olevaa CI-palvelinta, joista monet ovat myös ilmaisia ainakin kuluttajakäytössä. Vaikka CI-palvelin onkin yleisesti käytetty niin esimerkiksi Alégroth, Karlsson ja Radway (2018) käyttivät bash skriptiä, joka toimi yleisesti käytössä olevien CI-palvelinten tilalta. Koonti tehtiin cron ajastuksella niin, että bash skripti haki muutokset versionhallintajärjestelmästä, ajoi koonnin ja testit. Tämä valinta oli tehty siksi että testaajat tunsivat bash skriptaamisen eivätkä esimerkiksi Jenkinsiä tai TFS:ää, joiden käyttö olisi tuonut myös lisäkustannuksia. CI-palvelimen tilalta voidaan siis käyttää myös muunlaisia toteutuksia jotka toteuttavat jatkuvan integraation tarkoituksen. Kuviossa 9 on esitetty tutkimuksissa käytetyt CI-palvelimet. Joissain artikkeleissa mainittiin useampi CI-palvelin.



Kuvio 9. Käytetyt CI-palvelimet.

Tutkimuksessa CruiseControl, Hudson ja Buildbot ovat olleet käytössä tutkimuksissa jotka on julkaistu vuosina 2006-2009. Jenkins puolestaan on ollut käytössä tutkimuksissa jotka on julkaistu välillä 2014-2022. Esimerkiksi Fowler (2006) mainitsee vuoden 2006 julkaisussaan parhaaksi CI-palvelimeksi CruiseControlin. Tästä huomaa, että tutkimuksissa käytetyt työkalut riippuvat jonkin verran siitä milloin tutkimus on julkaistu.

4.5 Staattinen koodianalyysi

Staattista koodianalyysiä käytettiin 12 tutkimuksessa. Staattisella koodinalyysillä, pyrittiin parantamaan mm. koodikattavuutta, koodauskäytänteiden noudattamista ja koodinlaatua. Esimerkiksi Lu, Yang ja Qian (2014) totesivat että staattisella koodianalyysityökalulla voidaan löytää monia ongelmia ja vähentää niistä aiheutuvia kustannuksia. Jonkinlainen staattinen koodianalyysityökalu on ollut käytössä myös vanhemmissa artikkeleissa, joten tutkimuksen perusteella ei voida todeta, että staattinen koodianalyysi olisi vain uudemmissa tutkimuksissa käytössä. Staattisen koodinanalyysin käyttöä ei mainittu testaamiskategorian tutkimuksissa.

Kategoria	Staattinen koodianalyysi käytössä	Staattinen koodianalyysi ei käytössä
CI	12	25
Testaaminen	0	15

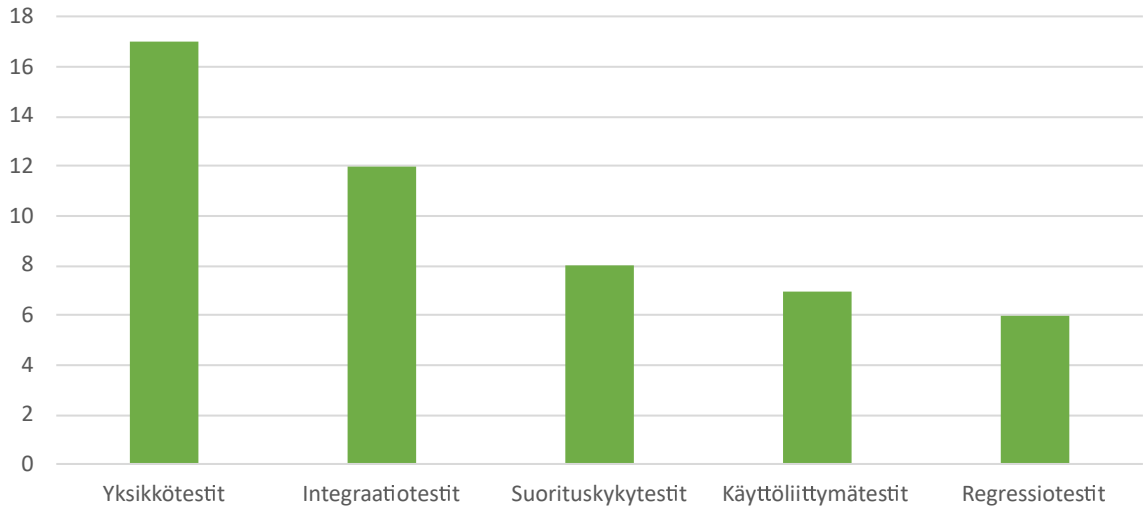
Taulukko 3. Staattisen koodianalyysin käyttö kategorioittain.

4.6 Testaustasot

Testaustasojen painotus tutkimuksissa oli samankaltainen mitä teoriaosiossa esitettiin siltä osin, että yksikkötestejä ajettiin eniten ja myös integraatiotestit ja käyttöliittymätestit olivat eniten mainituissa testaustasoissa. Jonkin verran tehtiin myös suorituskykytestejä. Testaustasoja ei mainittu kaikissa artikkeleissa, vaikka testaamista tutkittiinkin jollain tasolla jokaisessa artikkelissa. Artikkeleita joissa testaustasoa ei mainittu oli niin CI-kategoriassa kuin myös testaaminen kategoriassa.

