TAMPEREEN
AMMATTIKORKEAKOULU

VERSIONVIENNIN SUJUVOITTAMINEN

Ville Talvitie

Opinnäytetyö
Huhtikuu 2018
Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

TALVITIE, VILLE:
Versionviennin sujuvoittaminen

Opinnäytetyö 30 sivua, joista liitteitä 3 sivua
Huhtikuu 2018

Opinnäytetyön tavoitteena oli parantaa toimeksiantajan Evolvit Oy:n asiakkaalle tehtävien versiopäivitysten sujuvuutta ja vähentää näihin liittyvien inhimillisten virheiden määrää. Versiopäivitysten julkaisemista hidastivat haasteet, jotka aiheuttivat tarpeettomia virheilmoituksia ja viiveitä prosessiin. Tarkoituksena oli selvittää, miksi versiopäivityksen yhteydessä yksi asiakkaan palvelin jäi virhetilaan eikä käynnistänyt palvelimella suoritettavia palveluita uudelleen. Selvityksen lisäksi tarkoituksena oli korjata tämä vikatilanne. Opinnäytetyössä selvitettiin virheen syytä tapaustutkimuksena.

Työn tuloksena vikatilanteeseen johtavat syyt saatiin selvitettyä ja luotua toimiva korjaus. Korjaus ratkaisi versiopäivityksen julkaisun yhteydessä ilmenneen virhetilanteen tavoitellusti.

Jatkoselvityksen alaiseksi jäi sopivan kuormitustestaustyökalun etsintä, sillä korjauksen luotettava testaaminen testipalvelimella osoittautui haastavaksi palvelimen matalan kuormitustason vuoksi.

Asiasanat: versiopäivitys, jatkuva integraatio

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Development

TALVITIE, VILLE:
Streamlining Software Releases

Bachelor's thesis 30 pages, appendices 3 pages
April 2018

The purpose of this thesis was to improve the fluency of software releases made for a customer of the company Evolvit Oy, who was the client for this thesis. Releasing new software versions had challenges, which impacted the process negatively by causing errors and delays. The thesis' aim was to find out why one of the servers used to host the software would not restart successfully after a new version was released. The thesis work was carried out as a case study.

As a result, the cause for the error was found and a working fix was established.

Investigating and choosing a suitable load testing tool is left under further study, as the need for such tool rose during the testing of the fix on a test server. The normal server load in the test environment was not sufficient to reliably assess the validity of the fix.

Key words: version update, continuous integration

SISÄLLYS

1 JOHDANTO 6
2 JATKUVA INTEGRAATIO 7
2.1 Mitä on jatkuva integraatio 7
2.2 Mitä hyötyä on jatkuvasta integraatiosta 7
2.3 Jatkuvan integraation jälkeen 7
3 PROJEKTIYMPÄRISTÖ 9
3.1 Git 10
3.2 TeamCity 10
3.3 YouTrack 11
4 KOONTIVERSION JULKAISU EPÄONNISTUU 12
4.1 Ongelma 12
4.2 Selvitystyö 13
5 KOONTIVERSION JULKAISU TOIMII JÄLLEEN 20
6 POHDINTA 26
LÄHTEET 27
LIITTEET 28
Liite 1. Ajastuspalvelun sammutuksen lähdekoodi ennen korjauksia 28
Liite 2. Ajastuspalvelun sammutuksen lähdekoodi korjausten jälkeen 29
Liite 3. Korjauskustannuslaskelmia vastaanottavan työn lähdekoodi ennen ja jälkeen korjauksien 30

LYHENTEET JA TERMIT

CI	Continuous Integration, jatkuva integraatio
IIS	Internet Information Services, Microsoftin palvelinympäristö internetsivujen ja palveluiden isännöintiin
MSBuild	Visual Studion koontiversiot koostava kokonaisuus
PowerShell	Microsoftin komentorivitulkki
Projekti	Sisältää ohjelmiston lähdekoodit ja tarvittavat tiedostot, jotta ohjelma voidaan kääntää
Soluutio	Kokonaisuus, joka voi sisältää yhden tai useamman projektin Visual Studiossa
TeamCity	JetBrains Oy:n jatkuvan integraation ympäristö
Visual Studio	Microsoftin graafinen ohjelmointiympäristö
XXXX-1234	Tikettijärjestelmän tiketit ovat muotoa projektin tunnus ja tiketin numero
YouTrack	Projektin- ja tikettienhallintajärjestelmä
.ps1	PowerShell-skriptin tiedostotunniste

1 JOHDANTO

Opinnäytetyön toimeksiantaja on työnantajani Evolvit Oy, joka on perustettu vuonna 2007. Evolvit tarjoaa pääasiallisesti IT-palveluratkaisuja finanssitoimialan yrityksille sekä Business Intelligence -ratkaisuja. Yrityksellä on tällä hetkellä toimipisteet Espoossa, Tampereella, Jyväskylässä ja Lahdessa. Työntekijöitä on kaikissa toimipisteissä yhteensä noin sata henkilöä ja Tampereen toimipisteessä noin kolmekymmentä henkeä. (Evolvit Oy 2016.) Yrityksen liikevaihto vuonna 2015 oli 7,4 miljoonaa euroa. (Kauppalehti 2016.)

Toimeksiantajalla on asiakas, jolle on toteutettu toiminnanohjausjärjestelmä ja verkkokauppa vuonna 2014 sekä asiakkaan kumppaniportaali 2015. Järjestelmät on toteutettu Microsoft-tekniikoita hyödyntäen eli ohjelmointiin on käytetty ASP.NET MVC:tä, joka on Microsoftin ohjelmistokehys internetpalveluiden ja sivujen luomiseen. Järjestelmät pyörivät Internet Information Services (IIS)- alustaisilla palvelimilla.

Asiakkaalle tehdään tavallisesti kerran viikossa tuotantopäivitys, jossa viedään korjauksia ja mahdollisesti uusia ominaisuuksia tuotantopalvelimille jatkuvaa integraatiota hyödyntäen. Tuotantopäivityksen julkaisemisen prosessissa on tällä hetkellä ongelmia, jotka hidastavat tuotantopäivityksen suorittamista ja aiheuttavat turhia virheilmoituksia sekä tarpeettomia kuluja.

Opinnäytetyön tavoitteena on parantaa asiakkaalle tehtävien tuotantopäivitysten sujuvuutta ja vähentää inhimillisten virheiden määrää tuotantopäivityksiin liittyen. Tarkoituksena on selvittää, miksi päivityksen viennin yhteydessä eräs asiakkaan palvelimista kaatuu usein ja mahdollisuuksien mukaan korjata tämän aiheuttava ongelma. Tämän lisäksi tarkoituksena on luoda automaattinen tarkistus, joka tarkistaa asiakkaan palvelimien tilan automaattisesti versionviennin jälkeen ja tarvittaessa hälyttää versiopäivitystä tekevän henkilön huomion.

2 JATKUVA INTEGRAATIO

2.1 Mitä on jatkuva integraatio

Martin Fowlerin (2006) mukaan jatkuva integraatio on ohjelmistokehityksen menetelmä, jossa kehitystiimin jäsenet integroivat työnsä usein, vähintään kerran päivässä. Jokaisen integraation toimivuus todennetaan automaattisen uuden koontiversion luonnin ja automatisoitujen testien avulla. Jatkuva integraatio valvoo lähdekoodia, luo uuden koontiversion jokaisen versionhallinnan muutoksen jälkeen, testaa toimivuuden ja ilmoittaa kehittäjiä automaattisesti ja välittömästi, jos ongelmia ilmenee (Richardson 2006).

Jatkuva integraatio syntyi tarpeesta mahdollistaa muista erillään toimivien kehittäjien muutosten integroiminen muun kehitystiimin lähdekoodiin ilman, että joudutaan odottamaan päiviä tai jopa viikkoja, sillä tämä odottelu luo useita liittämisen aikaisia konflikteja, vaikeasti korjattavia ohjelmointivirheitä, eriäviä lähdekoodityylejä ja mahdollisesti toistettua työtä (Guckenheimer 2017).

2.2 Mitä hyötyä on jatkuvasta integraatiosta

Jatkuva integraatio vaatii, että kehittäjät liittävät tekemänsä muutokset versionhallintaan usein, mikä vähentää lähdekoodin liittämisen aikana tapahtuvien konfliktien mahdollisuutta ja määrää. Kun jatkuvaan integrointiin on liitetty automaattiset testit saavat kehittäjät mahdolliset virheet kiinni lähes välittömästi, jolloin niiden korjaaminen on helpompaa ja paljon vähemmän aikaa vievää kuin ilman välitöntä palautetta antavaa jatkuvan integroinnin järjestelmää. Tämä takaa eri koontiversioille tasaveroisen laadun. (Guckenheimer 2017). Myös Fowlerin (2006) mukaan jatkuvan integraation suurin ja laaja-alaisin hyöty on vähentynyt riski.

