Koodikatselmointisuunnitelman kehittäminen .NET-kehittäjätiimille

Amanda Kajander

2019 Laurea

Laurea-ammattikorkeakoulu

Koodikatselmointisuunnitelman kehittäminen .NET-kehittäjätiimille

Amanda Kajander
Tietojenkäsittely
Opinnäytetyö
Marraskuu, 2019

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittelyn koulutusohjelma
Tradenomi (AMK)

Amanda Kajander

Koodikatselmointisuunnitelman kehittäminen .NET-kehittäjätiimille

Vuosi 2019 Sivumäärä 48

Koodikatselmointi on ohjelmistokehityksen työvaihe, jolla pyritään parantamaan ohjelmakoodin laatua löytämällä virheet aikaisessa vaiheessa, lisäämään ja vahvistamaan katselmoinnissa mukana olevien osapuolten keskinäistä yhteistyötä sekä mahdollistamaan tietojen ja taitojen jakamisen osapuolten kesken. Koodikatselmointityyppejä on monia, joita tyypillisesti jaotellaan niiden muodollisuuden mukaan, eikä ole yhtä oikeaa tapaa katselmoida koodia, mikä sopisi automaattisesti kaikille organisaatioille. Kehittämistehtävän tarkoituksena oli selvittää Suomen Handelsbankenin .NET-kehittäjätiimin tarpeet koodikatselmoinnille ja luoda suunnitelma, jonka pohjalta saadaan yhtenäinen koodikatselmointiprosessi .NET-kehittäjätiimille.

Tutkimusmenetelmänä käytettiin aktiivista havainnointia eli osallistuvaa havainnointia, jolla saatiin todenmukainen kokonaiskuva kehittämiskohteesta. Havainnoinnin tulokset jaettiin luokkiin niiden aihealueen mukaan. Luokkia hyödynnettiin tutkimuskirjallisuuden läpikäynnissä ja niiden avulla oli mahdollista hyödyntää myös sisällön analyysia tietoperustan luomiseksi. Kehittämismenetelmistä hyödynnettiin SWOT-analyysia sopivien koodikatselmointityyppien löytämiseksi. Kehittäjien sitoutuminen koodikatselmointiprosessiin nousi esiin yhtenä merkittävistä tekijöistä onnistumisen kannalta. Myös koko organisaation riittävällä ymmärryksellä ja sopivilla koodikatselmointitavoilla ilmeni olevan vaikutusta koodikatselmointiprosessin toimivuuteen. Näiden perusteella koodikatselmointisuunnitelmassa esitettiin suositukset käytettävistä koodikatselmointitavoista sekä muista koodikatselmointiprosessissa huomioitavista asioista. Koodikatselmointisuunnitelma toimii runkona koodikatselmointiprosessille, jonka toimintatavoista päätetään lopulta .NET-kehittäjätiimissä, jolloin jokainen sitoutuu yhteisiin toimintatapoihin ja yhteiseen prosessiin.

Asiasanat: koodikatselmointi, koodikatselmointityypit, ohjelmistokehitys, analysointityökalut

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bachelor's thesis

Amanda Kajander

Developing a code review plan for .NET developer team

Year 2019 Pages 48

Code review is a part of software development that aims to increase the quality of program code by detecting defects on early phase, enhance parties to work in close collaboration and enable knowledge sharing between each other. There are several code review types that are typically categorized by the formality and there doesn't exist the one and only way to review code that automatically suits for every organisation. The purpose of development work was to investigate what are the needs for code review at Handelsbanken Finland .NET team and create a plan for coherent code review process.

Thesis' research method was active observation that enabled to create a big picture of subject. The results of observation were meted out by the topic which was useful for analysing the research literature and creating the knowledge basis. SWOT analysis was utilized in development work to discover suitable code review types for Handelsbanken Finland's .NET team. The research exposed that developer's commitment to code review is a significant factor to success in code review. Also, the understanding of code review in the whole organization and suitable code review types appeared to affect the functionality of code review process. Based on these points, the code review plan included the recommendations for code review types that would be most suitable and other things that must be considered in code review process. Ultimately, the .NET developers will decide the procedures in code review process that are based on code review plan.

Keywords: code review, code review types, software development, code analyzing tools

Sisällys

1 Johdanto

Tämän opinnäytetyön aiheena on koodikatselmointisuunnitelman kehittäminen Handelsbankenin .NET-kehittäjätiimin käyttöön. Handelsbankenin .NET-kehittäjätiimillä on käytettävissään konsernin asettamat vähimmäisvaatimukset koodikatselmoinnille, mutta ajan myötä vaatimukset ovat muuttuneet ja lisääntyneet sekä työmäärä on kasvanut, minkä vuoksi .NET-kehittäjätiimi on kokenut koodikatselmointisuunnitelman tarpeelliseksi. Yhteisen koodikatselmointitavan puuttuminen on vaikeuttanut erityisesti koodin laadun valvomista sekä ohjeiden ja sääntöjen noudattamisen seuraamista. Lisäksi koodikatselmoinnin taso on voinut vaihdella kehittäjästä riippuen. Koodikatselmointisuunnitelmalla pyritään löytämään ratkaisu nykyhetken haasteisiin, kehittämään olemassa olevia toimintatapoja sekä vahvistamaan koko kehittäjätiimin vastuuta kehitystöistä yksilöiden henkilökohtaisen vastuun sijaan.

Tämä opinnäytetyö on luonteeltaan tutkimuksellinen kehittämistyö, jonka teoreettisessa osassa kuvataan ensin ohjelmistokehityksen perusasiat ja sen jälkeen esitellään koodikatselmointia yleisesti, mitä sillä tavoitellaan, minkälaisilla toimintatavoilla sitä toteutetaan ja mitkä asiat vaikuttavat onnistuneeseen koodikatselmointiin. Lisäksi opinnäytetyö sisältää kehittämistyön, joka esitellään teoreettisen osion jälkeen. Kehittämiskohdetta tarkastellaan osallistuvan havainnoinnin keinoin. Havainnointia hyödynnetään nykyhetken kokonaiskuvan luomiseen ja sen avulla selvitetään muun muassa, miten koodikatselmointia toteutetaan tällä hetkellä ja miten siihen asennoidutaan .NET-kehittäjätiimissä. Tämä opinnäytetyö on laadullinen tutkimus ja aineiston keruussa hyödynnetään sisällön analyysia, jotta koodikatselmoinnista saadaan luotua kokonaisvaltainen ja mahdollisimman todenmukainen kuva. Kehittämismenetelmänä käytetään SWOT-analyysia, joka mahdollistaa koodikatselmointityyppien vertailun keskenään. Vertailun avulla valitaan koodikatselmointityypit, joita suunnitelmassa suositellaan. Näin huomioidaan toimeksiannossa esiin tuodut vaatimukset ja toiveet sekä havainnoinnin myötä ilmenneet haasteet. Kehittämistyön tuloksena on koodikatselmointisuunnitelma, joka on sellaisenaan valmis .NET-kehittäjätiimin käytettäväksi. Lopuksi yhteenvedossa käydään läpi olennaisia seikkoja tässä opinnäytetyössä, esitellään johtopäätökset sekä pohditaan jatkokehitysmahdollisuuksia.

2 Kehittämiskohde

Opinnäytteen kehittämiskohteena on Suomen Handelsbankenin .NET-kehittäjätiimin koodikatselmointikäytäntö eli koodikatselmointisuunnitelman kehittäminen .NET-kehittäjätiimille. Koodikatselmointisuunnitelmalla tarkoitetaan suunnitelmaa, miten koodikatselmointi toteutetaan Suomen Handelsbankenin .NET-kehittäjätiimissä. Konserni on asettanut sisäisiä vaatimuksia muun muassa ohjelmakoodin katselmoinnille sekä kehitystöiden etenemiselle, mutta jokaisen kehittäjätiimin vastuulla on huolehtia, että kaikissa työtehtävissä noudatetaan koodikatselmointiin liittyviä sääntöjä ja vaatimuksia. Vastuu on tällä hetkellä kokonaan kehittäjillä itsellään, sillä koko konsernin kattavaa koodikatselmointisuunnitelmaa ei ole olemassa.

