Jarno Salonen

Samanaikaisen työskentelyn ristiriitojen ennustaminen ja ratkaiseminen ohjelmistokehityksessä

Tietotekniikan kandidaatintutkielma

30. huhtikuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Jarno Salonen

Yhteystiedot: jarno.a.salonen@student.jyu.fi

Ohjaaja: Timo Tiihonen

Työn nimi: Samanaikaisen työskentelyn ristiriitojen ennustaminen ja ratkaiseminen ohjelmistokehityksessä

Title in English: Predicting and Resolving Merge Conflicts in Collaborative Software Development

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 19+0

Tiivistelmä: Samanaikaisessa ohjelmointityöskentelyssä kehittäjät muuttavat usein koodin osia olematta täysin tietoisia muiden tekemistä muutoksista. Vaikka tämä lisää työn tehoa, voi tällaisista muutoksista seurata ristiriitoja kun koodia tuodaan yhteen. Ymmärtääksemme paremmin näitä ristiriitoja tarkastelemme neljää niiden välttämiseen ja ratkaisemiseen kehitettyä työkalua. Tehokas ristiriitaisuuden ratkaisukeino näyttäisi olevan niiden välttäminen kokonaan, toisaalta taas pienien tarkkaan kohdistettujen muutoksien historian seuranta tuottaa tuloksia. Yleisesti yhdistämisen apuvälineistä on suuresti hyötyä. Ristiriitojen moninaisuuden takia yhtä parasta ratkaisua tuskin on.

Avainsanat: git, yhdistämisen ristiriita, versionhallinta, scm

Abstract: In collaborative software development, developers often change code without fully being aware of changes made by other people. While this increases productivity, it might result in conflicts when merging code. To better understand these conflicts, we study four tools developed to resolve and avoid them. Effective merge conflict resolution appears to be to completely dodge them, on the other hand having access to fine-grained change history is beneficial. In general using merge conflict tools lowers the risk of conflict. The spectrum of merge conflicts is quite wide so it is unlikely there is one ultimate answer or tool.

Keywords: git, merge conflict, version control, scm

Termiluettelo

Git	ilmainen ja avoimeen lähdekoodiin perustuva hajautettu versionhallintajärjestelmä
merge conflict	kahden koodin yhdistämisessä tapahtunut ristiriita
VCS	versionhallintajärjestelmä (version control system), jolla pidetään yllä historiaa ohjelmiston versioista
SCM	ohjelmiston konfiguraationhallinta (software configuration management), ohjelmistotiedostoihin tehtyjen muutoksien seuranta ja hallinta

Kuviot

Sisällys

1 Johdanto

Nykyaikaisessa ohjelmistokehityksessä versionhallinta näyttelee suurta osaa ja on monen projektin kulmakivi. Suuri hyöty tuo mukanaan kuitenkin myös haasteita. Useamman ohjelmistokehittäjän samanaikainen työskentely väistämättä johtaa tilanteisiin, joissa koodin yhteen tuominen ei ole yksiselitteistä. Näiden ongelmien ratkaisuun on hyvä varautua jo etukäteen.

Tämän tutkielman tarkoitus on tarkastella neljää koodin ristiriitaisuuksien löytämiseen ja ratkaisemiseen kehitettyä työkalua ja selvittää niiden siinä käyttämiä keinoja ja strategioita. Tutkimus toteutetaan kirjallisuuskatsauksena ja sen luonteesta johtuen tutkielma ei sisällä parannus- tai kehitysehdotuksia, vaan antamaan mahdollisia lähtökohtia niiden kehittämiselle jälkeenpäin.

Tutkielma aloitetaan pohjustuksella versionhallintaan ja sen historiaan. Tarkennetaan mikä versionhallinta oikein on ja mistä se on saanut alkunsa. Seuraava luku menee syvemmälle ristiriitaisuuksiin ja niiden syihin. Viidennessä luvussa päästään itse järjestelmiin, käydään ne yksitellen, mutta pintapuolisesti läpi johtuen niiden teknillisestä luonteesta. Lopuksi viimeisessä luvussa pohditaan ja analysoidaan läpikäytyä ja että onko työkaluilla yhtäläisyyksiä.