2.3 Jatkuvan integraation jälkeen

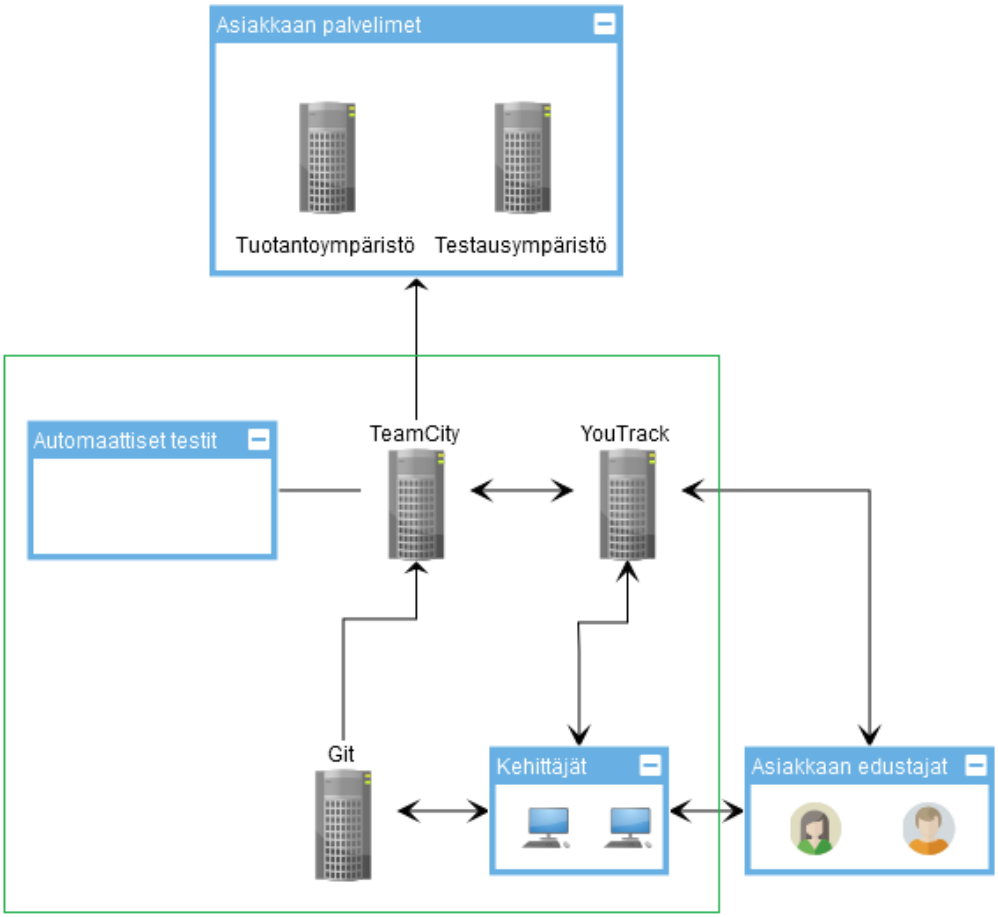
Jatkuvan integraation luonnollinen jatke on jatkuva toimitus, jossa jatkuvan integraation avulla automaattisesti luodut ja testatut koontiversiot ovat käytännössä julkaisuvalmiina yhden napin takana. Jatkuvan toimituksen järjestelmä säästää kehittäjien aikaa vähentä-

mällä manuaalisten vaiheiden ja siten samalla virheiden määrää julkaisuprosessissa. Pahimmillaan kehittäjän pitää valita oikean koontiversion kääntäminen useiden eri koontikonfiguraatioiden joukosta, odotella koontiversion kääntämiseen kuluva aika ja tämän jälkeen kopioida kaikki versiopäivitystä varten tarvittavat tiedostot, esimerkiksi tietokantamuutoksia varten luodut skriptitiedostot, päivitettävään ympäristöön ja viedä tiedostot oikeisiin sijainteihinsa ja suorittaa tarvittavat asennustoimet. Sen ollessa mahdollista ympäristörajoitteiden kuten virtuaalisten erillisverkkojen salliessa, jatkuva toimitus poistaa tämän kaiken ylimääräisen kehittäjien harteilta.

Jatkuvalle toimitukselle on jatkeena jatkuva julkaisu, joka on kokonaan automatisoitu jatke jatkuvalle integraatiolle. Jokainen uusi muutos on myös uusi julkaisu (Huotarinen 2016) eli kun kehittäjä liittää uuden ominaisuuden tai korjauksen versionhallintaan, menee se julkaisuprosessin läpi suoraan käyttöön, kunhan se läpäisee automaattiset testit.

3 PROJEKTIYMPÄRISTÖ

Ympäristö koostuu käytännössä kahdesta eri kokonaisuudesta, jotka ovat projektiympäristö ja asiakkaan palvelimet (kuvio 1). Projektiympäristö sisältää versionhallinnan (Git), projektinhallinta- ja tikettijärjestelmän (YouTrack) ja jatkuvan integraation järjestelmän (TeamCity).



Kuvio 1. Ympäristöä havainnollistava kuvio. Vihreän rajauksen sisällä projektiympäristö.

Asiakkaan edustajat kirjaavat havaitsemansa ongelmat tai muutospyynnöt verkkopohjaiseen projektinhallinta- ja tikettijärjestelmään YouTrackiin, josta kehittäjät poimivat tehtävät. Normaalitilanteessa korjaus- ja muutospyynnöille tehdään kehityksen ajaksi oma haara versionhallintaan, joka valmistuttuaan liitetään testihaaraan ja viedään asiakkaan testattavaksi testiympäristöön. Asiakkaan testattua ja todettua korjauksen tai uuden ominaisuuden toimivaksi voidaan se merkitä YouTrackissa valmiiksi ja kun testiympäristö sisältää vain hyväksyttyjä korjauksia tai muutoksia, voidaan testihaara liittää tuotantohaaraan ja aloittaa uuden tuotantoversion koostaminen.

3.1 Git

Versionhallintatyökaluna projektissa on hajautettuun versionhallintaan pohjautuva Git, jonka avulla kehittäjät hallitsevat lähdekoodin eri versiota kehityksen aikana. Versionhallinnassa on eri haarat tuotanto-, testi- ja kehitysversioille, joiden lähdekoodin pohjalta koostetaan testiin tai tuotantoon koontiversio TeamCityä käyttäen.

Hajautettu versionhallintajärjestelmä tarkoittaa sitä, että lähdekoodille ei ole vain yhtä keskitettyä tietovarastoa, vaan jokaisella kehittäjällä on oma, keskitettyä tietovarastoa vastaava paikallinen tietovarasto, jossa mukana seuraa lähdekoodin kaikki historiatiedot ja eri versiot. Tämä mahdollistaa muuan muassa kehittämisen ilman verkkoyhteyttä, kun riippuvuus keskitetystä säilöstä ei ole esteenä muutosten tekemiselle. Kun kehittäjä saa taas yhteyden keskitettyyn tietovarastoon, voi hän lähettää ilman verkkoyhteyttä tehdyt muutokset keskitettyyn tietovarastoon.

3.2 TeamCity

TeamCity on JetBrainsin alun perin vuonna 2006 julkaisema jatkuvan integraation palvelin kehittäjille. TeamCityn koosteympäristö muodostuu itse palvelimesta ja koontiagenteista, jotka varsinaisesti suorittavat koontiversioiden luomisen. Koontiagenttien ei tarvitse sijaita samalla palvelimella kuin itse TeamCity ja suorituskyvyn kannalta onkin suositeltavaa, että agentit ja itse TeamCity sijaitsevat eri palvelimilla. Mikäli sovelluksella on eri kohdeympäristöjä, voidaan näille jokaiselle luoma oma agentti, joka luo koontiversion vastaavalle kohdeympäristölle. Agentteja voi olla myös eri testejä varten, esimerkiksi agentti käyttöliittymätestejä ja agentti yksikkötestejä varten.