Tämä tarkoittaa, että .NET-kehittäjätiimillä ei ole tällä hetkellä käytössään suunnitelmaa, joka sisältäisi kunkin tiimin tarpeisiin valitut koodikatselmointitavat ja koodikatselmointiprosessin kuvauksen. Konserni on asettanut vähimmäisvaatimukseksi suorittaa koodikatselmointia, mutta ei ota kantaa kunkin yksikön siinä käyttämiin koodikatselmointityyppeihin tai suoritustapaan.

Koodikatselmointia tulee suorittaa turvallisuuden, tehokkuuden sekä ylläpidettävyyden kannalta. Koska Handelsbankenin käytössä ei ole konsernin yhteistä koodikatselmointisuunnitelmaa, jokainen kehittäjä on valinnut omaan työhönsä sopivimmat toimintatavat koodikatselmoinnin suorittamiseen tilanteesta ja projektista riippuen. Tällöin koodikatselmoinnin laatu on vaihdellut kehittäjästä ja projektissa käytössä olleista resursseista riippuen ja pahimmassa tapauksessa koodikatselmointi on voinut jäädä kokonaan toteuttamatta. Tiimillä ei ole myöskään ollut mahdollisuutta seurata ilmeneviä virheitä, sillä yhdenmukaista tapaa suorittaa seurantaa ei ole ollut. Seurannan puutteen vuoksi organisaation johdolla ja kehittäjätiimillä ei ole tilastoihin perustuvaa tietoa osa-alueista, joissa olisi eniten tarvetta kehittää ammattitaitoa, sillä tällä hetkellä koodikatselmoinnin tuloksien tallentamisesta ja tilastoinnista ei ole yhteistä toimintaohjetta. Ratkaisuna haasteisiin kehitetään koodikatselmointisuunnitelma, jonka avulla varmistetaan yhteiset koodikatselmointitavat .NET-kehittäjätiimissä ja saadaan koodikatselmointi kiinteäksi osaksi kehitystyöprosessia. (IT Development Process at Handelsbanken 2019; System Development 2019)

2.1 Toimeksiantaja

Opinnäytetyö on kehittämistyö, jonka toimeksiantajana on Svenska Handelsbanken AB (julk), Suomen sivukonttoritoiminta (Handelsbanken) on osa kansainvälistä pankkikonsernia, jonka kotipaikka on Tukholmassa, Ruotsissa. Handelsbanken perustettiin vuonna 1871 ruotsalaisten liikemiesten ja yritysten toimesta. Laajentuminen Suomessa alkoi perustamalla edustusto Helsingin keskustaan vuonna 1985. Vuonna 1990 perustettiin Handelsbankenin Suomen tytärpankki ja sitä seuraavana vuonna se sai täydet sivukonttorioikeudet ollen Suomen ensimmäinen ulkomaisessa omistuksessa oleva rahoituslaitos. Ensimmäinen pankkikonttori perustettiin helmikuussa 1994, minkä jälkeen konttoreita on avattu eri puolille Suomea. Vuonna 2019 Handelsbanken listaa kotimarkkinoikseen Ruotsin lisäksi Suomen, Norjan, Tanskan, Iso-Britannian ja Alankomaiden markkinat. Kansainvälisesti Handelsbankenilla on myös konttoreja ja edustustoja muissa Euroopan maissa, kuten esimerkiksi Saksassa ja Ranskassa sekä Aasiassa ja Pohjois-Amerikassa, joissa palvellaan kaikkien kotimarkkinoiden asiakkaita. (Handelsbanken 2019a; Handelsbanken 2019b; Handelsbanken 2019c)

Handelsbankenissa IT-yksiköiden tehtävät ja niiden koko vaihtelevat maittain ja osa IT-toiminnoista ja -tehtävistä on keskitetty Ruotsin IT-yksikköön. Yhteensä koko konsernissa työskente-

lee IT-tehtävissä noin 1300 henkilöä, joista valtaosa Ruotsissa. Suomen IT-yksikössä työskentelee yli 30 henkilöä tehtävänään koko Suomen Handelsbankenin ja sen tytäryhtiöiden järjestelmien kehitys ja ylläpito sekä infraan liittyvät tehtävät. (Handelsbanken 2019d)

.NET-kehittäjätiimissä työskentelee kymmenen henkilöä, joista osa on konsultteja. Kehittäjätiimin vastuulla on muun muassa sovelluskehitys ja -ylläpito sekä järjestelmäintegraatiot. Pääasiallisiin tehtäviin kuuluvat niin sisäisten sovellusten kuin myös asiakkaille tarjottavien palveluiden kehitys ja ylläpito. Tiimille osoitetut tehtävät ovat keskenään hyvin erilaisia ja vaihtelevat muun muassa sisältönsä ja kokonsa mukaan. Sisäiseen käyttöön tarkoitettuja sovelluksia kehitetään muun muassa tukemaan raportointia tai muita hallinnollisia tehtäviä. Asiakasrajapinnassa muun muassa verkkopankin Säästä ja Sijoita -osion, mobiilipankin sekä PSD2-direktiivin myötä kolmansille osapuolille tarjottavien rajapintojen toteutuksessa on käytetty .NET-tekniikkaa.

2.2 Tavoite ja tutkimusongelma

Tämän opinnäytteen tavoitteena on luoda suunnitelma, jota toteuttamalla koodikatselmoinnista tulee kiinteä osa kehitystyöprosessia. Koodikatselmointisuunnitelman ansiosta kehittäjien ei tarvitse miettiä jokaisen tehtävän kohdalla, kuinka koodikatselmointivaatimukset tulee täytettyä, vaan suunnitelmaa noudattamalla vaatimukset täytetään varmasti. Lisäksi tavoitteena on ohjelmakoodin laadun parantaminen, osaamisen jakaminen sekä mahdollisuus tehtävien tasaisempaan jakoon. Tarkoituksena on luoda kokonaiskuva koodikatselmoinnista ja selvittää, millaisilla toimenpiteillä saavutetaan .NET-kehittäjätiimille asetetut vaatimukset koodikatselmoinnin osalta samalla maksimoiden koodikatselmoinnista saatavat hyödyt ja minimoiden siitä aiheutuvat haitat ja haasteet.

Kehittämistyön tavoitteena on suunnitelma, joka sisältää kuvauksen .NET-kehittäjätiimin yhteisistä koodikatselmointitavoista, siinä hyödynnettävistä työvälineistä sekä vastuualueet, jotta koodikatselmointi saadaan toimivaksi prosessin vaiheeksi. Koodikatselmointisuunnitelman avulla varmistetaan, että ohjelmakoodissa on kehittäjästä riippumatta huomioitu turvallisuus, tehokkuus sekä ylläpidettävyys. (IT Development Process at Handelsbanken 2019; System Development 2019)

Tässä opinnäytetyössä pyritään vastaamaan seuraavaan tutkimuskysymykseen: Millainen koodikatselmointitapa sopii Suomen Handelsbankenin .NET-kehittäjätiimille? Tutkimusongelma on koko opinnäytetyön lähtökohta ja näin ollen pohjautuu opinnäytetyön tarkoitukseen ja tavoitteeseen, pyritään näin ollen vastaamaan myös seuraaviin tutkimuskysymyksiin. Mitä koodikatselmointi on? Mitä hyötyä koodikatselmoinnista on? Mitä haasteita koodikatselmointi voi tuoda? Mitä asioita koodikatselmointisuunnitelman tulee sisältää, jotta sillä saavutetaan toimiva koodikatselmointiprosessi osana koko kehitysprosessia?