Eniten mainitut yksikkötestit luovat monen tutkimuksen perusteella pohjan testaamiselle. Guşeilă, Bratu ja Moraru (2019) kuvaavat yksikkötestien olevan tyypillisesti kehittäjien käyttämiä testejä koodin kirjoitusvaiheessa. Niiden avulla pystytään testaamaan yksittäisiä osioita ilman, että muiden osioiden tarvitsee olla valmiita. Ne ovat myös nopeampia suorittaa ja kustannustehokkaampia kuin muut testaustasot.

Kuviossa 10 on esitetty eniten mainitut testaustasot. Joissain artikkeleissa mainittiin useampi testaustaso.



Kuvio 10. Eniten mainitut testaustasot.

4.7 Haasteet, hyödyt ja jatkotutkimusaiheet

Tutkimuksissa jatkuvalla integraatiolla nähtiin monia hyötyjä. Lu, Yang ja Qian (2014) mukaan jatkuvan integraation toteuttaminen ja automaattiset testit paransivat muun muassa kehittäjien tuottavuutta ja koodin laatua. Samassa tutkimuksessa todettiin, että jatkuvan integraation toteutus saa kehittäjät jakamaan muutokset usempiin osiin ja yhdistämään muutokset säännöllisesti. Than ja Phyu (2019) toteavat jatkuvan integraation vähentävän integraatioriskejä ja Amrit ja Meijberg (2018) tutkimuksessa jatkuvan integraation toteutuksen avulla löydettiin enemmän virheitä ja niiden korjaamiseen kuluva aika väheni.

Haasteet jatkuvassa integraatiossa tulivat esiin selkeiten testaamiseen keskittyvissä tutkimuksissa, joissa tutkittiin jollain lailla testien nopeuttamista. Tällaisia tutkimuksia joissa testien nopeuttamista tutkittiin oli 11 ja niissä käyettyjä testimenetelmiä olivat testien valinta, testien priorisointi, muutosten yhdistäminen, sekä testien ajo perustuen samankaltaisuuteen. Niissä kaikissa nähtiin haasteena testien ajamisen hitaus tai kustannukset.

Useassa artikkelissa nähtiin haasteena sekä testien ajamisen hitaus, että kustannukset. Hitaat testit hidastavat kehitystyötä ja vievät resursseja. Pienemmissä projekteissa kaikkien testien ajaminen joka muutoksen yhteydessä toimii, mutta esimerkiksi Beheshtian, Bavand ja Rigby (2022) toteavat tutkimuksessaan että suurten yritysten kuten Googlen tai Facebookin, joilla

on massiivinen määrä koodin yhdistämisiä tai kallis testi-infrastruktuuri kuten Ericssonilla, täytyy yhdistää muutoksia vähentääkseen testien ajon määrää. Esimerkiksi jos kahdeksan muutosta yhdistetään yhdeksi ja testit menevät läpi säästetään 7 testien ajoa. Toisaalta jos testit eivät mene läpi ei tiedetä heti mikä muutos on aiheuttanut sen, että testit eivät mene läpi.

Teixeira, Arrais ja Veiga (2021) tutkimuksen mukaan jatkuvan integraation käyttö on vähäisempää robotiikassa kuin muilla sovelluskehityksen aloilla. Tutkimuksessa esitettiin miten simulaatiotestit voidaan yhdistää jatkuvan integraation toteutukseen. Tutkimuksen mukaan esitettyyn toteutukseen on tehtävissä monia parannuksia, kuten simulaatiotestien onnistumisen arvioinnin kehittäminen. Rangnau ym. (2020) puolestaan esittävät että jatkotukimusaiheena he aikovat tutkia havaittujen haavoittuvuuksien korjaamista ja automaattista testien generointia sovelluksen käyttäytymisen muuttuessa.

5 Pohdinta ja yhteenveto

Tutkielman tavoitteena oli selvittää systemaattisen kirjallisuuskartoituksen avulla, minkälaista tutkimusta jatkuvaan integraatioon liittyvästä testaamisesta on tehty, minkälaisia käytäntöjä jatkuvan integraation yhteydessä tehtävään testaukseen on sovellettu, sekä minkälaisia haasteita tutkimukseen valikoiduissa artikkeleissa on havaittu jatkuvaan integraatioon tai testaamiseen liittyen.

Tutkimukseen valikoitui 52 aiheeseen sopivaa artikkelia, joiden avulla esitettiin minkälaista tutkimusta jatkuvaan integraatioon liittyvästä testaamisesta on tehty. Tutkimukset jakautuivat aikavälille 2006-2022. Artikkelit olivat keskenään erilaisia, mutta niistä oli mahdollista luoda kaksi eri kategoriaa, joilla pystyttiin havainnollistamaan tutkimusten erilaisia näkökulmia aiheen tutkimiseen. Kategorioista toisessa keskityttiin jatkuvaan integraatioon ja toisessa testaamiseen.

Varsinkin testaamiseen keskittyvissä artikkeleissa huomattiin yleisenä haasteena jatkuvan integraation keskeiset ongelmat, jotka ovat isoissa projekteissa testien hidas ajaminen sekä kustannukset. Tästä johtuen testaamiseen keskittyvissä artikkeleissa pyrittiin löytämään testien priorisoinnin tyylisillä tekniikoilla nopeampia keinoja ajaa testejä. Jatkuvaan integraatioon keskittyvissä artikkeleissa pyrittiin esimerkiksi esittämään malli jatkuvan integraation toteutuksesta.

Aiempiin kirjallisuuskartoituksiin verrattuna, tämän tutkimuksen tutkimustulokset olivat samankaltaisia eniten käytetyn CI-palvelimen ja versionhallintajärjestelmän, sekä mainittujen testaustasojen osalta. Aiemmissa kirjallisuus kartoituksissa on myös todettu, että käytettyjä jatkuvan integraation työkaluja ei välttämättä mainita kovin usein.