TeamCity on liitetty versionhallintaan ja aina kun kehittäjä vie uuden haaran tai päivittää vanhaa haaraa, havaitsee TeamCity tämän ja tarkistaa, osuuko muutos johonkin sen säännöistä. Mikäli jonkin säännön ehto toteutuu, lisää TeamCity työn jonoon, josta se jakaa työt vapaalle koontiagentille järjestyksessä. Koontiagentti alkaa suorittamaan saamaansa koontityötä raportoiden työn etenemisen lokitietojen ja muun koontityöstä syntyvän tiedon suoraan TeamCityn palvelimelle, josta kehittäjä voi seurata tilannetta reaaliajassa. Saadessaan koontityön valmiiksi agentti kopioi luodut koontiartefaktit eli koontiversion

tiedostot lokitietojen ohella TeamCityn palvelimelle, josta ne ovat ladattavissa ja TeamCityn käytettävissä.

3.3 YouTrack

YouTrack on JetBrainsin verkkopohjainen tiketöinti- ja projektinhallintatyökalu. Asiakas kirjaa YouTrackiin uusien ominaisuuksien toteutuspyynnöt ja mahdolliset havaitsemansa virheet kuvauksineen tiketeiksi. Kehittäjät antavat aika-arvion luotuihin tiketteihin ja tämän jälkeen asiakas päättää tehdäänkö tiketti annetulla aika-arviolla vai mahdollisesti halutaanko tiketin työhön muutoksia, jotka muuttaisivat aika-arviota.

Projektipäällikkö priorisoi asiakkaan hyväksymät työmääräarvion sisältämät tiketit asiakkaan priorisoinnin huomioiden ja kehittäjät käyvät poimimassa näistä priorisoiduista tiketeistä seuraavat työtehtävänsä saatuaan edellisen tehtävän valmiiksi.

YouTrack mahdollistaa projektipäällikölle monipuoliset työkalut projektin tilan seurantaan. Esimerkiksi projektipäällikkö voi seurata tiketteihin käytettyä aikaa suhteessa aika-arvioon ja nähdä nopealla vilkaisulla avoimet tiketit per kehittäjä työkuorman selvittämiseksi.

4 KOONTIVERSION JULKAISU EPÄONNISTUU

4.1 Ongelma

Asiakkaan ohjelmistokokonaisuuteen tehdään tavallisesti kerran viikossa torstaisin päivitys, jossa viedään mahdolliset uudet ominaisuudet ja korjaukset tuotantoon. Tuotantoon vienti tapahtuu TeamCityn kautta ajamalla ensin ”Create Production Build” -niminen koontiversion luova työ, jonka jälkeen ajetaan ”Publish Production Build” -niminen työ, joka julkaisee juuri luodun koontiversion asiakkaan tuotantopalvelimille (kuva 1).

KUVA 1. TeamCityn työjonot. Kehittäjä näkee helposti viimeisimmän koontiversion muutokset ja milloin viimeisin koontiversio on muodostettu ja julkaistu.

Koontiversion luomisessa ei yleensä ilmene virheitä, mutta koontiversion julkaisussa on tällä hetkellä usein toistuva ongelma. Ongelman vuoksi koontiversiota ensimmäisen kerran julkaistaessa prosessi keskeytyy ja tuotantopäivitystä tekevä kehittäjä joutuu ajamaan julkaisevan työn uudelleen, sillä ajamalla julkaiseva työ uudelleen koontiversion julkaisu onnistuu ilman virheitä. Huolimatta siitä, että uudelleen ajettu julkaiseva työ onnistuu, eivät kaikki järjestelmän osat, esimerkiksi toiminnanohjausjärjestelmä, osaa käynnistyä enää uudelleen automaattisesti vaan julkaisua suorittavan kehittäjän pitää käydä käynnistämässä tämä verkkosovellus manuaalisesti palvelimelta IIS-hallintapaneelin kautta.

Mikä järjestelmän osista ei välttämättä osaa käynnistyä uudelleen ei ole selvää, usein se on edellä mainittu toiminnanohjausjärjestelmä, mutta joillakin kerroilla se voi olla yksi verkkokaupan kahdesta solmusta yhdessä tai erikseen toiminnanohjausjärjestelmän kanssa. Tämä lisää inhimillisen virheen mahdollisuutta, sillä mikäli kehittäjä ei muista käydä tarkistamassa ovatko kaikki järjestelmän osat käynnistyneet uudelleen, voi esimerkiksi pahimmassa tapauksessa verkkokauppa jäädä tavoittamattomiin joillekin toimeksiantajan asiakkaan asiakkaille kesken kiivaan tarjouskilvan. Tästä seuraa reklamaatioita ja tulonmenetystä toimeksiantajan asiakkaalle ja tyytymättömiä asiakkaan asiakkaita.

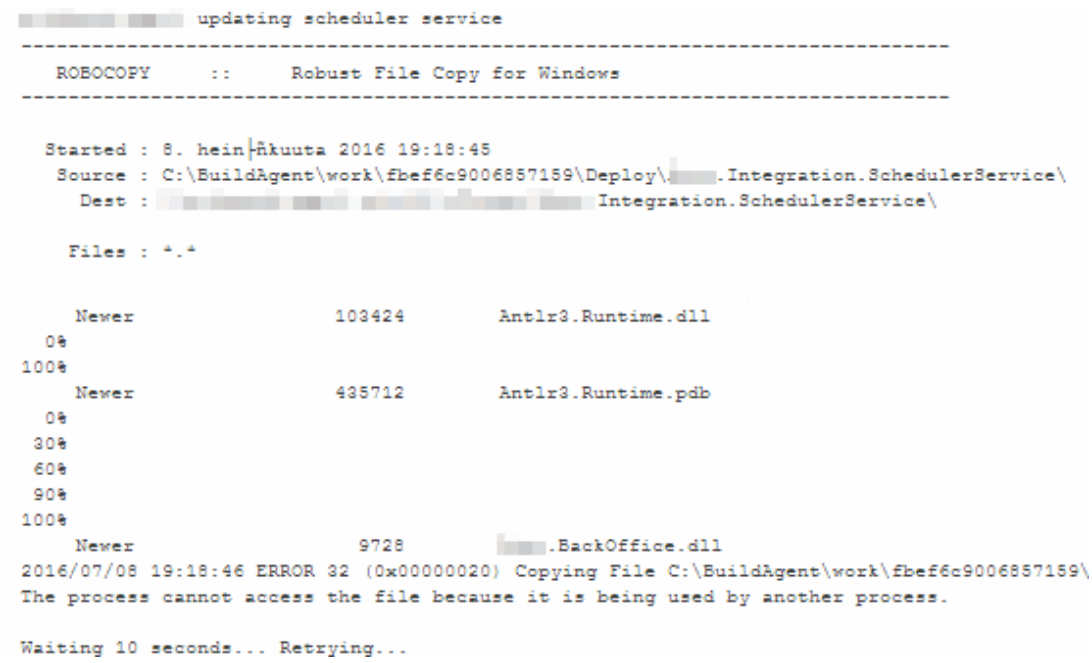
4.2 Selvitystyö

Ongelman selvitystyö alkaa TeamCityn lokitiedoista (kuva 2), jotka muodostuvat aina automaattisesti jokaisesta suoritetusta työstä riippumatta siitä, onnistuuko se vai ei. Seuraamalla virheellisistä julkaisuyrityksistä syntyneitä lokitietoja taaksepäin voidaan havaita ensimmäisen toistuvan virheen tapahtuneen 8. heinäkuuta 2016 (kuva 3).

	Results	Artifacts	Changes	Started	Duration
#147	Success	None	No changes	07 Dec 16 21:30	6m:02s
#146	Exit code 1 (new)	None	Changes (5)	07 Dec 16 21:23	7m:20s
#145	Success	None	No changes	04 Dec 16 21:13	5m:41s
#144	Exit code 1 (new)	None	Ville Talvitie (3)	04 Dec 16 21:05	7m:43s
#143	Success	None	No changes	01 Dec 16 17:16	5m:58s
#142	Exit code 1 (new)	None	Changes (32)	01 Dec 16 17:02	11m:53s
#141	Success	None	No changes	24 Nov 16 19:48	5m:36s
#140	Exit code 1 (new)	None	Changes (11)	24 Nov 16 19:40	7m:19s
#139	Success	None	No changes	18 Nov 16 18:40	5m:30s
#138	Exit code 1 (new)	None	Ville Talvitie (1)	18 Nov 16 18:33	6m:50s
#137	Success	None	No changes	17 Nov 16 20:11	5m:34s
#136	Exit code 1 (new)	None	Changes (8)	17 Nov 16 20:03	7m:23s
#135	Success	None	No changes	10 Nov 16 17:31	5m:29s
#134	Exit code 1 (new)	None	Changes (7)	10 Nov 16 17:22	7m:06s
#133	Success	None	Changes (8)	08 Nov 16 16:06	5m:14s

KUVA 2. Yleiskuva TeamCityn koontiversion julkaisun lokinäkymästä.