2.3 Aiheen rajaus

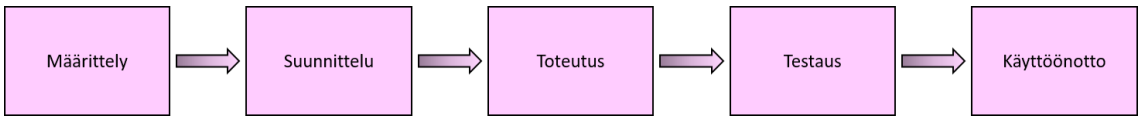
Tämän kehittämistyön toimeksiannossa esitetään, että koodikatselmointisuunnitelman tulee olla toteutuskelpoinen olemassa olevilla työvälineillä ainoastaan työtapoja ja kehitystyöprosessia muuttamalla. Suunnitelman toteutuksen ei tulisi aiheuttaa lisäkustannuksia toimeksiantajalle lukuun ottamatta työtunteja, jotka koodikatselmoinnin implementointi osaksi kehitystyötä vaatii .NET-kehittäjätiimiltä. Koska minkäänlaisia investointeja uusiin ohjelmistoihin tai lisensseihin ei tehdä, tässä opinnäytetyössä käsitellään vain sellaisia työvälineitä, jotka ovat jo konsernissa käytössä.

Suunnittelu-, arkkitehtuuri- ja muut katselmoinnit rajautuvat aiheen ulkopuolelle, sillä tämän opinnäytteen tarkoituksena on kehittää koodikatselmointisuunnitelma eli aihetta käsitellään ainoastaan ohjelmakoodin katselmoinnin näkökulmasta. Koska koodikatselmointiin liittyy sen teknisen toteutuksen lisäksi muitakin osa-alueita, ei aihetta käsitellä pelkästään teknisestä näkökulmasta. Tässä opinnäytetyössä huomioidaan aihe myös työntekijöiden ja tiimin kannalta eli käsitellään koodikatselmointiin vahvasti liittyviä vuorovaikutustaitoja sekä kehittäjien, että katselmoijien roolien näkökulmasta.

3 Ohjelmistokehitys

Ohjelmistokehitys on prosessi, jossa pyritään ratkaisemaan ongelma tai muu tarve kehittämällä tietokoneohjelma. Ohjelmistokehitysprosessi vaihtelee tarpeen koon ja tavoitteiden mukaan. Toisinaan se voi kestää viikkoja, toisinaan vuosia, mutta prosessin vaiheet ovat pitkälti samat prosessin kestosta huolimatta. Ohjelmistokehitys alkaa tarpeen määrittelyllä, joka voidaan toteuttaa vaatimusmäärittelyillä tai kuvaamalla käyttötapauksia. Vaatimusmäärittelyssä esitetään ne ominaisuudet ja toiminnallisuudet, jotka ohjelmalta vaaditaan täyttääkseen tarpeen. Vaatimusmäärittelyt sisältävät usein myös ominaisuuksia, jotka eivät ole olennaisia tarpeen täyttämiseksi, mutta tehostavat esimerkiksi työntekoa, jolloin vaateiden priorisointi on tärkeää. Tällöin varmistustaan, että olennaisimmat ominaisuudet ja toiminnot toteutetaan ensin ja sen jälkeen seuraavaksi tärkeimmät ja niin edelleen. Suunnitteluvaiheessa valitaan toimintatavat, kuinka ohjelmaa lähdetään toteuttamaan ja mitä tekniikoita sen toteuttamisessa tullaan käyttämään. Toteutusvaiheessa kehittäjä ohjelmoi vaatimusmäärittelyissä kuvattujen ominaisuuksien ja toimintojen mukaisen ohjelman sekä testaa ohjelmaa esimerkiksi yksikkötestein. Seuraavana on testausvaihe, jossa tilaajan tai projektinjohdon määrittämät testaajat käyttävät ohjelmaa siinä tarkoituksessa, jota varten se on luotu. Ohjelman toimivuutta testataan vaatimuksiin nähden ja varmistetaan, että ohjelma täyttää kaikki sille asetetut vaatimukset. Käyttöönottovaiheessa ohjelma julkaistaan yleensä ensin tuotannonkaltaiseen ympäristöön, jossa voidaan vielä varmistua sen luotettavuudesta ja sen jälkeen tuotantoympäristöön ensin vain pilottikäyttöön tai suoraan loppukäyttäjille. (Immonen 2002; Pressman 2010, 31-32; Sommerville 2011, 29-42)

Tavallisesti ohjelmistokehitysprosessit sisältävät kaikki kuviossa 1 esitetyt vaiheet, mutta projektista riippuen eteneminen vaiheesta toiseen voi vaihdella. Yksi perinteisimmistä prosessimalleista on vesiputousmalli, joka etenee lineaarisesti vaiheesta toiseen ja siirtyminen seuraavaan vaiheeseen tapahtuu vasta sitten, kun edellinen vaihe on valmis.



Kuvio 1: Ohjelmistokehitysprosessin vaiheet (lineaarinen prosessikulku)

Ketterässä kehityksessä vaiheesta toiseen siirtyminen tapahtuu iteratiivisesti eli aiemmin suoritettuihin vaiheisiin voidaan palata uudelleen. Päinvastoin kuin vesiputousmallissa, jossa käyttöönoton jälkeen on valmis lopputuote, ketterässä kehityksessä kaikki vaiheet toistetaan useita kertoja niin kuin kuviossa 2 on esitetty ja lopputuotetta rakennetaan pienissä osissa.

Kuvio 2: Iteratiivinen prosessikulku

Kehitysprosessi on joustavampi, jos esimerkiksi vaatimukset muuttuvat projektin aikana. Vesiputousmallin ja ketterän kehityksen lisäksi on myös muita malleja kuten protoilu ja evo-malli. Protoilumallin tarkoituksena on luoda prototyyppi, joka sisältää osan vaadituista ominaisuuksista ja toiminnoista, mutta ei kaikkia niistä. Prototyypissä on esimerkiksi toteutettu vain korkeimman prioriteetin omaavat vaatimukset. Prototyypin valmistuttua sitä testataan ja testauksen tulosten perusteella joko hyväksytään tai hylätään. Jos prototyyppi hyväksytään, tulee sen jälkeen päättää, kehitetäänkö prototyypistä lopputuote eli jatketaan prototyypin kehittämistä vai aloitetaanko lopputuotteen rakentaminen alusta ja hyödynnetään prototyyppiä tehdessä saatuja kokemuksia. Evo-mallissa lopputuotteesta luodaan useita versioita. Evo-mallilla tarkoitetaan useiden peräkkäisten vesiputousmallin mukaisten prosessien toteuttamista niin, että jokaisen lopputuloksena on aina uusi versio lopputuotteesta. Evo-malli etenee lineaarisesti vaiheesta seuraavaan ja sen jälkeen prosessista seuraavaan. (Immonen 2002; Pressman 2010, 31-32; Sommerville 2011, 58-64)

3.1 Ohjelmointikäytännöt

Ohjelmointikäytännöt ovat yleisiä ohjeita, jotka koskevat muun muassa ohjelmointityyliä, -menetelmiä ja ohjelmakoodin rakennetta. Ohjelmointikäytäntöjä noudattamalla ohjelma-

koodi on johdonmukaisempaa, mikä helpottaa kehittäjätiimin työtä, sillä ohjelmointikäytäntöjen myötä ohjelmakoodi on yhdenmukaisempaa huolimatta siitä, kuka sen on kirjoittanut. Ohjelmointikäytäntöjä on olemassa ohjelmointikielittäin, kuten esimerkiksi Microsoftin esittämät ohjelmointikäytännöt C#-ohjelmointikielelle, mutta organisaatioilla on usein myös omia ohjelmointikäytäntöjä. Organisaatiokohtaiset ohjelmointikäytännöt saattavat perustua ohjelmointikielikohtaisiin ohjelmointikäytäntöihin, mutta niissä on lisäksi huomioitu organisaatiolle tärkeitä osa-alueita. Ohjelmointikäytänteiden tarkoituksena on ohjata kehittäjää kirjoittamaan laadukkaampaa koodia. Yleisesti ohjelmakoodi koetaan laadukkaaksi silloin, kun se on ylläpidettävää, joustavaa, siirrettävää, uudelleenkäytettävää, luettavaa, testattavaa ja ymmärrettävää. (McConnell 2004, 464-465, 565; Microsoft 2015)