2 Versionhallinnan historiaa

Bell Laboratorioiden vuonna 1972 kehittämä Source Code Control System (SCCS) oli ensimmäinen varsinainen versionhallintajärjestelmäksi luokiteltu ohjelmisto. Sitä käytettiin esimerkiksi useissa UNIX jakeluversioissa standardina käskyjoukkona. (Ruparelia 2010)

SCCS:n syrjäytti vasta 80-luvulla Walter F. Tichyn luoma Revision Control System. Se oli paljon edeltäjäänsä kehittyneempi sisältäen esimerkiksi automaattisen varastoinnin, haun ja lokitietojen pitämisen. Tiedostojen versioiden väliset muutokset tallennettiin tietokantaan. Muutoksien ajo mahdollisti edellisiin versioihin palaamisen. Sen huonoina puolina oli mm. rajoittuminen yhteen tiedostoon, eikä se tukenut kansiorakenteita. Ehkä kuitenkin vielä merkittävämpää oli samanaikaisen työskentelyn eri lokaatioissa mahdollistavan ominaisuuden puuttuminen. (Ruparelia 2010)

Tähän puutteeseen tuli ratkaisu vuonna 1985 Concurrent Versioning System:n (CSV) muodossa. Dick Grune kirjoitti sen omiin tarpeisiinsa opiskeluaikanaan (Ruparelia 2010). Se oli ominaisuuksiltaan enemmän nykyaikaisempi versionhallintajärjestelmä. Esimerkiksi nykyisistä järjestelmistä Subversion on paljon velkaa sille. Tarkastellaan seuraavaksi modernia versionhallintajärjestelmää yleisesti.

3 Versionhallintajärjestelmä

Versionhallinnan tarkoitus on pohjimmiltaan ylläpitää projektien kehityshistoriaa. Käytännössä tämä tarkoittaa käyttäjän valitsemia tallennuskohtia, jotka jäävät muistiin historiaan ja joita voidaan katsella tai palauttaa tarpeen vaatiessa (Sink 2011). Samalla koko projektin muutoshistoria varmuuskopioidaan ja mahdollistetaan myös monen kehittäjän samanaikainen työskentely. Projektia voi työstää useassa haarassa ja yhdistellä näitä tarpeen ja tilanteen vaatiessa.

Historiaa ylläpidetään tekemällä committeja tasaisin väliajoin, ne ovat kuin jonossa olevia tallennuksia. Nämä commitit muodostavat haaran. Yleensä projektilla on yksi päähaara, josta otetaan kopio työhaaraksi vaikka jonkin ominaisuuden kehittämiseksi. Näitä työhaaroja voi olla useita samaan aikaan. Kun ominaisuus on valmis voidaan työhaara yhdistää päähaaraan ja näin tuoda ominaisuuden muutokset osaksi projektia (Sink 2011).

3.1 Versionhallinnan tyypit

Versionhallintajärjestelmiä on kahdenlaisia, keskitettyjä ja hajautettuja. Näillä on joitakin hyvin perustavanlaatuisia eroja.

3.1.1 Hajautettu

Hajautettu versionhallintajärjestelmä on nimensämukaisesti hajautettu eli jokaisella projektiin osallistujalla on oma kopionsa tietovarastosta. Käyttäjien tekemät muutokset sitten viedään keskitettyyn etätietovarastoon ja yhdistetään mahdollisiin muiden tekemiin muutoksiin. Hajautuksen suurin valtti on työskentelyn mahdollistaminen ilman yhteyttä etätietovarastoon. Keskitetyn palvelimen vikaantuessa ei koko historiaa välttämättä menetetä, koska jokaisella käyttäjällä on koko historia aina paikallisesti omalla koneellaan (Somasundaram 2013).