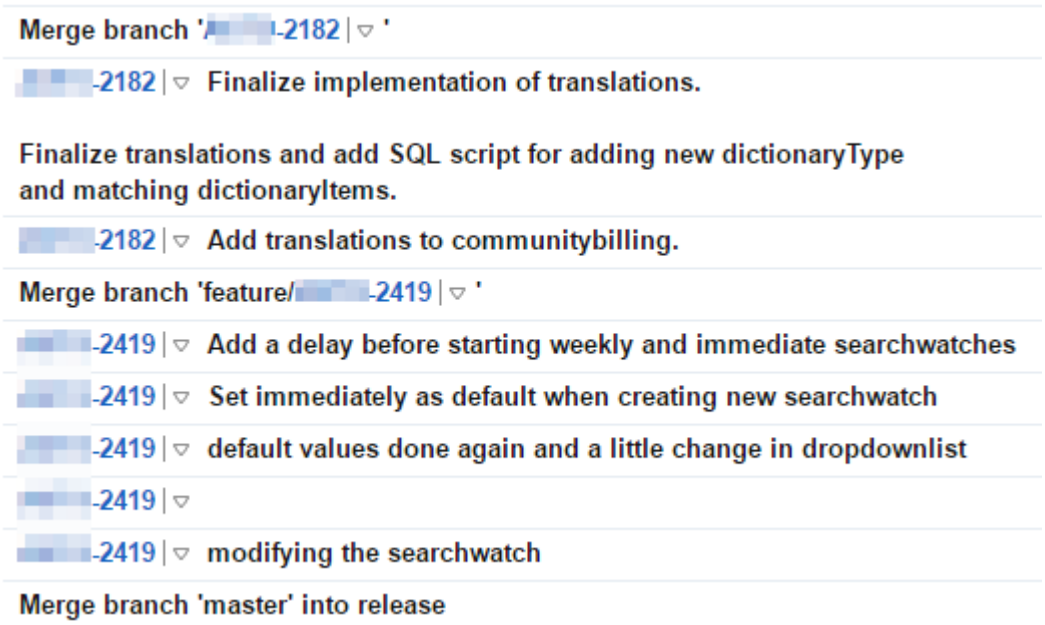
Virhe näyttää syntyvän, kun ajastettuja töitä suorittavan palvelun Schedulerservicen uutta koontiversiota ollaan kopioimassa palvelimelle ja kopiointi epäonnistuu tiedoston ollessa jonkin toisen prosessin käytössä. Julkaisutyö yrittää kopiointia uudelleen (kuva 3), mutta saavuttaessaan asetetun uusintayritysten ylärajan se jatkaa muiden päivitettyjen tiedostojen kopioimista. Jotta julkaisutyö kuitenkin suoritettaisiin loppuun asti, tulee kaikkien osavaiheiden onnistua ilman virheitä, ja koska palvelun kopiointi epäonnistuu, koko julkaisutyökin epäonnistuu.



KUVA 3. Virhetilanteen syntymäkohta lokitiedoista. Tiedoston kopiointi epäonnistuu, sillä se on yhä lukittuna toisen prosessin käyttöön.

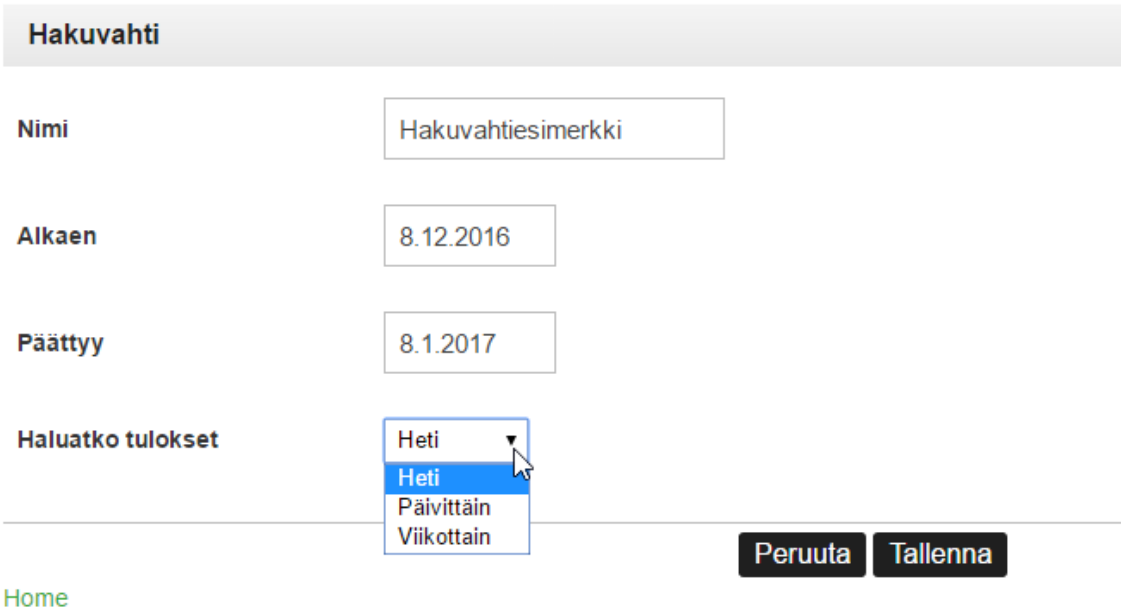
Nyt kun tiedossa on virhetilanteen ylimalkainen syntymähetki sekä virheen aiheuttava ongelma, on mahdollista katsoa oikean koontiversion luomisesta muodostuneista lokitiedoista, mitä muutoksia kyseisessä koontiversiossa on tehty ja julkaistu. Tämän jälkeen on mahdollista selvittää versionhallinnasta vielä tarkemmin kooditasolla, mitä muutoksia ajastettuja töitä ajavaan palveluun on mahdollisesti tehty.

Virheen tapahtuessa julkaisussa ensimmäisen kerran koontiversiossa ei ole kuin yksi pieni muutos, joka ei liity ajastettujen töiden palveluun, mutta edellinen koontiversio (koontiversio #198) on todella suuri kahdeksankymmenen kahden muutoksen myötä (kuva 4).



KUVA 4. Osa koontiversion #198 muutoksista. Tiketin XXXX-2419 muutokset liittyvät ajastettuja töitä suorittavaan palveluun.

Tarkastamalla koontiversion #198 lokitiedot nousee tiketti XXXX-2419 esille, sillä siinä on tehty verkkokaupan hakuvahtiin muutoksia. Verkkokaupan hakuvahti antaa toimeksiantajan asiakkaan asiakkaille mahdollisuuden luoda hakuvahteja heitä kiinnostavista tuotteista ja hakuvahtia luotaessa asiakas saa valita kuinka usein hakuvahti lähettää tiedon uusista tuotteista hakuvahdin voimassaoloaikana. Valittavissa ovat vaihtoehdot heti, päivittäin tai viikoittain (kuva 5).

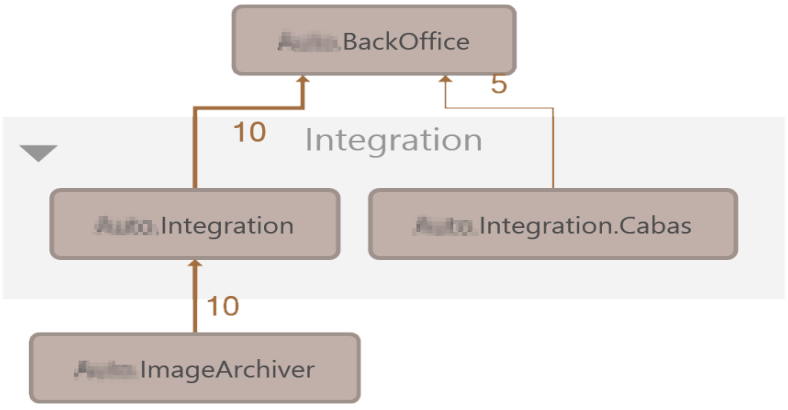


KUVA 5. Hakuvahdin luominen

Hakuvahti toimii ajastettuna palveluna, josta luodaan työ ajastuspalvelua varten ja joka ajaa kyseistä työtä konfiguraatiossa määritetyin aikavälein. Tässä kyseisessä koontiversiossa #198 tuli mahdollisuus luoda hakuvahteja, jotka lähettävät tiedon uusista tuotteista heti.