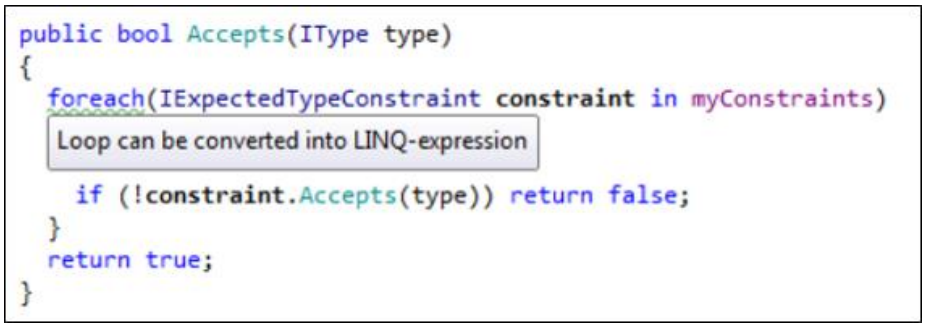
Ylläpidettävyys mahdollistaa muutosten, lisäysten, parannusten ja korjausten tekemisen koodiin tehokkaasti ilman merkittäviä muutoksia olemassa olevaan koodiin tai ohjelmiston rakenteeseen. Joustavuudella ja siirrettävyydellä tarkoitetaan ohjelmistokoodin riippumattomuutta tiettyyn ympäristöön. Joustavan ohjelmistokoodin käyttöä tai ympäristöä on mahdollista muuttaa alkuperäisestä ja siirrettävää ohjelmistokoodia on mahdollista käyttää erilaisissa ympäristöissä. Uudelleenkäytettävyys mahdollistaa sananmukaisesti ohjelmistokoodin osien käytön toisissa ohjelmistoissa eli ohjelmistokoodi on kirjoitettu mahdollisimman modulaarisesti. Luettavuudella ja ymmärrettävyydellä tarkoitetaan osin samaa, mutta ne mainitaan erikseen, sillä ymmärrettävyydellä tarkoitetaan erityisesti yleistä ymmärrettävyyttä kokonaistasolla, kun taas luettavuudella tarkoitetaan ohjelmistokoodin selkeää esitystä ja yksityiskohtien ymmärrettävyyttä. Testattavuuden ansiosta koodin eri osia voidaan testata järkevästi ja sille voidaan kirjoittaa muun muassa kattavia yksikkötestejä. (McConnell 2004, 464-465, 565)

3.2 Ohjelmakoodin analysointityökalut

Ohjelmakoodin analysointia varten on olemassa erilaisia työvälineitä ja ohjelmointiympäristöön asennettavia lisäosia eri ohjelmointikielille ja -ympäristöille. Tällaiset työvälineet auttavat havaitsemaan virheet ja puutteet heti kehityksen varhaisessa vaiheessa, mikä on edullista organisaation kannalta, sillä virheiden korjaaminen vie vähemmän resursseja, mitä aikaisemmassa vaiheessa ne havaitaan. Automaattiset analysointityökalut auttavat kehittäjää noudattamaan ohjelmointikäytäntöjä. Tätä varten Handelsbankenin .NET-kehittäjätiimillä on käytössään ReSharper ja SonarQube. (IT Development Process at Handelsbanken 2019; McConnell 2004, 75)

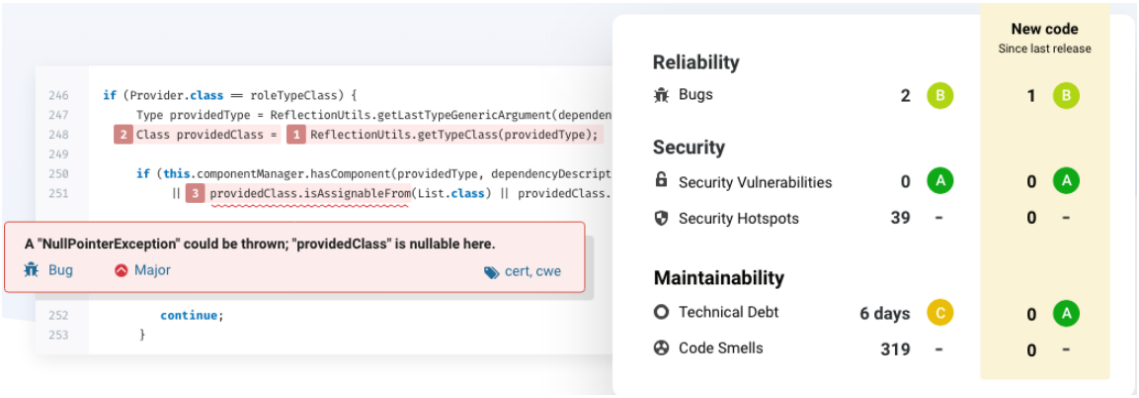
ReSharper on Visual Studio -ohjelmointiympäristöön asennettava lisäosa, joka auttaa kehittäjää noudattamaan ohjelmointikäytänteitä. JetBrains, jonka tuotteita ReSharper on, tarjoaa maksullisia lisenssejä yksityis- ja yrityskäyttöön sekä ilmaisia lisenssejä avoimen lähden koodin projekteihin. ReSharper perustuu Microsoft Visual Studion IntelliSenseen, mutta sisältää lisäksi muita ominaisuuksia, joita IntelliSensessä ei ole lainkaan. ReSharperin ominaisuuksiin kuuluvat muun muassa ohjelmakoodin analysointi, muokattavuuden helpottaminen, generointi

sekä virheiden ja epämääräisen ohjelmakoodin tunnistaminen ja eliminointi. ReSharper tekee jatkuvaa ohjelmakoodin analyysia ja näyttää mahdolliset ongelmat kehittäjälle heti. Tällöin virheet on mahdollista saada korjattua jo ennen kuin ohjelmakoodi edes käännetään. Jos ReSharper havaitsee kirjoitetussa ohjelmakoodissa virheitä, se korostaa ne näyttämällä aaltoviivan ongelman kohdalla ja tarjoamalla korjausehdotuksia, jos mahdollista niin kuin kuviossa 3 esitetään. ReSharperissa käytettäviä hyvän ohjelmointitavan sääntöjä on mahdollista muuttaa sen asetuksissa, jotta ne vastaavat organisaation ohjelmointikäytäntöä. (Visual Studio 2019; Jetbrains 2019a; Jetbrains 2019b)



Kuvio 3: ReSharperin koodin analysointi (Jetbrains 2019c)

SonarQube on työkalu automaattiseen koodikatselmointiin. Ohjelmointisäännöt eli ohjelmointikäytänteet viedään SonarQubeen, joita vastaan se analysoi ohjelmakoodin. Asetuksilla on mahdollista vaikuttaa esimerkiksi siihen, mitä tiedostoja SonarQube käy läpi. Analysoinnin tuloksena SonarQube muodostaa ohjelmakoodista raportin, jossa näytetään siitä löytyneet virheet, haavoittuvuudet sekä epämääräisen ohjelmakoodin, kerrotaan, miksi havaittu ongelma kannattaa korjata ja mahdollisesti myös, miten se voidaan korjata. Raportti näyttää myös aika-arvion siitä, kuinka kauan korjausten tekemiseen kuluu aikaa. Ongelmat luokitellaan niiden vakavuuden mukaan ja ohjelmakoodi luokitellaan luotettavuuden mukaan, kuinka paljon se sisältää ongelmia ja minkä tasoisia ongelmia ne ovat. Ohjelmakoodin analysoinnin lisäksi SonarQube voi varmistaa, että vain ohjelmakoodi, joka kääntyy, siirtyy eteenpäin repositoriin. Kuviossa 4 esitetään SonarQuben tekemän analyysin lopputulosta. Kuviossa oikealla näytetään raportti analyysin tuloksista, jossa näytetään myös vertailu, onko ohjelmakoodin laatu parantunut aiempaan versioon verrattuna ja vasemmalla on näkymä, kun virhettä tarkastelee SonarQuben kautta tarkemmin. (SonarQube 2019a; SonarQube 2019b; SonarQube 2019c; SonarQube 2019d; SonarQube 2019e)