Esimerkiksi Shahin, Ali Babar ja Zhu (2017) kirjallisuuskartoituksessa todettiin, että vain 36.2 % heidän tutkimistaan artikkeleista mainitsivat mitä työkaluja ja millä perusteella jatkuvan integraation toteutuksessa käytettiin. Shahin, Ali Babar ja Zhu (2017) tutkimissa artikkeleissa joissa työkaluja oli mainittu, käytetyimpiä versionhallintatyökaluja olivat Subversion ja Git/GitHub ja käytetyin CI-palvelin Jenkins.

Pando ja Dávila (2022) kirjallisuuskartoituksessa todetaan, että vaikka testaamista pidetään tärkeänä heidän tutkimissaan artikkeleissa, niin käytettyjä testaustasoja ei mainita 46.5 % artikkeleista. Mainituissa testaustasoissa yksikkötestit ja integraatiotestit ovat eniten tutkittuja. Samassa Pando ja Dávila (2022) tutkimuksessa todettiin Gitin olleen mainituin versionhallintajärjestelmä ja Jenkinsin mainituin CI-palvelin.

Pando ja Dávila (2022) tutkimuksessa todetaan myös, että artikkelien luonteen ja tavoitteiden takia työkalujen maininta ei ole tarpeellista. Molemmissa kirjallisuuskartoituksissa myös todetaan artikkelien määrän painottuneen uudempiin artikkeleihin. Kummassakin tutkimuksessa artikkelit etsittiin ilman rajoitusta tutkimuksen julkaisuvuodessa.

Tämän tutkimuksen artikkeleissa vain noin 42 %:ssa mainittiin versionhallintajärjestelmä ja noin 62 %:ssa CI-palvelin. Käytetyin versionhallintajärjestelmä oli Git ja käytetyin CI-palvelin Jenkins. Noin 23 %:ssa artikkeleista ei mainita testaustasoja. Myös tässä tutkimuksessa artikkelien määrä painoittui uudempiin artikkeleihin.

Työkalujen ja artikkelien julkaisuvuosien osalta voidaan todeta, että tämän tutkimuksen tulokset vastaavat hyvin näiden kahden aiemman tutkimuksen tuloksia. Lähempänä tätä tutkimusta on Pando ja Dávila (2022) tutkimus, jossa tutkitaan jatkuvan integraation lisäksi testaamista. Shahin, Ali Babar ja Zhu (2017) tutkimuksessa puolestaan tutkitaan enemmän jatkuvan integraation toteutusta.

Teknologian kehittyessä myös käytännöt jatkuvassa integraatiossa kehittyvät ja niin sanotut parhaat käytännöt muuttuvat, kun yhä useampi tutkii tai käyttää jatkuvaa integraatiota. Näin ollen tässäkin tutkimuksessa esitetyt asiat voivat joiltain osin vanhentua uusien parempien käytäntöjen tullessa tilalle. Tällaisia ovat varsinkin työkalut, jotka kehittyivät tämänkin tutkimuksen ensimmäisenä julkaistun ja uusimpien tutkimusten välillä. Myös testien ajamiseen kehitetään uudempia ja tehokkaampia keinoja ja esimerkiksi tekoäly tulee muuttamaan varsinkin testien priorisoinnin mahdollisuuksia ellei ole jo muuttanut.

Tämän tutkimusprosessin aikana luettujen artikkelien perusteella jatkuvan integraation tutkiminen vaikuttaa monipuoliselta ja kattavalta. Luetun perusteella vaikuttaa myös siltä, että yleisistä jatkuvan integraation periaatteista on tutkimusten välillä yhteisymmärrys. Tämä ei tarkoita kuitenkaan sitä, että tutkimukset olisivat samanlaisia, vaan jatkuvasta integraatiosta

löytyy tutkimuksia eri näkökulmista. Tässä tukimuksessa mukana olevissa tutkimuksissa jatkuvaa integraatiota tutkittiin perinteisten ohjelmistojen lisäksi muun muassa korkean laskentakyvyn sovelluksissa, koneoppimismallissa, sähköjärjestelmissä ja laserlaitteiston hallintajärjestelmässä. Jatkuva integraatio ei ole siis pelkästään perinteisen ohjelmistokehityksen käytäntö vaan laajempi käytäntö testata muutoksia ja tuottaa nopeammin tuotantovalmiita sovelluksia.

Tässä tutkimuksessa rajoitteena on ollut siihen käytetyt resurssit. Tutkimuksen laajus sekä töiden ohella tehty tutkimus rajoittivat artikkelien määrää ja valittuihin artikkeleihin käytettyä aikaa. Tästä johtuen tutkimuksessa luodaan yleiskuva jatkuvan integraation tutkimuksen tilasta. Varsinkin testaaminen kategorian artikkeleiden tarkempi analysointi olisi ollut hankalaa tutkimuksen laajuuden puitteissa. Jos tutkimukseen olisi käyttänyt enemmän resursseja olisi esimerkiksi artikkelien määrää ollut hyvä kasvattaa, jotta tutkimuskentästä olisi saatu kattavampi kuva.

Jatkotutkimusaiheena olisi mielenkiintoista tutkia erilaisia keinoja testien priorisointiin liittyen esimerkiksi tekoälyä käyttäen.