Versionhallinnasta lähdekoodin muutoksia tarkasteltaessa on huomattavissa, että hakuvahtityö osaa ja on osannut aina ottaa vastaan ja käsitellä ajastuspalvelun tarvittaessa lähettämän keskeytyspyynnön, joka vapauttaa varatut resurssit sekä tiedostot. Joudutaan toteamaan, että hakuvahti ei ole virhetilanteen aiheuttaja, vaikka aluksi vahvasti siltä vaikutti.

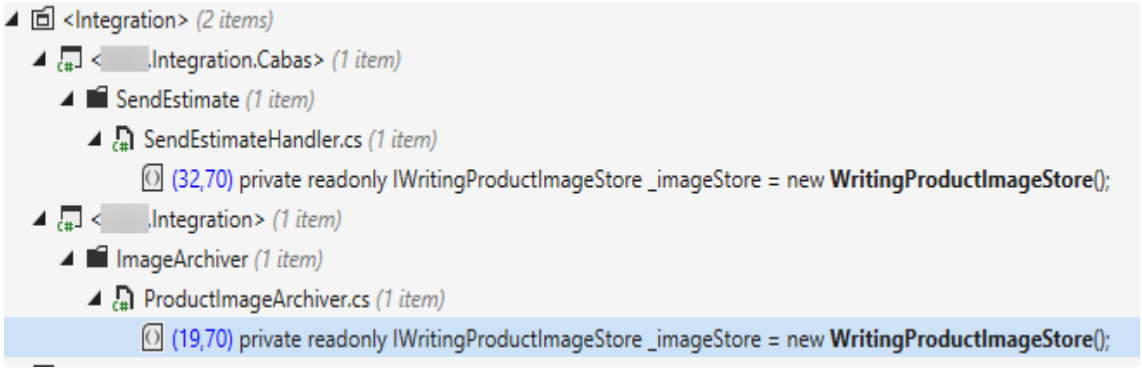
TeamCityn koontiversion julkaisun lokitiedostoja uudelleen tarkastelemalla huomio kiinnittyy kuvassa 3 näkyvään käytössä olevaan BackOffice.dll tiedostoon. TeamCityn lokitiedot eivät sisällä muita viitteitä kyseiseen tiedostoon, mutta kehitysympäristöstä viitteitä kyseiseen tiedostoon, eli kirjastoksi käännettyyn projektiin, etsittäessä viitteitä löytyy kahdesta eri järjestelmän osakokonaisuudesta (kuva 6).



KUVA 6. BackOffice-projektin viitteet projektitasolla

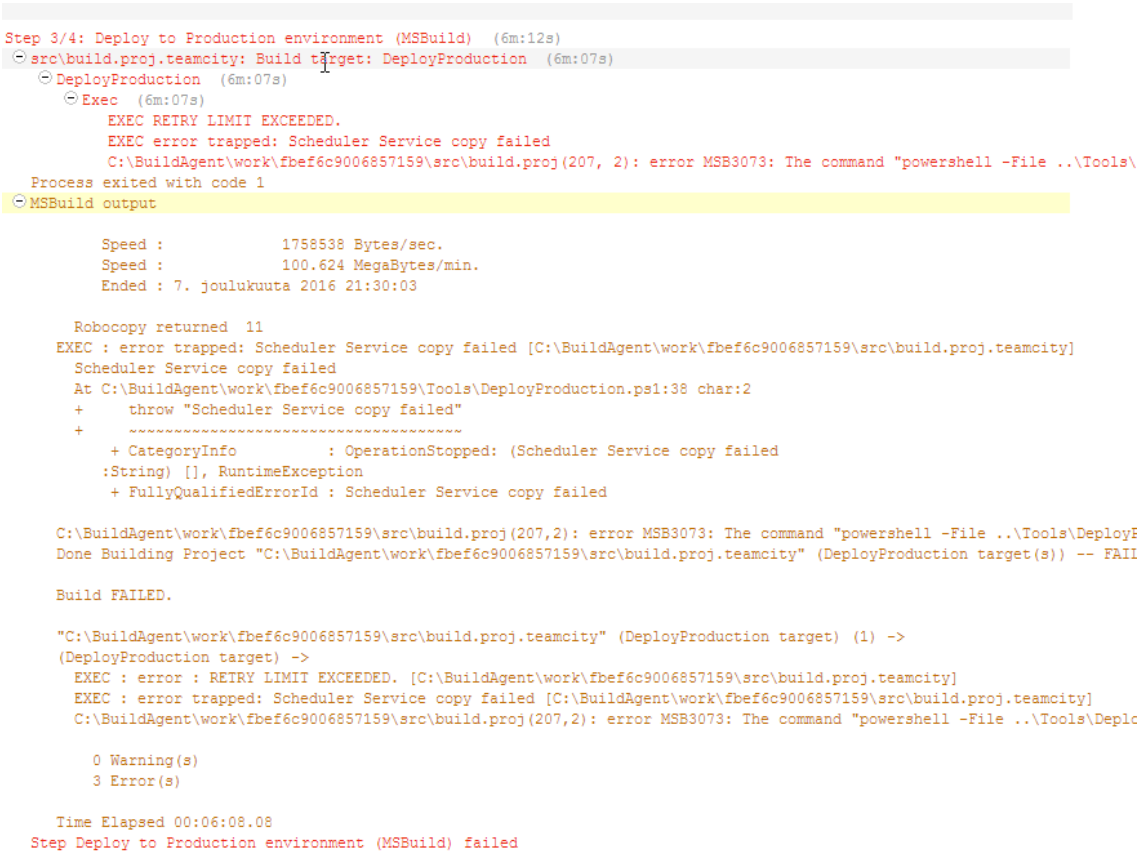
Projekti ImageArchiver arkistoi vanhoja tuotekuvia, jotta asiakkaan tuotantopalvelimen levytila ei täyttyisi tarpeettomasti ja aiheuttaisi ylimäärisiä kustannuksia. Projekti

Integration.Cabas on ajoneuvojen korjauskustannuslaskelmia eli CABAS-tiedostoja vastaanottava palvelu, joka muodostaa tai päivittää vastaanotettujen korjauskustannuslaskelmien sisältämien tietojen pohjalta tuotteita asiakkaan järjestelmään. Kumpikin projekteista viittaa luokkaan WritingProductImageStore (kuva 7), joka pitää sisällään metodit kuvien tallentamiseen, poistamiseen, arkistoimiseen ja pikkukuvien luomiseen.



KUVA 7. BackOffice -projektin viitteet lähdekooditasolla.

Kuvien arkistoija sekä korjauskustannuslaskelmia vastaanottava palvelu ovat molemmat ajastettuja palveluita, joita ajastuspalvelu ajaa viiden ja viidentoista minuutin välein vastaavasti. Kumpikaan ajastuspalvelua varten luotavista töistä ei osaa ottaa vastaan keskeytyspyyntöä, eli uutta koontioversiota julkaistessa ja ajastetun työn ollessa käynnissä on projektista käännetty kirjasto varattuna järjestelmän käyttöön, ja uuden käännetyn version kopioiminen epäonnistuu (kuva 8).



KUVA 8. Käännetyn kirjaston kopioinnissa syntyvä virheviesti.

Versionhallinnan lokitietoja tutkittaessa on huomattavissa epäkohta havaittuun virheen alkamisaikaan liittyen, sillä korjauskustannuslaskemia vastaanottava palvelu on ollut tuotantokäytössä aina tuotantoonotosta asti ja ensimmäinen merkintä kuvien arkistoijasta löytyy 16.6.2016. Kuvien arkistoijaan liittyvältä tiketiltä löytyy kuitenkin epäkohtaan selitys (kuva 9). Arkistoija on aluksi käynnistetty manuaalisesti ilman automaattista ajastusta. Arkistoijalle on vasta myöhemmin lisätty ajastus, joka on aiheuttanut virheen toistuvan ilmenemisen.

OA Asiakas 04 Jul 2016, 11:00

En pääsentota testaamaan vielä pitkään aikaan mut jos nyt tuntuisi toimivan niin en näe tossa niin kriittistä etteikö voisi viedä tuotantoon. Jos jotain ongelmaa niin kuvathan ei katoa vaan ovat jommassa kummassa päässä ja kyse vanhenmmista kuvista.