Kuvio 4: SonarQuben koodin analysointi (SonarQube 2019f)

4 Koodikatselmointi

Koodikatselmointi on kehitystyön vaihe, jossa joku muu kuin kehittäjä itse käy läpi tehdyn ohjelmakoodin. Toisin sanoen koodikatselmointi on lähdekoodille tehtävä vertaisarviointi, jonka tarkoituksena on löytää mahdollisia virheitä, puutteita ja muita kehitettäviä kohtia ohjelmakoodista. Koodikatselmoinnin avulla voidaan varmistua, että ohjelmointikäytäntöjä noudatetaan ja näin ollen ohjelmakoodi täyttää kaikki sille asetetut vaatimukset. (Wiegers 2002, 13)

Koodikatselmoinnissa keskitytään aina ohjelmakoodin arvioimiseen, ei kehittäjän ammattitaitoon arvioimiseen. Välillisesti se kuitenkin vaikuttaa aina kaikkiin, jotka osallistuvat koodikatselmointiprosessiin. Tästä kerrotaan lisää kappaleissa 4.3 ja 4.4. Vaikka keskiössä on aina ohjelmakoodi, ei koodikatselmointia yleensä suoriteta ainoastaan teknisestä näkökulmasta. Koodikatselmoinnilla on myös vahvat sosiaaliset vaikutukset kehittäjien välillä. Hyvin suunniteltu ja toteutettu koodikatselmointisuunnitelma maksimoi siitä saatavat hyödyt ja minimoi siitä mahdollisesti aiheutuvat haasteet. (Wiegers 2002, 13)

Koodikatselmointiin liittyy olennaisesti tarkistuslista, jossa määritellään ne osa-alueet, joita koodikatselmoinnissa halutaan käytävän läpi. Tarkistuslista pohjautuu yleisesti ohjelmointikäytäntöihin, joita kehittäjien halutaan noudattavan. Katselmoijat voivat hyödyntää tarkistuslistaa muistilistana, jolla varmistetaan, että kaikki vaaditut asiat huomioidaan ohjelmakoodia katselmoitaessa. Lisäksi tarkistuslistaa käytetään katselmointitilaisuudessa muistiona, jatkotoimenpiteiden seurannassa sekä puutteiden tilastoinnissa. Tarkistuslistan tarkoituksena on helpottaa ohjelmakoodin lukemista. Jos tarkistuslista on tehty yksityiskohtaisesti sisältäen myös ohjeistusta, vähentää se katselmoijan eli ohjelmakoodin lukijan muistinvaraista tulkintaa. Toisaalta tarkistuslista voi olla vain lista asioita, jotka katselmoinnissa tulee huomioida. (Zhu 2016, 816, 830, 846, 861, 876, 891 & 901)

4.1 Katselmointityypit

Koodikatselmoinnissa käytetään yleisiä katselmointityyppejä. Näitä katselmointityyppejä voidaan hyödyntää erilaisten tuotosten katselmoinnissa, kuten esimerkiksi projektien, suunnitelmien tai ohjelmakoodin katselmoinnissa. Tapoja suorittaa koodikatselmointia on olemassa useita ja usein ne muovautuvatkin kunkin organisaation tarpeiden mukaiseksi niin, että koodikatselmoinnilla saavutetaan mahdollisimman hyvin sille asetetut tavoitteet ja vaatimukset. Lisäksi käytettävään koodikatselmointitapaan vaikuttaa muun muassa kehittäjätiimi, resurssit sekä organisaation kulttuuri. Katselmointityyppejä tulkitaan joissain tapauksissa hieman eri tavalla lähteestä riippuen. Esimerkiksi Finnish Software Testing Boardin (FiSTB) julkaiseman Perustason sertifikaattisisällön mukaan virallisia katselmointityyppejä ovat tarkastus, tekninen katselmointi, läpikäynti sekä epämuodollinen katselmointi, joita ovat paritarkastus ja parikatselmointi (ISTQB 2018, s. 44-46). Sen sijaan Wiegers (2002, s. 13) jakaa katselmoinnin seuraaviin katselmointityyppeihin: tarkastus, ryhmäkatselmointi, läpikäynti, pariohjelmointi, työpisteen ääressä tapahtuva katselmointi sekä ad hoc -katselmointi. Edellä mainituista tavoista esimerkiksi läpikäynnissä, ryhmä- ja muodollisessa katselmoinnissa on hyvin paljon yhtäläisiä piirteitä, eikä ryhmäkatselmointia tulkita aina erilliseksi katselmointityypiksi. Katselmointityypit eroavat toisistaan muun muassa niiden muodollisuudessa. Tarkastus on näistä muodollisin ja ad hoc -katselmointi vähinten muodollinen. (ISTQB 2018; Wiegers 2002, 31-32)

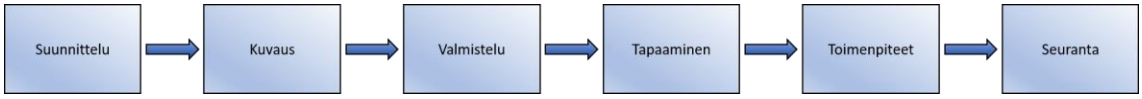
Luokittelua voidaan tehdä myös sen perusteella, missä kohtaa kehitysprosessia katselmointi tapahtuu. Tarkastus tapahtuu yleensä kehitysprosessin loppuvaiheessa, kun taas vähemmän muodollisia katselmointityyppejä kuten pariohjelmointia, parin kanssa tehtävää tarkastusta sekä pull requestejä toteutetaan usein kehitysprosessin varhaisemmissa vaiheissa. Pull request on nostettu yhdeksi esiteltävistä koodikatselmointityypeistä, sillä sen tarkoituksena on mahdollistaa ohjelmakoodille tehtävä katselmus muiden kuin tekijän toimesta silloin, kun käytössä on versionhallinta. (Github 2019)

4.1.1 Tarkastus

Tarkastukset ovat yksi koodikatselmoinnin tyypeistä, joita hyödynnetään niin suurien kokonaisuuksien kuin myös koodin katselmointiin. Tarkastusten järjestäminen vaatii muita koodikatselmointityyppejä enemmän resursseja sen muodollisuutensa vuoksi. Tarkastuksiin ominaista on usein se, että siihen osallistuu useita henkilöitä, jotka voivat olla myös kehittäjätiimin ulkopuolelta. Jokaisella tarkastukseen osallistuvalla on oma, ennalta määritelty roolinsa. Kaikki osallistujat käyvät etukäteen läpi materiaalit ja valmistautuvat tarkastukseen huolella. Yleensä tarkastuksessa on mukana vähintään seuraavat roolit: fasilitaattori, katselmoinnin vetäjä, tekijä, katselmoijat sekä sihteeri. Joissain tapauksissa mukana voi olla myös erillinen lukija. Lisäksi tarkastusprosessiin kuuluu johto, jonka tehtäviin kuuluu päätöksen teko katselmointien suorituksesta ja tavoitteiden saavuttamisesta sekä resursseista huolehtiminen. Fa-