Lähteet

Alégroth, Emil, Arvid Karlsson ja Alexander Radway. 2018. “Continuous Integration and Visual GUI Testing: Benefits and Drawbacks in Industrial Practice”. Teoksessa *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST),* 172–181. https://doi.org/10.1109/ICST.2018.00026.

Ali, Sadia, Yaser Hafeez, Shariq Hussain ja Shunkun Yang. 2019. “Enhanced regression testing technique for agile software development and continuous integration strategies”. *Software Quality Journal* 28, numero 2 (syyskuu): 397–423. https://doi.org/10.1007/s11219-019-09463-4.

Amrit, Chintan, ja Yoni Meijberg. 2018. “Effectiveness of Test-Driven Development and Continuous Integration: A Case Study”. *IT Professional* 20 (1): 27–35. https://doi.org/10.1109/MITP.2018.014121554.

Bai, Xiaoying, W.T. Tsai, R. Paul, Techeng Shen ja Bing Li. 2001. “Distributed end-to-end testing management”. Teoksessa *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference,* 140–151. https://doi.org/10.1109/EDOC.2001.950430.

Bavand, Amir Hossein, ja Peter C. Rigby. 2021. “Mining Historical Test Failures to Dynamically Batch Tests to Save CI Resources”. Teoksessa *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME),* 217–226. https://doi.org/10.1109/ICSME52107.2021.00026.

Beheshtian, Mohammad Javad, Amir Hossein Bavand ja Peter C. Rigby. 2022. “Software Batch Testing to Save Build Test Resources and to Reduce Feedback Time”. *IEEE Transactions on Software Engineering* 48 (8): 2784–2801. https://doi.org/10.1109/TSE.2021.3070269.

Fowler, Martin. 2006. “Continuous Integration”. https://martinfowler.com/articles/continuousIntegration.html.

Guşeilă, Ligia Georgeta, Dragoş-Vasile Bratu ja Sorin-Aurel Moraru. 2019. “Continuous Testing in the Development of IoT Applications”. Teoksessa *2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI),* 1–6. https://doi.org/10.1109/ISSI47111.2019.9043692.

Hamdan, Saba, ja Suad Alramouni. 2015. “A Quality Framework for Software Continuous Integration”. *Procedia Manufacturing* 3:2019–2025. https://doi.org/10.1016/j.promfg.2015.07.249.

Hilton, Michael, Timothy Tunnell, Kai Huang, Darko Marinov ja Danny Dig. 2016. “Usage, costs, and benefits of continuous integration in open-source projects”. Teoksessa *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering.* ACM, elokuu. https://doi.org/10.1145/2970276.2970358.

“Julkaisufoorumi”. 2023. https://www.tsv.fi/julkaisufoorumi/haku.php.

Kato, Daiju, Ayumu Shimizu ja Hiroshi Ishikawa. 2022. “Quality Classification for Testing Work in DevOps”. Teoksessa *Proceedings of the 14th International Conference on Management of Digital EcoSystems.* ACM, lokakuu. https://doi.org/10.1145/3508397.3564840.

Laukkanen, Eero, Juha Itkonen ja Casper Lassenius. 2017. “Problems, causes and solutions when adopting continuous delivery—A systematic literature review”. *Information and Software Technology* 82 (helmikuu): 55–79. https://doi.org/10.1016/j.infsof.2016.10.001.

Leung, H.K.N., ja L. White. 1990. “A study of integration testing and software regression at the integration level”. Teoksessa *Proceedings. Conference on Software Maintenance 1990,* 290–301. https://doi.org/10.1109/ICSM.1990.131377.

Lima, Jackson A. Prado, ja Silvia R. Vergilio. 2020. “Test Case Prioritization in Continuous Integration environments: A systematic mapping study”. *Information and Software Technology* 121 (toukokuu): 106268. https://doi.org/10.1016/j.infsof.2020.106268.

Lu, Jixiang, Zhihong Yang ja Junxia Qian. 2014. “Implementation of continuous integration and automated testing in software development of smart grid scheduling support system”. Teoksessa *2014 International Conference on Power System Technology,* 2441–2446. https://doi.org/10.1109/POWERCON.2014.6993503.

Marijan, Dusica, Marius Liaaen ja Sagar Sen. 2018. “DevOps Improvements for Reduced Cycle Times with Integrated Test Optimizations for Continuous Integration”. Teoksessa *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC),* 01:22–27. https://doi.org/10.1109/COMPSAC.2018.00012.

Niranjan, D.R., ja Mohana. 2022. “Jenkins Pipelines: A Novel Approach to Machine Learning Operations (MLOps)”. Teoksessa *2022 International Conference on Edge Computing and Applications (ICECAA),* 1292–1297. https://doi.org/10.1109/ICECAA55415.2022.9936252.

Pando, B., ja A. Dávila. 2022. “Software Testing in the DevOps Context: A Systematic Mapping Study”. *Programming and Computer Software* 48, numero 8 (joulukuu): 658–684. ISSN: 1608-3261. https://doi.org/10.1134/s0361768822080175.

Penson, Wade, Eric Huang, Dana Klamut, Eliana Wardle, Graeme Douglas, Scott Fazackerley ja Ramon Lawrence. 2017. “Continuous integration platform for Arduino embedded software”. Teoksessa *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE),* 1–4. https://doi.org/10.1109/CCECE.2017.7946696.

Petersen, Kai, Robert Feldt, Shahid Mujtaba ja Michael Mattsson. 2008. “Systematic Mapping Studies in Software Engineering”. Teoksessa *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering,* 68–77. EASE’08. Italy: BCS Learning & Development Ltd.