Git on esimerkki hajautetusta versionhallinnasta ja se onkin tämän hetken suosituin hajautettu versionhallintajärjestelmä valtavirrassa. Sen suosiota osaltaan selittää sen maksuttomuus

ja pääsy lähdekoodiin. Gitiä voi käyttää pelkästään paikallisesti omalla koneellaan tai sitten synkronoida etätietovaraston kanssa. Lukuisat palvelut tarjoavat ratkaisuja etätietovarastojen ylläpitoon, esimerkiksi GitHub ja GitLab (Chacon ja Straub 2014).

3.1.2 Keskitetty

Versionhallintajärjestelmä, jossa tietokantapalvelin on keskitetysti yhdessä paikassa, kutsutaan keskitetyksi versionhallintajärjestelmäksi. Käyttäjät kommunikoivat tietokannan kanssa ladaten tiedostoja ja tekemällä niihin muutoksia. Nämä muutokset näkyvät tietokannan muille käyttäjille ja näin systeemi tukee samanaikaista työskentelyä. Ylläpito myös helpottuu tietokannan ollessa yhdellä serverillä ja yhdessä paikassa.

Keskitetyn versionhallintajärjestelmän suurin kompastuskivi on kuitenkin juuri sen olennaisin ominaisuus eli sijainti yhdessä paikassa. Vikatilanteen sattuessa kaikkien projektin jäsenten yhteys tietokantaan ja sitä kautta versionhallintaan katkeaa. Myös varmuuskopioiden tekeminen on keskitetyssä järjestelmässä erityisen tärkeää. Palvelimen rikkoutuessa ja datan korruptoituessa koko versiohistoria menetetään (Sink 2011).

Yksi suosituimmista ellei jopa suosituin keskitetty versionhallintajärjestelmä on edellä mainittu Subversion. Subversinonille tyypillistä on monen projektin pitäminen samassa tietokannassa tai tietovarastossa (Pilato, Collins-Sussman ja Fitzpatrick 2008).

3.2 Yhdistämisen ristiriita

Versionhallintana liittyy vahvasti myös ristiriidat. Kahden tai useamman henkilön samanaikaisessa työskentelyssä käyttäjät usein luovat tai muuttavat tiedostoja ja nämä muutokset eivät ole aina reaaliajassa ilmeisiä muille käyttäjille. Näiden muutosten yhdistäminen esimerkiksi yhteiseen keskitettyyn etätietovarastoon voi aiheuttaa ristiriitaisuuksia (merge conflict). Toisinaan taas ne tulevat ilmi kääntäjänvirheestä (Sarma, Redmiles ja Hoek 2012). Käytännössä ristiriitaisuus syntyy kun käytettävä versionhallintajärjestelmä ei pysty itse yhdistämään kahta erillistä versiota projektista tai tiedostosta. Tällöin yhdistämisen toimivuus jää käyttäjän käsin tehtäväksi. Ristiriitaisuudet ovat yleisiä, ja niitä löytyy suuria määriä teollisista ja avoimista koodiympäristöistä (Perry, Siy ja Votta 2001).

Ristiriitaisuudet ymmärrettävästi vaikuttavat ohjelmistokehitykseen negatiivisesti. Välttääkseen tätä käsin tehtävää yhdistämisen ratkaisemista kehittäjät ovat turvautuneet riskialttiisiin ratkaisuihin, esimerkiksi kiiruhtamaan ollakseen ensimmäinen muutosten tekijä tai tekemällä puolittaisia tarkastuksia (Grinter 1996) (Sarma, Redmiles ja Hoek 2012).

4 Ristiriitaisuudet

Kuten edellisen kappaleen lopussa mainittiin ristiriitaisuudet koodin yhdistämisessä syntyvät, kun versionhallinta ei osaa itse yhdistää kahta eri versiota (Ghiotto ym. 2018). Jos esimerkiksi toinen kehittäjä on lisännyt työhaaraan uutta koodia sen commitin jälkeen, josta aloitettiin, ei koodin yhdistäminen automaattisesti onnistu. Näillä versioilla on eri edeltäjät, jotenka versionhallinta ei pysty päätöstä automaattisesti tekemään tekemään (Da Silva, Borba ja Pires 2022).