 Ville Talvitie 28 Jul 2016, 13:49

Tiedoksi: Tällä hetkellä Kehittäjä #1 arkistoi vanhojen jo luovutettujen tuotteiden kuvia tuotannon puolella pienemmissä erissä, jotta arkistoitavien kuvien määrä ei ole automaattisen arkistoinnin alkaessa liian suuri

OA Asiakas 01 Aug 2016, 10:11

Nyt näyttäisi tuotannostakin jo arkistoituvan, mutta pelkästääm ajoneuvojen kuvia. Pitäisikö varaosakuvienkin jo siirtyä?

 Kehittäjä #1 01 Aug 2016, 10:13

Ei vielä, ajattelin laittaa ajoneuvokuvat ensin ja varaosakuvat sen jälkeen.

OA Asiakas 08 Aug 2016, 14:00

Nyt katselin että ajoneuvojen kuvia siirtynyt viimeksi 4.8. Pyöriikö vielä normaalisti?

 Kehittäjä #1 08 Aug 2016, 15:49

Arkistoija oli tosiaan pysähtynyt. Laitan sen päälle taas huomenna, mutta on hyvä että se ei ole käynnissä samaan aikaan kuin osien siirto varaosahakuun.

KUVA 9. Kuvien arkistoinnin tiketillä käytyä keskustelua (muokattu).

5 KOONTIVERSION JULKAISU TOIMII JÄLLEEN

Virhetilanteen syyn ollessa nyt selvillä selvitetään ratkaisu virheeseen. Ajastettuja töitä suorittava SchedulerService on toteutettu kirjaston Quartz.NET-avulla, joka pohjautuu suosittuun avoimen lähdekoodin Java-pohjaiseen ajastuskehykseen nimeltä Quartz (Quartz.NET 2016).

Ajastuspalvelun sammuttava metodi on lähtötilanteessa tehty siten, että se sammuttaa vain käsin määriteltyjä töitä (liite 1). Sammutusmetodin rivillä 13 sijaitseva varsinainen sammutusmetodi ottaa vastaan parametrina tiedon siitä, odottaako ajastuspalvelu käynnissä olevien töiden sammumista (Quartz.Net. 2016. API dokumentaatio). Parametrin arvon ollessa epätosi ajastuspalvelu ei odota töiden sammumista, vaan sammutuskomennon ajastuspalvelua ajavalta isännältä saatuaan ajastuspalvelu sammuttaa itsensä välittömästi.

Ensimmäisenä korjaustoimena parametri muutetaan arvosta epätosi arvoon tosi, jolloin ajastuspalvelu jää odottamaan käynnissä olevien töiden sammumista ennen itsensä sammuttamista (liite 2, rivi 19.) Sammutusmetodin tapa käsitellä sammutettavat työt muutetaan geneeriseksi silmukaksi, sillä ei ole mielekästä ylläpitää listaa sammutettavista töistä itse sammutusmetodissa. Tämä vähentää inhimillisen virheen mahdollisuutta jatkossa, kun kehittäjän ei tarvitse muistaa lisätä uutta ajastettua työtä sammutettavien listalle käsin, vaan hän voi luottaa siihen, että ajastuspalvelu lähettää sammutuspyynnön automaattisesti kaikille käynnissä oleville töille.

Ajastuspalvelulla on tiedossa tällä hetkellä suorituksen alaisena olevat työt, jotka saadaan silmukkaa varten listamuotoisena kutsumalla ajastuspalvelun metodia GetCurrentlyExecutingJobs(). Silmukan sisällä voidaan nyt antaa kaikille töille keskeytyspyyntö, mutta koska välttämättä kaikki suorituksen alaisena olevat työt eivät ole keskeytettäviä töitä, lisätään silmukan sisälle tarkastelu, jolla keskeytyspyyntö annetaan vain sellaisille töille, jotka osaavat keskeytyspyynnön käsitellä (liite 2, rivit 5-11).

Ajastetuille töille tarvitsee tehdä myös muutoksia, jotta ne saadaan tukemaan keskeyttämistä. Työt periytyvät luokasta ”IJob”, joka ei tue keskeyttämistä. Korjauksena työt vaihdetaan periytymään luokasta ”AbstractInterruptableJob”, joka mahdollistaa keskeytyspyyntöjen käsittelyn (liite 3, rivi 2). Töiden suorittava metodi muutetaan metodista

”Execute” metodiin ”ExecuteJob”, joka ottaa parametreinä vastaan kontekstin lisäksi peruutussymbolin. Peruutussymboli pitää vielä saada välitettyä työn sisällä olevalle metodille, jossa varsinaisesti suoritetaan kunkin työn tehtävää (liite 3, rivi 10). Ajastuspalvelun vastaanottaessa peruutuskäskyn tieto peruutuksesta menee peruutussymbolille, joka on nyt välitetty tarvittavalle metodille. Peruutussymbolilla on ”IsCancellationRequested”-niminen ominaisuus, jota tarkkailemalla voidaan saada tieto mahdollisesta peruutuskäskystä. Tarkkailu suoritetaan ehdollisella lauseella ja mikäli ehto täyttyy, eli peruutussymbolille on tullut tieto peruutuspyynnöstä, lisätään tieto tästä lokitietoihin ja palautetaan suoritus pois lohkosta (lähdekoodinäyte 1).

```
01 public async Task HandleSendEstimates(CancellationToken token)
02 {
03
04     /*
05     Koodia
06     */
07     foreach (var id in estimateIds)
08     {
09         try
10         {
11             if (token.IsCancellationRequested)
12             {
13                 _log.Info("SendEstimateHandler cancelled by scheduler");
14                 return;
15             }
16             /*
17             Koodia
18             */
19         }
20         catch (Exception e)
21         {
22             _log.Error(string.Format("General failure in Cabas estimate
{0} processing", id), e);
23         }
24     }
25
26     _log.Info("Send Estimate Handler done");
27 }
```

LÄHDEKOODINÄYTE 1. Peruutussymbolin käsittely

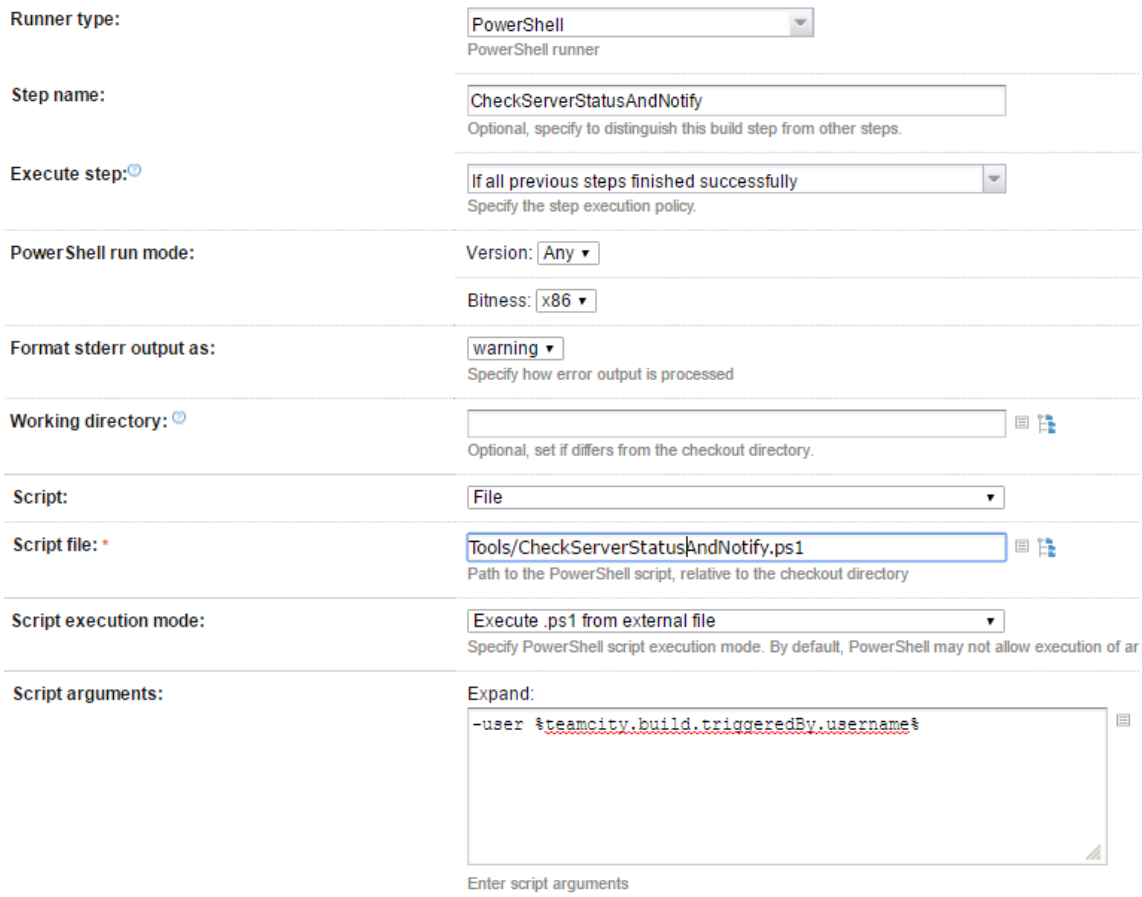
Aluksi tuki peruutuspyynnön käsittelyn lisäykselle tehtiin vain kuvien arkistointiin sekä lähdekoodinäytteessä 1 olevaan korjauskustannuslaskelmia käsitteleviin töihin. Koska lähes kaikki muutkin ajastetut työt toimivat hyvin samankaltaisella periaatteella kuin jo korjatut työt, lisäsin tuen peruutuspyyntöjen käsittelylle myös näille muille ajastetuille töille.