Rangnau, Thorsten, Remco v. Buijtenen, Frank Fransen ja Fatih Turkmen. 2020. “Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines”. Teoksessa *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC),* 145–154. https://doi.org/10.1109/EDOC49727.2020.00026.

RedHat. 2022. “What is CI/CD?” https://www.redhat.com/en/topics/devops/what-is-ci-cd.

Runeson, P. 2006. “A survey of unit testing practices”. *IEEE Software* 23 (4): 22–29. https://doi.org/10.1109/MS.2006.91.

Shahin, Mojtaba, Muhammad Ali Babar ja Liming Zhu. 2017. “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices”. *IEEE Access* 5:3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629.

Soares, Eliezio, Gustavo Sizilio, Jadson Santos, Daniel Alencar da Costa ja Uirá Kulesza. 2022. “The effects of continuous integration on software development: a systematic literature review”. *Empirical Software Engineering* 27, numero 3 (maaliskuu). https://doi.org/10.1007/s10664-021-10114-1.

Stolberg, Sean. 2009. “Enabling Agile Testing through Continuous Integration”. Teoksessa *2009 Agile Conference,* 369–374. https://doi.org/10.1109/AGILE.2009.16.

Teixeira, Sérgio, Rafael Arrais ja Germano Veiga. 2021. “Cloud Simulation for Continuous Integration and Deployment in Robotics”. Teoksessa *2021 IEEE 19th International Conference on Industrial Informatics (INDIN),* 1–8. https://doi.org/10.1109/INDIN45523.2021.9557476.

Than, Phyu Phyu, ja Myat Pwint Phyu. 2019. “Continuous integration for Laravel applications with GitLab”. Teoksessa *Proceedings of the 1st International Conference on Advanced Information Science and System.* ACM, marraskuu. https://doi.org/10.1145/3373477.3373479. https://doi.org/10.1145/3373477.3373479.

Tronge, Jake, Jieyang Chen, Patricia Grubel, Tim Randles, Rusty Davis, Quincy Wofford, Steven Anaya ja Qiang Guan. 2021. “BeeSwarm: Enabling Parallel Scaling Performance Measurement in Continuous Integration for HPC Applications”. Teoksessa *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE),* 1136–1140. https://doi.org/10.1109/ASE51524.2021.9678805.

Vasilescu, Bogdan, Yue Yu, Huaimin Wang, Premkumar Devanbu ja Vladimir Filkov. 2015. “Quality and productivity outcomes relating to continuous integration in GitHub”. Teoksessa *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering.* ESEC/FSE’15. ACM, elokuu. https://doi.org/10.1145/2786805.2786850.

Vassallo, Carmine, Sebastiano Panichella, Fabio Palomba, Sebastian Proksch, Harald C. Gall ja Andy Zaidman. 2019. “How developers engage with static analysis tools in different contexts”. *Empirical Software Engineering* 25, numero 2 (marraskuu): 1419–1457. https://doi.org/10.1007/s10664-019-09750-5.

Wongkampoo, Supaket, ja Supapom Kiattisin. 2017. “Atom-Task Precondition Technique to Optimize Large Scale GUI Testing Time based on Parallel Scheduling Algorithm”. Teoksessa *2017 21st International Computer Science and Engineering Conference (ICSEC),* 1–5. https://doi.org/10.1109/ICSEC.2017.8443913.

Hyväksytyt artikkelit

Alégroth, Emil, Arvid Karlsson ja Alexander Radway. 2018. “Continuous Integration and Visual GUI Testing: Benefits and Drawbacks in Industrial Practice”. Teoksessa *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST),* 172–181. https://doi.org/10.1109/ICST.2018.00026.

Ali, Sadia, Yaser Hafeez, Shariq Hussain ja Shunkun Yang. 2019. “Enhanced regression testing technique for agile software development and continuous integration strategies”. *Software Quality Journal* 28, numero 2 (syyskuu): 397–423. https://doi.org/10.1007/s11219-019-09463-4.

Amrit, Chintan, ja Yoni Meijberg. 2018. “Effectiveness of Test-Driven Development and Continuous Integration: A Case Study”. *IT Professional* 20 (1): 27–35. https://doi.org/10.1109/MITP.2018.014121554.

Arcuri, Andrea, José Campos ja Gordon Fraser. 2016. “Unit Test Generation During Software Development: EvoSuite Plugins for Maven, IntelliJ and Jenkins”. Teoksessa *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST),* 401–408. https://doi.org/10.1109/ICST.2016.44.

Azizi, Maral. 2021. “A Tag-based Recommender System for Regression Test Case Prioritization”. Teoksessa *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW),* 146–157. https://doi.org/10.1109/ICSTW52544.2021.00035.

Bani Muhamad, Fachrul Pralienka, Riyanarto Sarno, Adhatus Solichah Ahmadiyah ja Siti Rochimah. 2016. “Visual GUI testing in continuous integration environment”. Teoksessa *2016 International Conference on Information & Communication Technology and Systems (ICTS),* 214–219. https://doi.org/10.1109/ICTS.2016.7910301.

Bavand, Amir Hossein, ja Peter C. Rigby. 2021. “Mining Historical Test Failures to Dynamically Batch Tests to Save CI Resources”. Teoksessa *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME),* 217–226. https://doi.org/10.1109/ICSME52107.2021.00026.