4.1 Syitä ristiriidoille

Ristiriidat koodien eri versioiden välillä voivat johtua monesta syystä. Silva, Borba ja Pires tarkastelussaan (Da Silva, Borba ja Pires 2022) listaavat 451 avoimen lähdekoodin java-projektista löytynyttä 239 ristiriidan syitä:

65 prosenttia löydetyistä ristiriidoista (112 osumaa) koski viittauksia puuttuviin luokkiin, metodeihin, muuttujiin tai luokkiin. Näitä ristiriitoja on siis ylivoimaisesti eniten (Da Silva, Borba ja Pires 2022).

Suunnittelemattomat riippuvuudet aiheuttava myös ristiriitoja. Nämä johtuvat usein siitä, kun toinen kehittäjä lisää metodin rajapintaan ja toinen kehittäjä lisää olemassa olevaan luokkaan "implements-käskyn viittaamaan kyseiseen rajapintaa. Koodi ei silloin käänny, koska luokka ei alusta rajapinnassa olevaa metodia. Näitä ristiriitoja löytyi kuitenkin vain 5 prosenttia (12 kpl) (Da Silva, Borba ja Pires 2022).

Niin sanotusta copy/paste -efektistä johtuvia ristiriitoja, eli samalla nimellä varustettuja metodeja samassa luokassa tai samalla nimellä varustettuja muuttujia samassa metodissa. Näitä oli kuitenkin marginaalisesti.

Myös metodien palautusten tyypit voivat aiheuttaa harmia. Nämä vaativat enemmän huomiota, koska kehittäjän odotetaan tutustuvan ensin metodeihin joiden parissa työskentelee (Da Silva, Borba ja Pires 2022).

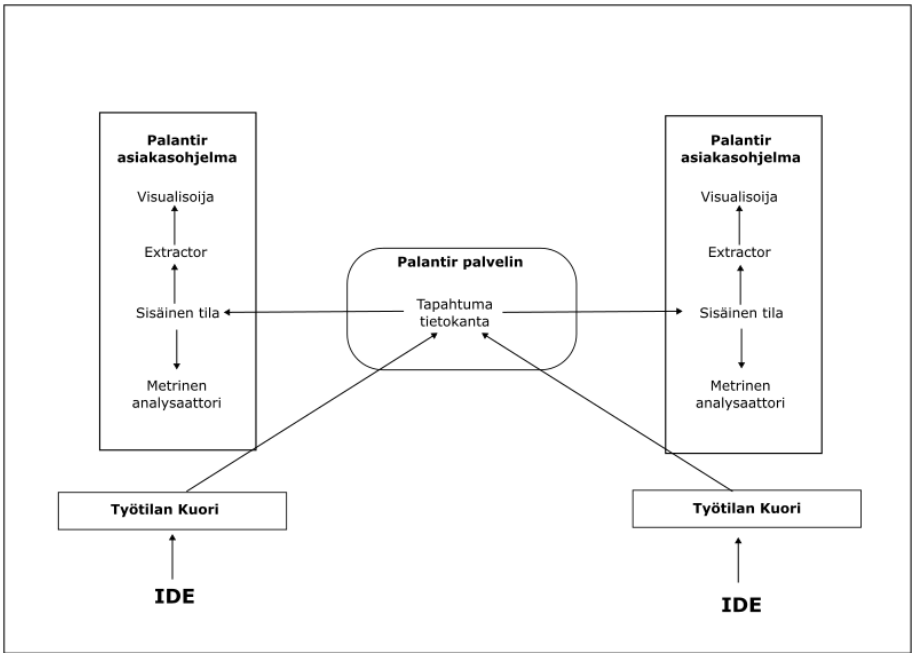
Ei koodin kääntämiseen liittyvät ristiriidat ovat mahdollisia ja voivat liittyä vaikka lisenssin otsikkoon (Da Silva, Borba ja Pires 2022).