silitaattorin tehtävänä on toimia katselmointitilaisuuden puheenjohtajana huolehtien katselmoinnin tehokkaasta läpiviennistä ja toimien sovittelijana mahdollisissa erimielisyyksissä. Katselmoinnin vetäjän tehtäviin kuuluu organisoida katselmointitilaisuuden järjestäminen, huolehtia siitä, että jokaisella osallistujalla on omat tehtävänsä sekä seurata jatkotoimenpiteiden toteutuminen. Katselmoija tekee kriittisen analyysin tarkastuksen kohteesta. Katselmoinnin vetäjän rooli on merkittävä katselmointitilaisuuden onnistumisen kannalta, sillä hänelle osoitetuissa tehtävissä onnistuminen on verrattavissa koko katselmoinnissa onnistumiseen. Katselmoinnin vetäjällä on myös suuri vaikutus muun muassa osallistujien motivaatioon sekä tilaisuuden yleiseen toimivuuteen. Tekijän roolin vastuulla on tuottaa katselmoitava aineisto. Katselmoija käy läpi tarkastuksessa olevan aineiston pyrkien havaitsemaan siitä puutteita ja virheitä. Katselmoijia voi osallistua useita samaan tarkastukseen. Yleensä tällöin pyritään löytämään sellaisia katselmoijia, jotka edustavat eri osa-alueita, jotta puutteita ja virheitä olisi mahdollista löytää laajasti läpikäytävästä aineistosta. Kirjurin tehtävänä on dokumentoida kaikki katselmoinnissa ilmenevät löydökset sekä avoimet asiat. Lukijan rooli ei ole välttämätön tarkastuksen järjestämisen kannalta, mutta tietyissä tilanteissa erillisestä lukijasta voi olla hyötyä. Lukija esittää tarkastuksen kohteen niin kuin ymmärtää sen. Lukijan roolissa erityistä on se, että kyseinen rooli ei ota kantaa tarkastuksen kohteeseen, vaan esittää sen juuri sellaisena kuin sen ymmärtää. Tällöin tarkastuksen kohde voidaan esittää osallistujille sellaisena kuin se ymmärretään, eikä niin kuin tekijä on sen tarkoittanut, miten tekijä itse saattaisi sen esittää ryhmälle. (ISTQB 2018, 44-46; McConnell 2004, 486-487; Wiegers 2002, 33)

Tyypillisesti tarkastuksen läpiviennissä hyödynnetään organisaation omaa, yleistä tarkastuslistaa, johon on listattu vähimmäisvaatimukset asioista, jotka on käytävä läpi tarkastuksessa. Tarkistuslistaa käytetään perustana kulloinkin järjestettävään katselmointitilaisuuteen, johon lisätään tarkastettavan kohteeseen oleellisesti liittyviä piirteitä, jotka tulee käydä läpi. Tarkastuksen vaiheet ovat suunnittelu, kuvaus, valmistelu, tapaaminen, jatkotoimenpiteiden suorittaminen sekä seuranta, jotka on esitetty kuviossa 5.



Kuvio 5: Tarkastuksen vaiheet

Vaiheesta toiseen siirrytään lineaarisesti niin, että tarkastus alkaa aina suunnittelusta ja päättyy seurantaan. Suunnitteluvaiheessa valmistellaan aineisto sekä valitaan osallistujat muodollista katselmointia varten. Kuvauksessa määritellään roolit ja esitetään aineisto osallistujille. Valmisteluvaiheessa osallistujat käyvät läpi katselmoitavan aineiston oman roolinsa näkökulmasta. Tapaamisessa esitetään huomioita havaituista virheistä valvojan johdolla. Tapaamisen jälkeen suoritetaan asiat, jotka kirjattiin tarkistuslistaan, minkä jälkeen järjestetään seurantatapaaminen, johon vähintään valvoja osallistuu. Tarkastuksessa havaitut virheet kirjataan

ylös täsmällisesti niin, että virheestä mainitaan muun muassa sen vakavuus, tyyppi sekä tieto siitä, missä kehitystyön vaiheessa virhe on syntynyt. Tarkastuksissa havaitut virheet tallennetaan seurantaa ja mahdollista tilastointia varten. Tilastojen avulla voidaan seurata tarkastusten tuomia hyötyjä pitkällä tähtäimellä ja näin ollen kehittää katselmointitilaisuuksia ja -prosessia. (ISTQB 2018, 45-46; McConnell 2004, 485, 487-489; Wiegers 2002, 34)

Tarkastuksissa havaitaan yleensä enemmän virheitä ja kehityskohteita kuin vapaamuotoisemmissa katselmoinneissa, koska osallistujat ovat yleensä ehtineet valmistautua tilaisuutta varten hyvin ja suurempi määrä osallistujia mahdollistaa sen, että eri osa-alueiden osallistujat havaitsevat erilaisia virheitä. Tarkastus on tehokas tapa löytää virheitä, mutta vähiten tehokas järjestelyiden kannalta. Tarkastukset vaativat huomattavasti resursseja, sillä niihin osallistuu useita henkilöitä ja sopivan ajankohdan löytäminen voi olla haasteellista. Jokaisen osallistujan on myös varattava aikaa ennen virallista tapaamista ehtiäkseen käydä läpi katselmoitava ohjelmakoodi etukäteen. Katselmointitilaisuudet voivat nopeasti muuttua melko kankeiksi ja katselmoinnin vetäjällä on suuri merkitys, että keskustelu pysyy riittävällä tasolla, eikä mene liian yksityiskohtaiseksi tai jää liian ylimalkaiseksi. (McConnell 2004, 488-489; Wiegers 2002, 34)

4.1.2 Ryhmäkatselmointi

Ryhmäkatselmointi on hieman kevennetty versio tarkastuksesta. Se on yleensä etukäteen suunniteltu, mutta vähemmän virallinen. Ryhmäkatselmoinnissa käytetään tavallisesti vastaavia rooleja kuin tarkastuksessa, mutta esimerkiksi erillistä lukijaa ei tarvita. Muutoin katselmoinnin rakenne ja siinä eteneminen ovat pitkälti yhteneväiset tarkastuksen kanssa. Merkittävimpänä erona tarkastukseen verrattuna on, että ryhmäkatselmointi voi edetä myös tekijän johdolla. Tekijä voi toimia oman roolinsa lisäksi myös katselmoinnin vetäjänä. Samankaltaisuuksistaan johtuen myös ryhmäkatselmoinnin ja tarkastuksen hyödyt ja haasteet ovat pitkälti samat. Etuna tarkastukseen verrattuna on kuitenkin tilaisuuden vapaamuotoisuus, joka osaltaan pienentää siitä aiheutuvia kustannuksia. Ryhmäkatselmoinnissa havaitaan kuitenkin usein vähemmän virheitä verrattuna tarkastuksiin, mikä osittain johtuu sen vapaamuotoisuudesta. Ryhmäkatselmointi ei välttämättä etene prosessina yhtä määrätietoisesti kuin tarkastus. Lisäksi roolijako on väljempi ja vapaampi. (van Veenendaal, 1999; Wiegers 2002, 35)

4.1.3 Läpikäynti

Läpikäynti on vapaamuotoinen tapa katselmoida ohjelmakoodia ryhmässä. Läpikäyntiä varten järjestetään tapaaminen, johon osallistuu useampi henkilö. Osallistujilla ei ole ennalta määrättyjä rooleja, vaan katselmointi etenee vapaasti tekijän esittäessä työtään. Läpikäynnin tukena voi olla tarkistuslista, mutta se ei ole välttämätöntä. Läpikäynnin hyödyt ja haitat ovat samankaltaisia kuin ryhmäkatselmoinnissa ja tarkastuksessa ja vaihtelevatkin hyvin pitkälti

sen mukaan, kuinka muodollisena läpikäynti järjestetään. Mitä vapaamuotoisempana läpikäynnin järjestää, sitä suuremmaksi riski keskustelun rönsyilyyn, hallitsemattomiin erimielisyyksiin sekä epäolennaisiin asioihin keskittymiseen kasvaa. Läpikäynnissä hyödynnettävä tiukempi etenemiskaava ja erillinen vetäjä tuovat muodollisuutta ja näin ollen yhteneväisempiä hyötyjä tarkastuksen ja ryhmäkatselmoinnin kanssa. Toisaalta vapaampi keskustelu ja avoimempi tilaisuus mahdollistavat taas otollisemman ympäristön oppia toisilta ja jakaa tietoa kehittäjätiimin kesken laajemmin. (ISTQB, 33; Wiegers 2002, 36-38)