Beheshtian, Mohammad Javad, Amir Hossein Bavand ja Peter C. Rigby. 2022. “Software Batch Testing to Save Build Test Resources and to Reduce Feedback Time”. *IEEE Transactions on Software Engineering* 48 (8): 2784–2801. https://doi.org/10.1109/TSE.2021.3070269.

Biringa, Chidera, ja Gökhan Kul. 2021. “Automated User Experience Testing through Multi-Dimensional Performance Impact Analysis”. Teoksessa *2021 IEEE/ACM International Conference on Automation of Software Test (AST),* 125–128. https://doi.org/10.1109/AST52587.2021.00024.

Boone, Casper, Carolin Brandt ja Andy Zaidman. 2022. “Fixing Continuous Integration Tests From Within the IDE With Contextual Information”. Teoksessa *2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC),* 287–297. https://doi.org/10.1145/3524610.3527908.

Cannizzo, Fabrizio, Robbie Clutton ja Raghav Ramesh. 2008. “Pushing the Boundaries of Testing and Continuous Integration”. Teoksessa *Agile 2008 Conference,* 501–505. https://doi.org/10.1109/Agile.2008.31.

Feoktistov, A.G., S.A. Gorsky, I.A. Sidorov ja A. Tchernykh. 2019. “Continuous Integration in Distributed Applied Software Packages”. Teoksessa *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO),* 1531–1536. https://doi.org/10.23919/MIPRO.2019.8757002.

Guşeilă, Ligia Georgeta, Dragoş-Vasile Bratu ja Sorin-Aurel Moraru. 2019. “Continuous Testing in the Development of IoT Applications”. Teoksessa *2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI),* 1–6. https://doi.org/10.1109/ISSI47111.2019.9043692.

Hamdan, Saba, ja Suad Alramouni. 2015b. “A Quality Framework for Software Continuous Integration”. *Procedia Manufacturing* 3:2019–2025. https://doi.org/10.1016/j.promfg.2015.07.249. https://doi.org/10.1016/j.promfg.2015.07.249.

Hill, James H., Douglas C. Schmidt, Adam A. Porter ja John M. Slaby. 2008. “CiCUTS: Combining System Execution Modeling Tools with Continuous Integration Environments”. Teoksessa *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008),* 66–75. https://doi.org/10.1109/ECBS.2008.20.

Holmes, A., ja M. Kellogg. 2006. “Automating functional tests using Selenium”. Teoksessa *AGILE 2006 (AGILE’06),* 6 pp.–275. https://doi.org/10.1109/AGILE.2006.19.

Hung, Phan Duy, ja Do Thai Giang. 2019. “Continuous Integration for Android Application Development and Training”. Teoksessa *Proceedings of the 2019 3rd International Conference on Education and Multimedia Technology - ICEMT 2019.* ACM Press. https://doi.org/10.1145/3345120.3345158. https://doi.org/10.1145/3345120.3345158.

Jiang, Bo, ja W.K. Chan. 2016. “Testing and Debugging in Continuous Integration with Budget Quotas on Test Executions”. Teoksessa *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS),* 439–447. https://doi.org/10.1109/QRS.2016.66.

Kim, Eun Ha, Jong Chae Na ja Seok Moon Ryoo. 2009a. “Implementing an Effective Test Automation Framework”. Teoksessa *2009 33rd Annual IEEE International Computer Software and Applications Conference,* 2:534–538. https://doi.org/10.1109/COMPSAC.2009.188.

———. 2009b. “Test Automation Framework for Implementing Continuous Integration”. Teoksessa *2009 Sixth International Conference on Information Technology: New Generations,* 784–789. https://doi.org/10.1109/ITNG.2009.260.

Kim, Seojin, Sungjin Park, Jeonghyun Yun ja Younghoo Lee. 2008. “Automated Continuous Integration of Component-Based Software: An Industrial Experience”. Teoksessa *2008 23rd IEEE/ACM International Conference on Automated Software Engineering,* 423–426. https://doi.org/10.1109/ASE.2008.64.

Knauss, Eric, Miroslaw Staron, Wilhelm Meding, Ola Söder, Agneta Nilsson ja Magnus Castell. 2015. “Supporting Continuous Integration by Code-Churn Based Test Selection”. Teoksessa *2015 IEEE/ACM 2nd International Workshop on Rapid Continuous Software Engineering,* 19–25. https://doi.org/10.1109/RCoSE.2015.11.

Lai, Sen-Tarng, ja Fang-Yie Leu. 2015. “Applying Continuous Integration for Reducing Web Applications Development Risks”. Teoksessa *2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA),* 386–391. https://doi.org/10.1109/BWCCA.2015.54.

Liang, Jingjing, Sebastian Elbaum ja Gregg Rothermel. 2018. “Redefining Prioritization: Continuous Prioritization for Continuous Integration”. Teoksessa *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE),* 688–698. https://doi.org/10.1145/3180155.3180213.

Lim, Zui Young, Jia Min Chua, Kaiting Yang, Wei Shin Tan ja Yinn Chai. 2020. “Web accessibility testing for Singapore government e-services”. Teoksessa *Proceedings of the 17th International Web for All Conference.* ACM, huhtikuu. https://doi.org/10.1145/3371300.3383353. https://doi.org/10.1145/3371300.3383353.

Liu, Hehui, Zhongjie Li, Jun Zhu, Huafang Tan ja Heyuan Huang. 2009. “A Unified Test Framework for Continuous Integration Testing of SOA Solutions”. Teoksessa *2009 IEEE International Conference on Web Services,* 880–887. https://doi.org/10.1109/ICWS.2009.28.