5 Tarkasteltavat järjestelmät

Ristiriitaisuuksien tunnistamiseen on luotu useita työkaluja. Tarkastellaan tässä niistä neljää.

5.1 Palantir

Palantir on SCM:n päällä toimiva huomiotyökalu. Se on kirjoitettu Javalla ja on saatavilla lisäosana Eclipse IDE:lle. Työkalun toiminta perustuu ristiriitaisuusinformaation välittämiseen kunkin koodia työstävään tahon työskentelytilaan reaaliaikaisesti. Ristiriidat välitetään visuaalisesti suoraan IDE:n käyttöliittymään. (Sarma, Redmiles ja Hoek 2012)



Kuvio 1. Palantirin arkkitehtuuri

Palantir käsittelee suoria ja epäsuoria ristiriitaisuuksia eri tavoin. Suorat eli samaa tiedostoa koskevat ristiriidat Palantir löytää valvomalla muutoksia tiedostoon ja jakamalla informaatiota jatkuvasti muiden tiedostoa muokkaavien tahojen kanssa. Epäsuorissa ristiriitaisuuksissa tarvitaan ylimääräinen askel. Tässä vaiheessa Palantir lähettää tiedon poikkeavuuksista (diff) luokan julkisista muuttujista ja metodeista joita muut työtilat voivat hyödyntää löytämään epäsuorat ristiriidat. (Sarma, Redmiles ja Hoek 2012)

Palantir koostuu seuraavista komponenteista: Työtilan kuori (Workspace Wrapped), joka seuraa SCM:ään ja työskentelyeditoriin liittyviä tapahtumia, Sisäinen tila (Internal State) tallettaa sisäiseen välimuistiin tapahtumia joita käytetään potentiaalisten ristiriitojen löytämiseen ja vakavuuden määrittelyyn, Metrinen analysaattori (Metric Analyzer) analysoi työtiloja ristiin löytääkseen epäsuoria ristiriitoja, Extractor, joka suodattaa ja formatoi Sisäisen tilan tapahtumia ja Visualisoija (Visualization), jonka tehtävä on tuoda ristiriitainformaatio käyttäjälle IDE:hen. Lopuksi keskitetty palvelin, joka pitää kirjaa kaikista työtilojen tapahtumista. (Sarma, Redmiles ja Hoek 2012)

Sarma, Remiles ja Hoek (Sarma, Redmiles ja Hoek 2012) kokeissaan toteavat Palantirin käytön johtavan aikaisempaan ristiriitojen tunnistamiseen ja ratkaisuun suuressa määrässä tapauksia, jonka lisäksi he raportoivat muutoksia käyttäjien suhtautumisessa liittyen ristiriitojen ratkaisustrategiaan.

5.2 Program Synthesis -metodi

Program Synthesis -metodin käyttö perustuu havaittujen ristiriitaisuuksien luokitteluun tiedostotyypin, ristiriidan sijainnin tiedostossa ja ristiriidan koon mukaan. Tämän jälkeen käytetään täsmäkieltä (DSL, domain-specific language), joka on tietyllä sovellusalustalle erikoistunut ohjelmointikieli (Van Deursen ja Klint 2002). Tällä ohjelmointikielellä pyritään löytämään toistuvat ristiriitojen ratkaisumallit ja oppimaan strategioita ratkaisuihin (Pan ym. 2021).

DSL:n avainidea on kuvata ristiriitojen ratkaisut tietovaraston pää- ja sivuhaarojen osien suojattuna ketjuna. Nämä osat on rakennettu käyttäen joitakin järjestettyjä joukko-operaattoreita, kuten numerointia, suodatusta ja vähennystä (Pan ym. 2021).

Rangeet Pan tiimeineen (Pan ym. 2021) soveltaa täsmäkieltä Chromium -päähaaran Microsoft Edge -haaraan. Opittuja ratkaisumalleja hyväksikäyttäen DSL ratkaisi 41% havaituista 1-2 koodirivin ristiriidoista 93,2% tarkkuudella (11,4% kaikista havaituista ristiriidoista).