Ajastuspalvelun korjauksien lisäksi luotiin palvelinten tilaan tarkastava powershell-skripti (lähdekoodinäyte 2), joka osaa ottaa julkaisutyön käynnistäneen kehittäjän nimen parametrina ja lähettää julkaisutyön jälkeen sähköpostilla hänelle tiedon palvelinten tilasta.

```
01 param (
02     # User is from teamcity variable, format first.last
03     [string]$user = $( Read-Host "User:" ),
04     $PSEmailServer = "127.0.0.1",
05     $emailSuffix  = "@evolvit.fi",
06     $email = $user + $emailSuffix,
07     #Server urls for checking, correct urls emitted
08     $serverUrls = @("http://www.google.com","http://www.google.fi"),
09     $serverStatus = @()
10 )
11
12 #Loop urls and get statuscode to array
13 foreach ($url in $serverUrls) {
14   $req = [system.Net.WebRequest]::Create($url)
15   Write-Host ("Checking server {0} status" -f $url)
16   try {
17         $res = $req.GetResponse()
18   } catch [System.Net.WebException] {
19         $res = $_.Exception.Response
20   }
21   #Append status to string array for email body
22   if($res.StatusCode -eq 200) {
23     $serverStatus += ($url + ": " + "status OK")
24   } else {
25     $serverStatus += ("Check server "+ $url)
26   }
27
28 }
29 #Send email
30 Send-MailMessage -From "serverstatus@serv.er" -To $email -Subject "Server
status report" -Body ([string]$serverStatus)
```

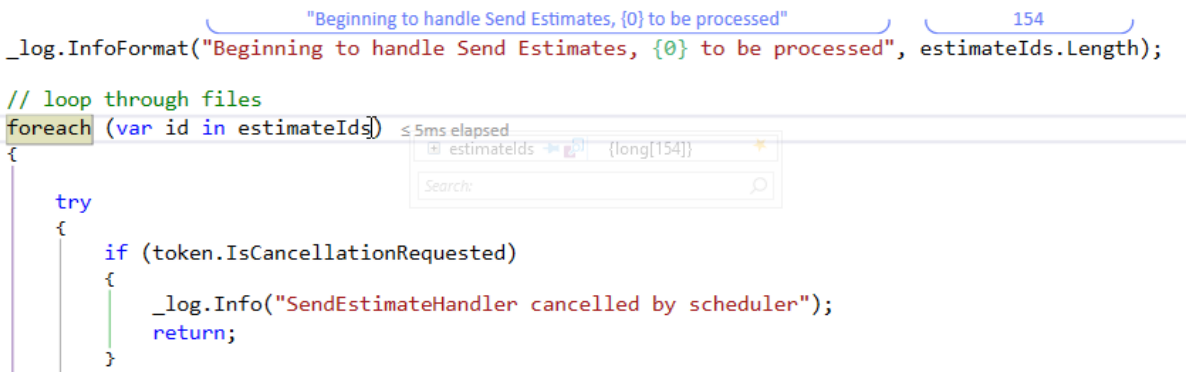
LÄHDEKOODINÄYTE 2. Palvelinten tilan tarkastava Powershell-skripti

Skriptiä varten julkaisutyön asetuksiin (kuva 10) luotiin uusi julkaisuvaihe, johon asetettiin tarpeelliset muuttujat eli tässä tapauksessa ” teamcity.build.triggeredBy.username”-niminen muuttuja, joka on koontiversion vakiomuuttuja ja löytyy TeamCityn koontiversion parametrien dokumentaatiosta (TeamCity. 2017. Koontiversion parametrit).



KUVA 10. Uusi koontiversion julkaisun vaihe palvelinten tilan tarkistusta varten

Korjaukset testattiin ensin paikallisesti kehitysympäristössä käynnistämällä ajastuspalvelu testaustilassa, jotta sen toimintaa voitiin tarkkailla ajastuspalvelun käynnistämän konsoli-ikkunan lokiviestien ja sopiviin kohtiin asetettujen pysäytyspisteiden avulla. Korjauskustannuslaskelmia sisään lukevaa työtä varten asetettiin kehitysympäristön tietokantaan jo käsitellyt korjauskustannuslaskelmat uudelleen käsiteltäviksi, jotta ajastetulle työlle saatiin luotua tehtäviä (kuva 11).



KUVA 11. Pysäytyspiste foreach-silmukan kohdalla

Kun pysäytyspisteen avulla oli varmistettu, että oltiin oikean lohkon sisällä, voitiin lähettää ajastuspalvelulle keskeytyspyyntö manuaalisesti valitsemalla ajastuspalvelun konsoli-ikkuna aktiiviseksi ikkunaksi ja lähettämällä sille keskeytyskomento näppäinyhdistelmällä ctrl-C. Keskeytyspyynnön oikean toiminnan varmistaminen tapahtuu pysäytyspisteen avulla, joka sijaitsee kuvan 11 ehdollisen lauseen sisällä. Ajastuspalvelu toimii ainakin manuaalisen keskeytyspyynnön saadessaan halutulla tavalla, joten voidaan siirtyä testaamaan korjausta tuotantoympäristöä mahdollisimman hyvin peilaavaan testiympäristöön.

Testiversion julkaisun yhteydessä virhe ei esiinny yhtä usein kuin tuotantoversion julkaisun yhteydessä, sillä testiympäristö on kevyemmällä kuormituksella ja testiympäristössä on osa ajastetuista palveluista asetettu pois käytöstä. Ennen korjauksen testausta vertailukohdan saamiseksi testiympäristön käytöstä poistetut ajastetut palvelut otettiin käyttöön. Tämän toimenpiteen jälkeen voidaan havaita testiversion julkaisevan työn lokitietoja tarkastelemalla, että virhe voidaan toistaa nyt myös testiympäristössä luotettavasti testiversion julkaisun yhteydessä. Normaalitapauksessa virheen tarkoituksenmukainen aiheuttaminen ei toki ole hyvä asia, mutta korjauksien toimivuuden takaamiseksi tämä on toivottu tila. Testiversion julkaisu suoritettiin lähes aina, kun testihaaraan oli yhdistetty jokin uusi, testiin menossa oleva ominaisuus tai korjaus. Näin saatiin lisättyä vertailukohdan luotettavuutta ja otosta suuremmaksi.

Lähdekoodimuutokset katselmoitiin projektin prosessin mukaisesti erillisessä lähdekoodin katselmointiin tarkoitetussa työkalussa ennen niiden viemistä testipuolelle testausta varten. Hyväksytyn katselmoinnin jälkeen muutokset sisältävä kehityshaara yhdistettiin testihaaraan ja suoritettiin testiversion julkaisun vaatimat toimenpiteet. Korjaushaaran sisältämän version julkaisemisen jälkeen testiversion tiheää julkaisuväliä jatkettiin, jotta saatiin luotua mahdollisimman monta virheen aiheuttaman tilanteen kaltaista tapahtumaa. Tiheän julkaisuvälin lisäksi korjauksen toimivuuden varmistamiseksi luotiin aikaisemmin todetuille, virheen varmasti aiheuttaneille, ajastetuille töille lisätyötä. Kuvien arkistoivaa työtä kuormitettiin asettamalla suuri joukko jo arkistoituja kuvia uudelleen arkistoitavaksi ja korjauskustannuslaskemia sisään lukevaa työtä varten muutettiin jo käsiteltyjä korjauskustannuslaskelmia käsittelemättömiksi, aivan kuten paikallisessa kehitysympäristössä testatessa. Näin voitiin keinotekoisesti luoda olosuhteet, jossa todennäköisyys virheen esiintymiselle on mahdollisimman suuri.