Lu, Jixiang, Zhihong Yang ja Junxia Qian. 2014. “Implementation of continuous integration and automated testing in software development of smart grid scheduling support system”. Teoksessa *2014 International Conference on Power System Technology,* 2441–2446. https://doi.org/10.1109/POWERCON.2014.6993503.

Marijan, Dusica, ja Marius Liaaen. 2016. “Effect of Time Window on the Performance of Continuous Regression Testing”. Teoksessa *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME),* 568–571. https://doi.org/10.1109/ICSME.2016.77.

Marijan, Dusica, Marius Liaaen ja Sagar Sen. 2018b. “DevOps Improvements for Reduced Cycle Times with Integrated Test Optimizations for Continuous Integration”. Teoksessa *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC),* 01:22–27. https://doi.org/10.1109/COMPSAC.2018.00012.

Milojković, Jovan, Vladimir Ćirić ja Dejan Rančić. 2020. “Design and Implementation of Cluster Based Parallel System for Software Testing”. Teoksessa *2020 Zooming Innovation in Consumer Technologies Conference (ZINC),* 276–279. https://doi.org/10.1109/ZINC50678.2020.9161436.

Ming, Fan, ZuDe Zhou ja Zhengying Li. 2016. “The design and implement of the cross-platform mobile automated testing framework”. Teoksessa *2016 5th International Conference on Computer Science and Network Technology (ICCSNT),* 182–185. https://doi.org/10.1109/ICCSNT.2016.8070144.

Niranjan, D.R., ja Mohana. 2022. “Jenkins Pipelines: A Novel Approach to Machine Learning Operations (MLOps)”. Teoksessa *2022 International Conference on Edge Computing and Applications (ICECAA),* 1292–1297. https://doi.org/10.1109/ICECAA55415.2022.9936252.

Niæetin, Stefan, Robert Šandor, Goran Stupar ja Nikola Tesliæ. 2018. “Maximizing the Efficiency of Automotive Software Development Environment Using Open Source Technologies”. Teoksessa *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin),* 1–3. https://doi.org/10.1109/ICCE-Berlin.2018.8576212.

Oliveira Neto, Francisco Gomes de, Azeem Ahmad, Ola Leifler, Kristian Sandahl ja Eduard Enoiu. 2018. “Improving Continuous Integration with Similarity-Based Test Case Selection”. Teoksessa *2018 IEEE/ACM 13th International Workshop on Automation of Software Test (AST),* 39–45.

Pachev, Benjamin, Georgia Stuart ja Clint Dawson. 2022. “Continuous Integration for HPC with Github Actions and Tapis”. Teoksessa *Practice and Experience in Advanced Research Computing*. PEARC ’22. Boston, MA, USA: Association for Computing Machinery. ISBN: 9781450391610. https://doi.org/10.1145/3491418.3535124. https://doi.org/10.1145/3491418.3535124.

Penson, Wade, Eric Huang, Dana Klamut, Eliana Wardle, Graeme Douglas, Scott Fazackerley ja Ramon Lawrence. 2017. “Continuous integration platform for Arduino embedded software”. Teoksessa *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE),* 1–4. https://doi.org/10.1109/CCECE.2017.7946696.

Pratama, Mohammad Rizky, ja Dana Sulistiyo Kusumo. 2021. “Implementation of Continuous Integration and Continuous Delivery (CI/CD) on Automatic Performance Testing”. Teoksessa *2021 9th International Conference on Information and Communication Technology (ICoICT),* 230–235. https://doi.org/10.1109/ICoICT52021.2021.9527496.

Rabuzin, Tin, Maxime Baudette ja Luigi Vanfretti. 2017. “Implementation of a continuous integration workflow for a power system Modelica library”. Teoksessa *2017 IEEE Power & Energy Society General Meeting,* 1–5. https://doi.org/10.1109/PESGM.2017.8274618.

Rangnau, Thorsten, Remco v. Buijtenen, Frank Fransen ja Fatih Turkmen. 2020. “Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines”. Teoksessa *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC),* 145–154. https://doi.org/10.1109/EDOC49727.2020.00026.

Rathod, Nikhil, ja Anil Surve. 2015. “Test orchestration a framework for Continuous Integration and Continuous deployment”. Teoksessa *2015 International Conference on Pervasive Computing (ICPC),* 1–5. https://doi.org/10.1109/PERVASIVE.2015.7087120.

Reichardt, Paul, Wladimir Hofmann, Tobias Reggelin ja Sebastian Lang. 2021. “From Logistics Process Models to Automated Integration Testing: Proof-of-Concept Using Open-Source Simulation Software”. Teoksessa *2021 Winter Simulation Conference (WSC),* 1–11. https://doi.org/10.1109/WSC52266.2021.9715310.

Rubasinghe, Iresha, Dulani Meedeniya ja Indika Perera. 2018. “Traceability Management with Impact Analysis in DevOps based Software Development”. Teoksessa *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI),* 1956–1962. https://doi.org/10.1109/ICACCI.2018.8554399.

Seth, Nikita, ja Rishi Khare. 2015. “ACI (automated Continuous Integration) using Jenkins: Key for successful embedded Software development”. Teoksessa *2015 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS),* 1–6. https://doi.org/10.1109/RAECS.2015.7453279.

Soni, Mitesh. 2015. “End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery”. Teoksessa *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM),* 85–89. https://doi.org/10.1109/CCEM.2015.29.

Straubinger, Philipp, ja Gordon Fraser. 2022. “Gamekins: Gamifying Software Testing in Jenkins”. Teoksessa *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion),* 85–89. https://doi.org/10.1145/3510454.3516862.