5.3 MergeBERT

MergeBERT on ohjelmanyhdistämisneurokehys, joka perustuu merkkitason kolmisuuntaiseen erotteluun. Hyödyntämällä yhdistämisen ristiriitojen ratkaisujen rajoittunutta luonnetta, MergeBert uudelleenmuodostaa ratkaisusekvenssitehtävän luokittelutehtäväksi primitiivisille yhdistymiskuvioille, jotka on poimittu todellisista ristiriidoista. (Svyatkovskiy ym. 2022)

Kehyksen malli on tekstuaalinen ja sen ytimessä on ominaisuus hajottaa ohjelman tekstiä syöte-esitysmuotoon, joka pystytään opettamaan muuntoenkooderille. Tätä mallia soveltamalla Svyatkovskiy kumppaneineen (Svyatkovskiy ym. 2022) saavuttaa 63-68 ratkaisuprosentin.

5.4 Hienojakoinen koodin muutoshistoria

Nishimura Yuichi ja Maruyama Katsuhisa esittävät julkaisemassaan artikkelissa ristiriitaisuuksien ratkaisemiseen MergeHelper-työkalua (Nishimura ja Maruyama 2016). MergeHelperin lähtökohtana on, että ohjelmoijien on hyvä tietää ristiriidan aiheuttavan koodiosion lisäksi miten kyseinen osio on muuttunut eli sen muutoshistoria. MergeHelperin sisältämä ChangeTracker tallentaa automaattisesti käyttäjän editoriin tallentamat muutokset ja commitoinnin yhteydessä siirtää ne tietovarastoon.

Havaitakseen ristiriidat MergeHelper käyttää apunaan muutoksien historian "viipalointia", joka poimii ne muokkausoperaatiot, jotka vaikuttavat tietyn luokan jäsenen historiaan. Se tunnistaa ristiriidassa olevat luokan jäsenet kahden version väliltä ja sallii käyttäjän tarkastella vain muokkausoperaatioita.

6 Johtopäätökset

Ristiriitaisuuksien moninaisuuden takia voidaan sanoa, että ei ole sitä yhtä parasta tapaa ratkaista ongelmaa. Tapauskohtaisesti valittava työkalu takaa varmasti aina parhaimman lopputuloksen. Esimerkiksi edellisessä luvussa käsitellyistä työkaluista vain Palantir ja MergeHelper tukee Eclipsen kanssa työskentelyä.

Käsitellyt työkalut ovat kuitenkin hyvin teknillisiä ja erilaisia. On haastavaa löytää yhtäläisyyksiä niiden toimintatapojen väliltä, koska ne luottavat hyvin erilaisiin strategioihin. Palantir on enemmän toteutuksen aikainen ja luottaa reaaliaikaisen informaation välittämiseen sekä voi toimia jopa ennaltaehkäisevästi, miltein ennustavasti. MergeBERT taas hyödyntää neurokehystä ja koneoppimista ristiriitojen löytämiseen ja analysointiin. Käyttäjäystävällisimpiä ovat varmasti edellä mainitut Palantir ja MergeHelper, jotka ovat suoraan liitettävissä IDE:n yhteyteen.

Työkalujen tehokkuuden vertailu ristiriitojen ratkaisuissa tai ennaltaehkäisyssä on omalta osaltaan haastavaa. Kuten sanottu, Palantir luottaa käyttäjän omaan reagointiin ristiriitainformaation selvittämiseen. Tästä kuitenkin oli saatu hyviä tuloksia (Sarma, Redmiles ja Hoek 2012). DLS -kieltä sovelletaan suoraan versionhallintaan saaden ratkaistuiksi 11.4% kaikista ristiriidoista lähes onnistuneesti (Pan ym. 2021), suosien pieniä 1-2 rivin ristiriitoja. MergeBERT ilmoittaa ratkaisuprosenteikseen 63-68. MergeHelper luottaa myös enemmän käyttäjän toimintaan välittäessään "viipaloitua"historiaa. Vaikka neurokehys antaa numeroina parhaimman tuloksen, on testitilanteet pystytty valitsemaan hyvinkin tarkasti, tästä syystä tulokset eivät välttämättä välitä suoraan oikeisiin tilanteisiin.