Testiversion julkaiseva työ ei korjauksen sisältäneen version viennin jälkeen epäonnistunut enää kertaakaan. Testiympäristön lokitiedoista oli nähtävissä sammutuspyynnöt ja onnistuneet, hallitut ajastuspalvelun sammutukset. Korjaus katsottiin valmiiksi tuotantoon vietäväksi, jotta korjaus voitaisiin todeta oikeasti toimivaksi.

Korjauksen sisältävä haara yhdistettiin tuotantohaaraan, jotta seuraavan tuotantoversion viennin yhteydessä korjaus saadaan vietyä tuotantopalvelimille. Tuotantoympäristössä korjausta varten ei ole tarvetta luoda keinotekoisesti kuormitusta, mutta testausta rajoittavana tekijänä on rajattu aikaikkuna tuotantoversion viennille ja tämän vuoksi korjauksen testaaminen ei ollut yhtä suoraviivaista ja nopeaa kuin testiympäristössä testaaminen. Korjauksen sisältäneen version jälkeisissä tuotantoversion julkaisutöissä ei ole myöskään tuotannossa esiintynyt enää virhettä.

6 POHDINTA

Virhetilanne on saatava ratkaistua, jotta säästyy aikaa ja sitä kautta rahaa. Toistuvasti ilmenevä tavallaan harmiton virhe voi myös turruttaa kehittäjät virheisiin ja aiheuttaa oikean virheen sattuessa kohdalle vaarallisen turvallisuuden tunteen, että virhe voidaan ohittaa selvittämättä sen syytä ja julkaisutyötä suorittava kehittäjä luulee virheellisesti voivansa jatkaa julkaisutyötä. Tulevaisuudessa on mahdollista, että uudet kehittäjät ovat viemässä versiota ja virhetilanteet tuotantoversion julkaisussa ovat aina stressaavia, joten korjaamalla virhetilanne voidaan vähentää turhaa stressiä.

Virheen selvitystyö oli lähtökohdiltaan tavallisesta poikkeavaa, sillä yleensä selvitystyö alkaa sovelluksesta syntyvästä virheestä, josta joko asiakas on ilmoittanut tai kehittäjä on havainnut sovelluksen lokitiedoista ja on tiedossa edes summittainen kohta lähdekoodissa, jossa virhe esiintyy. Tässä tapauksessa virheen aiheuttavaa kohtaa jouduttiin selvittämään suhteessa lähdekoodiin kaukaa, mikä oli haastavaa, mutta samalla hyvin mielenkiintoista. Korjausten testaamisen haasteet ei-tuotantoympäristössä herätti huomaamaan tarpeen kuormitustestaukselle, sillä testiympäristön tai kehitysympäristöjen normaali käyttökuorma ei riitä vastaamaan tuotantoympäristön kokemaa kuormitusta, jonka alaisena virheen mahdollisuus esiintyä syntyi.

Voidaan todeta, että asetetut tavoitteet korjata virhe ja parantaa versiopäivityksen sujuvuutta saavutettiin. Jatkoselvityksen alaiseksi jää sopivan kuormitustestaustyökalun etsiminen ja selvitys, kuinka se voitaisiin liittää osaksi jatkuvan integraation ympäristöä. Tämä opinnäytetyössä käsitelty projekti ei mitä todennäköisimmin ole ainoa toimeksiantajan projekti, joka hyötyisi tässä työssä tehdyistä havainnoista ja saavutetuista kokemuksista. Myös työn aikana luotua palvelinten tilan tarkistavaa skriptiä voi olla mahdollista hyödyntää muissa toimeksiantajan projekteissa.

Syventyminen jatkuvaan integraatioon ja jatkuvaan toimitukseen sai minut kiinnostumaan sovellusten elinkaaresta aina projektin perustamisesta julkaisuun asti, ja huomaamaan toimivan palauteketjun tärkeyden kehittäjille.

LÄHTEET

Evolvit Oy. 2016. Yritys. Luettu 3.11.2016. https://evolvit.fi/

Fowler, M. 2006. Continuous Integration. Luettu 21.05.2017
https://www.martinfowler.com/articles/continuousIntegration.html

Guckenheimer, S. 2017. Continuous Integration. Luettu 10.11.2017
https://www.visualstudio.com/learn/what-is-continuous-integration/

Huotarinen, J. 2016. Tiesitkö jatkuvan julkaisun olevan jo arkipäivää?. Luettu 26.03.2018
https://gofore.com/tiesitko-jatkuvan-julkaisun-olevan-jo-arkipaivaa/

Kauppalehti. 2016. Yrityshaku. Luettu 3.11.2016.
http://www.kauppalehti.fi/yritykset/yritys/evolvit+oy/21217890

Microsoft. 2016. PowerShell.
https://msdn.microsoft.com/en-us/powershell/mt173057.aspx

Quartz.NET. 2016.
http://www.quartz-scheduler.net/

Quartz.Net. 2016. API dokumentaatio.
http://quartznet.sourceforge.net/apidoc/2.0/html/

Richardson, J. 2006. Continuous Integration: The Cornerstone of a Great Shop. Luettu 10.11.2016.
http://www.methodsandtools.com/archive/archive.php?id=42

TeamCity. 2016. Continuous Integration with TeamCity. Luettu 27.03.2018.
https://confluence.jetbrains.com/display/TCD10/Continuous+Integration+with+TeamCity

TeamCity. 2017. Koontiversion parametrit.
https://confluence.jetbrains.com/display/TCD10/Predefined+Build+Parameters

LIITTEET

Liite 1. Ajastuspalvelun sammutuksen lähdekoodi ennen korjauksia

```
01 public bool Stop(HostControl hostControl)
02 {
03  try
04  {
05      _scheduler.Interrupt(new JobKey("searchWatchProcessingJob",
AutoJobsIden-tity));
06  }
07  catch (Exception ex)
08  {
09      _log.Warn("Interruption of jobs failed", ex);
10  }
11  try
12  {
13      _scheduler.Shutdown(false);
14  }
15  catch (Exception e)
16  {
17      _log.Error("Stop failed.", e);
18      return false;
19  }
20
21  return true;
22 }
```

Liite 2. Ajastuspalvelun sammutuksen lähdekoodi korjausten jälkeen

```
01 public bool Stop(HostControl hostControl)
02 {
03     try
04     {
05         foreach (var executingJob in
_scheduler.GetCurrentlyExecutingJobs())
06         {
07             if(executingJob.JobInstance is IInterruptableJob)
08             {
09                 _scheduler.Interrupt(executingJob.JobDetail.Key);
10             }
11         }
12      }
13      catch (Exception ex)
14      {
15          _log.Warn("Interruption of jobs failed", ex);
16      }
17      try
18      {
19          _scheduler.Shutdown(true);
20      }
21      catch (Exception e)
22      {
23          _log.Error("Stop failed.", e);
24          return false;
25      }
26      return true;
27 }
```

Liite 3. Korjauskustannuslaskelmia vastaanottavan työn lähdekoodi ennen ja jälkeen korjauksien

```
01 [DisallowConcurrentExecution]
02 public class CabasHandlerJob : IJob
03 {
04     public void Execute(IJobExecutionContext context)
05     {
06         try
07         {
08             SendEstimateHandler handler = new SendEstimateHandler();
09
10             handler.HandleSendEstimates().Wait();
11         }
12         catch (Exception e)
13         {
14            LogManager.GetLogger(GetType()).Error("SendEstimate handling
failed", e);
15         }
16     }
17 }
```

Ennen korjauksia

```
01 [DisallowConcurrentExecution]
02 public class CabasHandlerJob : AbstractInterruptableJob
03 {
04     protected override void ExecuteJob(IJobExecutionContext context,
CancellationToken token)
05     {
06         try
07         {
08             SendEstimateHandler handler = new SendEstimateHandler();
09
10             handler.HandleSendEstimates(token).Wait();
11         }
12         catch (Exception e)
13         {
14             if (e is OperationCanceledException)
15             {
16                 LogManager.GetLogger(GetType()).Info("SendEstimate
handling cancelled", e);
17             }
18             else
19             {
20                 LogManager.GetLogger(GetType()).Error("SendEstimate
handling failed", e);
21             }
22         }
23     }
24 }
```

Korjausten jälkeen