Teixeira, Sérgio, Rafael Arrais ja Germano Veiga. 2021. “Cloud Simulation for Continuous Integration and Deployment in Robotics”. Teoksessa *2021 IEEE 19th International Conference on Industrial Informatics (INDIN),* 1–8. https://doi.org/10.1109/INDIN45523.2021.9557476.

Than, Phyu Phyu, ja Myat Pwint Phyu. 2019. “Continuous integration for Laravel applications with GitLab”. Teoksessa *Proceedings of the 1st International Conference on Advanced Information Science and System.* ACM, marraskuu. https://doi.org/10.1145/3373477.3373479. https://doi.org/10.1145/3373477.3373479.

Tronge, Jake, Jieyang Chen, Patricia Grubel, Tim Randles, Rusty Davis, Quincy Wofford, Steven Anaya ja Qiang Guan. 2021b. “BeeSwarm: Enabling Parallel Scaling Performance Measurement in Continuous Integration for HPC Applications”. Teoksessa *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE),* 1136–1140. https://doi.org/10.1109/ASE51524.2021.9678805.

Wongkampoo, Supaket, ja Supapom Kiattisin. 2017. “Atom-Task Precondition Technique to Optimize Large Scale GUI Testing Time based on Parallel Scheduling Algorithm”. Teoksessa *2017 21st International Computer Science and Engineering Conference (ICSEC),* 1–5. https://doi.org/10.1109/ICSEC.2017.8443913.

Xiao, Lei, Huaikou Miao, Tingting Shi ja Yu Hong. 2020. “LSTM-based deep learning for spatial–temporal software testing”. *Distributed and Parallel Databases* 38, numero 3 (toukokuu): 687–712. https://doi.org/10.1007/s10619-020-07291-1. https://doi.org/10.1007/s10619-020-07291-1.

Yanjari, Ignacio, Beatriz Marín ja Giovanni Giachetti. 2022. “An open-source framework for cross-platform testing in agile projects”. Teoksessa *2022 41st International Conference of the Chilean Computer Science Society (SCCC),* 1–8. https://doi.org/10.1109/SCCC57464.2022.10000346.

Yu, Bing, Jin Liu, XiaoWei Zhou ja Baoran An. 2020. “Simulation Verification Platform for Complex Facility Control System”. Teoksessa *2020 Chinese Automation Congress (CAC),* 359–363. https://doi.org/10.1109/CAC51589.2020.9326558.

Liitteet

A Pilotin hakutulokset

Haku 1:

Haussa 1 kartoitettiin minkälaisia määriä tuloksia tulee hakemalla artikkeleita joissa mainitaan sekä jatkuva integraatio, että testit. Tuloksia tuli ensimmäisestä hakukoneesta Scopuksesta jo niin paljon, että päätettiin siirtyä seuraavaan hakuun.

Haettu 03.03.2023

Tietokanta	Hakusanat	Tuloksia
Scopus	TITLE-ABS-KEY ("continuous integration"AND tests)	702

Taulukko 4. Pilotin 1. haku

Haku 2:

Haussa 2 muutoksia ei juurikaan tehty vaan koitettiin miten hakutuloksien määrä muuttuu hieman toista hakusanaa muuttamalla. Tuloksena oli vieläkin tutkimuksen laajuuten nähden liian paljon artikkeleita.

Haettu 03.03.2023

Tietokanta	Hakusanat	Tuloksia
Scopus	TITLE-ABS-KEY ("continuous integration"AND testing)	772

Taulukko 5. Pilotin 2. haku

Haku 3:

Haussa 3 pyrittiin saamaan artikkeleita aiempaa vähemmän rajaamalla hakua uudella hakusanalla. Hakutuloksia tulikin aiempaa vähemmän ensimmäisestä tietokannasta. Kuitenkin

yhteensä molemmista tietokannoista haettuna hakutulosten määrä oli samaa luokkaa kuin aiemmissa hauissa.

Haettu 03.03.2023

Tietokanta	Hakusanat	Tuloksia
Scopus	TITLE-ABS-KEY ("continuous integration"AND testing AND methods)	199
IEEE	("All Metadata":continuous integration) AND ("All Metadata":testing) AND ("All Metadata":methods)	465

Taulukko 6. Pilotin 3. haku

Haku 4:

Haussa 4 koitettiin miten *tai* sanan käyttö vaikuttaa hakutulosten määrään.

Haettu 03.03.2023

Tietokanta	Hakusanat	Tuloksia
Scopus	TITLE-ABS-KEY ("continuous integration"AND testing OR tests AND methods)	267

Taulukko 7. Pilotin 4. haku

Haku 5:

Haussa 5 koitettiin saada hakutulosten määrää karsittua, mutta näillä hakusanoilla tuloksia karsiutui liikaa.

Haettu 03.03.2023

Tietokanta	Hakusanat	Tuloksia
Scopus	TITLE-ABS-KEY ("continuous integration"AND "testing methods")	12

Taulukko 8. Pilotin 5. haku

Haku 6:

Haussa 6 hakutulosten määrää saatiin kasvatettua edellisestä.

Haettu 03.03.2023

Tietokanta	Hakusanat	Tuloksia
Scopus	TITLE-ABS-KEY ("continuous integration"AND testing AND methods AND "software development")	70
IEEE	("All Metadata":continuous integration) AND ("All Metadata":testing) AND ("All Metadata":methods) AND ("All Metadata":software development)	74

Taulukko 9. Pilotin 6. haku