Yhteenvetona näyttäisi siltä, että koska yhtä parasta tai suosituinta tapaa löytää ristiriitaisuuksia ei näyttäisi olevan, on uusien työkalujen kehittäminen hyödyllistä ellei jopa tarpeellista. Tutkimuksen tarkastellessa kuitenkin vain neljää työkalua pitää todeta sen rajallisuus ja ymmärtää ettei aukottomuuteen päästä. Tulevaisuudessa työkalujen kehitykseen varmasti panostetaan lisää ja erilaiset järjestelmät muuttuvat yhä teknisemmiksi, mutta mahdollisesti ja toivottavasti myös käyttäjäystävällisemmiksi.

Lähteet

Chacon, Scott, ja Ben Straub. 2014. *Pro git.* Springer Nature.

Da Silva, Léuson, Paulo Borba ja Arthur Pires. 2022. “Build conflicts in the wild”. *Journal of Software: Evolution and Process* 34 (4): e2441. https://doi.org/https://doi.org/10.1002/smr.2441. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2441. https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2441.

Ghiotto, Gleiph, Leonardo Murta, Márcio Barros ja Andre Van Der Hoek. 2018. “On the nature of merge conflicts: a study of 2,731 open source java projects hosted by github”. *IEEE Transactions on Software Engineering* 46 (8): 892–915.

Grinter, Rebecca E. 1996. “Supporting articulation work using software configuration management systems”. *Computer Supported Cooperative Work (CSCW)* 5:447–465.

Nishimura, Yuichi, ja Katsuhisa Maruyama. 2016. “Supporting Merge Conflict Resolution by Using Fine-Grained Code Change History”. Teoksessa *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER),* 1:661–664. https://doi.org/10.1109/SANER.2016.46.

Pan, Rangeet, Vu Le, Nachiappan Nagappan, Sumit Gulwani, Shuvendu Lahiri ja Mike Kaufman. 2021. “Can Program Synthesis be Used to Learn Merge Conflict Resolutions? An Empirical Analysis”. Teoksessa *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE),* 785–796. https://doi.org/10.1109/ICSE43902.2021.00077.

Perry, Dewayne E, Harvey P Siy ja Lawrence G Votta. 2001. “Parallel changes in large-scale software development: an observational case study”. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 10 (3): 308–337.

Pilato, C Michael, Ben Collins-Sussman ja Brian W Fitzpatrick. 2008. *Version control with subversion: next generation open source version control.* "O’Reilly Media, Inc.".

Ruparelia, Nayan B. 2010. “The history of version control”. *ACM SIGSOFT Software Engineering Notes* 35 (1): 5–9.

Sarma, Anita, David F. Redmiles ja André van der Hoek. 2012. “Palantir: Early Detection of Development Conflicts Arising from Parallel Code Changes”. *IEEE Transactions on Software Engineering* 38 (4): 889–908. https://doi.org/10.1109/TSE.2011.64.

Sink, Eric. 2011. *Version control by example.* Nide 20011. Pyrenean Gold Press Champaign, IL.

Somasundaram, Ravishankar. 2013. *Git: Version Control for Everyone Beginner’s Guide.* Packt Pub.

Svyatkovskiy, Alexey, Sarah Fakhoury, Negar Ghorbani, Todd Mytkowicz, Elizabeth Dinella, Christian Bird, Jinu Jang, Neel Sundaresan ja Shuvendu K Lahiri. 2022. “Program merge conflict resolution via neural transformers”. Teoksessa *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering,* 822–833.

Van Deursen, Arie, ja Paul Klint. 2002. “Domain-specific language design requires feature descriptions”. *Journal of computing and information technology* 10 (1): 1–17.