4.1.4 Pariohjelmointi

Pariohjelmointi tarkoittaa nimensä mukaisesti ohjelmointia parin kanssa eli kaksi kehittäjää työskentelee yhdessä saman tehtävän toteuttamiseksi. Pari työskentelee yhden tietokoneen ääressä, minkä vuoksi menetelmässä on vakioidut roolit. Toinen kehittäjistä toimii ajajana ja toinen tarkkailijana. Ajajan roolissa olevalla kehittäjällä on käytössään näppäimistö sekä hiiri ja hänen tehtävänä on kirjoittaa ohjelmakoodia. Tarkkailijan roolissa olevan kehittäjän tehtäviin kuuluu ohjelmakoodin laadun tarkkailu eli virheiden ja puutteiden havaitseminen sekä varmistaa, että tuotos noudattaa ohjelmointikäytäntöjä. Näin ollen ajaja voi keskittyä tehtävän yksittäisiin osiin ja niiden suorittamiseen, kun tarkkailijan vastuulla on kokonaisuuden hallitseminen. Ongelmatilanteissa kehittäjät yrittävät yhdessä löytää parhaan mahdollisen ratkaisun. Rooleja on mahdollista vaihtaa kesken tehtävän ja roolien vaihtoa on tarkoituskin tehdä säännöllisesti työn edetessä. Myös pareja on hyvä vaihtaa mahdollisuuksien mukaan. Nykyiset ohjelmointiympäristöt mahdollistavat pariohjelmoinnin myös etänä. Tällöin kehittäjien ei tarvitse olla saman tietokoneen ääressä, vaan he voivat pariohjelmoida sijainnista huolimatta. Haasteena vanhanmalliseen pariohjelmointiin verrattuna on rooleissa pysyminen ja kommunikointi. (Wiegers 2002, 38; Williams 2000, 89-90)

Yksi pariohjelmoinnin huomattavimmista eduista on se, että työtehtävän parissa on tiiviisti kaksi kehittäjää, joilla molemmilla on oletusarvoisesti syvä näkemys tuotoksesta, toteutustavasta ja mahdollisista haasteista. Tämä luo kuitenkin myös yhden pariohjelmoinnin suurimmista haasteista koodikatselmoinnin kannalta. Koska molemmat kehittäjät omaavat paljon tietoa tehtävästä työstä ja sen toteutustavoista sekä haasteista, on katselmointi vaikeampaa toteuttaa uudesta näkökulmasta hieman etäämmältä. Tällöin molemmat voivat tulla sokeiksi yhdessä tehdyille virheille. Lisäksi pariohjelmoinnin haasteena ovat resurssit ja erityisesti resurssien jakaminen. Ensisijaisena ajatuksena pariohjelmoinnissa on virheiden havaitseminen kehitysvaiheessa, jolloin vältytään korjaustarpeilta myöhemmässä vaiheessa, jolloin muutosten tekeminen ohjelmakoodiin on kalliimpaa. (Wiegers 2002, 38-39; Williams 2000, 89-90)

Myös kehittäjien osaamistasolla on merkitystä pariohjelmoinnin kustannustehokkuuteen. Usein kehittäjätiimissä on kehittäjiä, joilla on eriasteista kokemusta itse työstä, ohjelmoinnista ylipäätään sekä siinä käytettävistä työkaluista. Näin ollen pareja voidaan muodostaa eri-

laisin kombinaatioin, kuten esimerkiksi seniori-seniori, seniori-juniori tai juniori-juniori -pareja. Lyhyesti kuvattuna seniorilla tarkoitetaan kehittäjää, jolla on pitkä kokemus kehitystyöstä ja siinä käytettävistä työkaluista, kun taas juniori on edellä mainitun vastakohta. Jako senioreihin ja junioreihin on yleisesti käytetty, vaikka se on karkea ja todellisuudessa kehittäjän osaamistaso voi vaihdella jopa yksittäisten tehtävien mukaan. Pariohjelmoinnilla voidaan löytää hyötyjä kaikissa edellä mainituissa variaatoissa. Esimerkiksi seniori-seniori -parit ja erityisesti seniori-juniori -parit hyötyvät tiedon jakamisesta ja pariohjelmointi voidaan nähdä jopa eräänlaisena opetus- tai perehdytystilaisuutena. Tutkimuksen mukaan juniori-juniori -parien on mahdollista toteuttaa tehtävä yhtä tehokkaasti saavuttaen yhtä laadukkaan lopputuloksen kuin yksin seniori-kehittäjän suorittamana (Arisholm, Dybå, Hannay, Sjøberg 2009, 1120-1121). Toisessa tutkimuksessa todetaan, että pariohjelmointi parantaa ohjelmakoodin laatua, eikä lisää kehittäjien kehitystyöhön käyttämää aikaa tai lisää sitä ainoastaan vähän (Williams 2000, 89-90). (Arisholm, Dybå, Hannay & Sjøberg, 2009 1120-1121; Wiegers 2002, 38-39; Williams 2000, 89-90)

4.1.5 Pull request

Pull requestejä on mahdollista käyttää silloin, kun kehittäjätiimin käytössä on versionhallinta. Versionhallinnan avulla seurataan muutoksia, joita siellä oleviin tiedostoihin tehdään eli jokainen muutos ohjelmakoodin näkyy versionhallinnassa, josta on mahdollista palauttaa aiempia versioita kyseisistä tiedostoista. Pull request tarkoittaa pyyntöä yhdistää tehdyt muutokset versionhallinnan päähaaraan. Näin varmistetaan, että päähaarassa ei ole ohjelmakoodia, jota kukaan muu kuin sen tekijä ei olisi nähnyt tai käynyt läpi. Käytännössä pull requestit toimivat niin, että kehittäjä luo sen hetkisestä päähaarasta oman haaran, johon muutokset tehdään. Kun muutokset ovat valmiit, kehittäjän omasta haarasta luodaan pull request, jolla pyydetään yhdistämään kehittäjän oma haara päähaaraan. On suositeltavaa nimetä oma haara esimerkiksi tehtävänumerolla, jota muutoksilla toteutetaan. Tämä helpottaa niin tarkastajan työtä kuin myös myöhempää seurantaa ja ylläpitotyötä. Tehtävänumeron avulla tarkastaja voi ensin käydä läpi tehtävänannon, joka on annettu kehittäjälle toteutettavaksi, ja sen jälkeen arvioida kehittäjän toteuttamaa ratkaisua. Pull requestissä näytetään vertailu oman sekä päähaaran välillä niin, että tehdyt muutokset ja lisäykset näkyvät vihreänä ja vanhat ohjelmakoodit ja poistetut osat punaisena. Versionhallinnan asetuksista riippuen pull requestistä lähtee sähköposti henkilölle, jolle se on osoitettu. Pull requestit voidaan osoittaa myös muun muassa ryhmälle. Sähköposti sisältää linkin versionhallintaan, jossa muutokset on korostettu värein ja näytetään vain ne kohdat ohjelmakoodista, joihin on tehty muutoksia. Tarkastaja voi lisätä kommentteja ja ehdotuksia sekä joko hyväksyä muutokset suoraan tai hyväksyä ne ehdotusten kera. Tarkastaja voi myös jättää muutokset hyväksymättä, kunnes tekijä on tehnyt toivotut korjaukset. Tekijällä on myös mahdollisuus kirjoittaa kommentteja ja selityksiä tarkastajalle. Tällä tavoin mahdollistetaan keskustelu kehittäjän ja katselmoijan välillä niin, että keskustelu

on nähtävissä myös myöhemmin, jos esimerkiksi valittua ratkaisua kyseenalaistetaan myöhemmin. Kun pull request hyväksytään ja merkitään valmiiksi, yhdistyvät muutokset päähaaraan. (Github)

Pull request on vaivaton toteuttaa kehitystyön ohessa kaikissa kehitysprosessin vaiheissa, sillä kehittäjää vaaditaan tekemään ainoastaan oma haara sen hetkisestä päähaarasta, jonka nimessä viitataan esimerkiksi tehtävänumeroon, jota työ koskee. Pull requestit koodikatselmointityyppinä eivät vaadi kehittäjältä lainkaan lisäresursseja ja mahdollistavat laajemman kohdeyleisön tavoittamisen helpommin, sillä pull requestin voi osoittaa kenelle tahansa organisaatiosta. Haittapuolena voidaan nähdä pull requestin epämuodollisuudesta aiheutuva priorisoinnin puute. Vaikka pull requestin osoittaisi tusinalle kehittäjiä, voi olla, ettei kukaan heistä ehdi katselmoida sitä. Lisäksi vaaditaan hyvää kommunikointia, tehokkaita kommunikointitapoja sekä yhteistä ohjeistusta pull requestien läpikäyntiin, joiden avulla voidaan välttyä tilanteilta, joissa useampi kehittäjä katselmoi samaa pull requestiä tai muut luulevat, että joku toinen katselmoi sitä ja lopulta kukaan ei katselmoi sitä. Pull requestit vaativat myös itsekuria sekä tekijöiltä, että erityisesti katselmoijilta, jotta tehdyt muutokset tulevat varmasti katselmoitua. Koska yleensä pull requestien läpikäyntiä varten ei järjestetä erillisiä tapaamisia, myös tämän vuoksi kommunikoinnin on toimittava kehittäjien kesken. Sähköpostitse tulevat ilmoitukset voivat myös hukkua muun sähköpostin sekaan, mikä saattaa venyttää kehitysprosessia turhan pitkäksi. Pull requestit eivät tarjoa mahdollisuutta esimiehille ja johdolle seurata, että koodi tulee katselmoitua asianmukaisesti. (Github; Wiegers 2002, 40-41)

4.1.6 Tarkastus parin kanssa

Tarkastus parin kanssa tarkoittaa tilannetta, jossa kehittäjä esittelee kirjoittamansa ohjelmakoodin toiselle kehittäjälle, jonka on pyytänyt työpisteensä ääreen. Tarkastus parin kanssa on suomennos englanninkielisestä termistä peer deskcheck. Yleensä kehittäjä, joka on kirjoittanut katselmoitavan ohjelmakoodin, ohjaa tilannetta ja hänen vastuulla on muistaa näyttää kaikki kohdat, joihin muutoksia tehtiin, mikä aiheuttaa suuren riskin, että jotain jää katselmoimatta. Tekijän tulee esitellä muutokset katselmoijalle sopivalla vauhdilla niin, että katselmoija ehtii havaita virheet. Parin kanssa tehtävästä tarkastuksesta on mahdollista saada hieman muodollisempi, jos katselmoijalla on käytettävissään tarkistuslista. (Wiegers 2002, 39)

Parin kanssa tehtävän tarkastuksen etuna on sen vaivattomuus. Se ei vaadi erityisiä järjestelyjä ja mahdollistaa ongelmien ratkaisun yhdessä tekijän ja katselmoijan kesken. Myös tiedon ja osaamisen jakaminen on otollista, sillä parin kanssa tehtävä tarkastus tapahtuu parina ja mahdollistaa tällöin hyvät puitteet hyödylliselle keskustelulle. Kommunikointi on myös helpompaa, kun molemmat ovat fyysisesti läsnä. Huonona puolena voidaan nähdä strukturoimattomuus. Katselmoijalta vaaditaan hyvää keskittymiskykyä käydä läpi ohjelmakoodia toisen esittämänä niin, että virheet havaitaan. Parin kanssa tehtävässä tarkastuksessa ei yleensä tehdä erillistä dokumentaatiota, minkä vuoksi sen hyödyllisyyttä on vaikea mitata. Ei ole

myöskään mahdollista saada varmuutta, korjattiinko havaitut virheet, jos katselmoija poistuu työpisteeltä heti läpikäynnin jälkeen. Parin kanssa tehtävää tarkastusta voidaan pitää hyvänä ensi askeleena koodikatselmointikulttuurin muodostamiseen. (Wiegers 2002, 39-40)

4.1.7 Ad hoc -katselmointi

Jokaiselle kehittäjälle tulee vastaan tilanteita, jossa ohjelmakoodissa oleva virhe tai vastaantuleva ongelma on sellainen, että hän kokee parhaimmaksi käydä pyytämässä kollegalta apua virheen korjaamiseen tai ongelman ratkaisemiseen. Ad hoc -katselmoinnissa tekijä puuttuu havaitsemaansa virheeseen, mutta tarvitsee joko apua sen korjaamiseen tai varmistaa, että hänen korjausehdotus on paras mahdollinen. Ad hoc -katselmointi tapahtuu siis aina tekijän aloitteesta. Ad hoc -katselmoinnin hyötyinä ovat tiedon jakaminen ja toteuttamisen vaivattomuus. Siinä ei kuitenkaan ole minkäänlaista dokumentaatiota ja sellaista olisi lähes mahdotonta edes käyttää, sillä tilanteet tulevat yllättäen ja vaihtelevat laajasti kokonsa ja aiheensa puolesta. Ad hoc -katselmoinnissa haittana on myös se, ettei katselmoija ehdi valmistautua siihen lainkaan, eikä sitä toteuteta erikseen sovittuna aikana. (Wiegers 2002, 41)

4.2 Resurssit

Aika on yksi merkittävimmistä resursseista, joita koodikatselmoinnin toteuttamiseen vaaditaan. Koodikatselmointi ei kuitenkaan välttämättä lisää kehittäjien työaikaa tai -määrää niin kuin ensin saattaisi ajatella. Kyseessä on pikemminkin kehittäjien työajan uudelleenorganisointia. Kehitysprosessiin kuluva aika lisääntyy, kun koodikatselmointi lisätään osaksi koko prosessia, mutta siihen käytetty aika otetaan aiemmin ylimääräisiin korjauksiin kuluneesta työajasta, eikä työajan kokonaisuutena pitäisi kasvaa lainkaan. Wiegers (2002, s. 12) kuvaa ajankäyttöä osuvasti ”If you don’t have time to do it right, when will you have time to do it over” eli ”jos sinulla ei ole aikaa tehdä sitä oikein, milloin sinulla on aikaa tehdä se uudelleen”. Jos koodikatselmointiprosessi on organisaatiolle uusi, ovat kustannukset todennäköisesti aluksi hieman normaalia korkeammat. Pitkällä tähtäimellä kustannukset kuitenkin tasoittuvat entiselle tasolleen tai jopa alle sen, kun kehittäjien työaika, joka on aiemmin kulunut virheiden korjaamiseen, kuluu muiden tekemän ohjelmakoodin katselmointiin. Kun koodikatselmointiprosessi on vakiintunut organisaatiossa, myös virheiden määrä vähennee, mikä tarkoittaa lisää työaikaa muille töille. (Elliott 2016; Greiler 2019; Wiegers 2002, 146-147)

Koodikatselmointityypeistä muodollinen tarkastus vaatii eniten resursseja jo pelkästään siihen osallistuvien henkilöiden määrän vuoksi. Lisäksi se on prosessina raskain, sillä se sisältää nimensä mukaisesti eniten muodollisuuksia eli dokumentaatioita ja tehtäviä. Keveimmillään koodikatselmointi voi olla sitä, että kehittäjä pyytää kollegaa tietokoneensa äärelle katsomaan tilannetta, jonka kanssa hän on sillä hetkellä jumissa. Koodikatselmointiin vaadittujen resurssien määrään vaikuttaa monenlaiset tekijät. Muun muassa kohteen kriittisyydellä liike-

toimintaan on merkitystä, sillä mitä kriittisempi komponentti on kyseessä, sitä varmempia halutaan olla, että virheet ohjelmakoodissa on minimoitu. Henkilöstövaihtuvuus vaikuttaa myös tarvittaviin resursseihin, sillä uusille henkilöille olemassa oleva lähdekoodi on vierasta ja sen läpikäyminen vie odotetusti enemmän aikaa kuin henkilöllä, joka on työskennellyt sen parissa useita vuosikymmeniä. (Wiegers 2002, 31-32)

4.3 Hyödyt

Koodikatselmoinnilla on monia etuja. Ei ole yhtä oikeaa tapaa tehdä koodikatselmointia ja siksi usein suurimmat hyödyt saadaan silloin, kun koodikatselmointi on suunniteltu hyvin, siihen käytetään riittävästi resursseja ja se on luonnollinen osa kehitysprosessia. Koodikatselmoinnin hyödyistä puhuttaessa mainitaan usein ensimmäisenä ohjelmakoodin laatu, mikä onkin yksi merkittävimmistä mukanaan tuomista hyödyistä. Ohjelmakoodin laadun parantamisen tavoitteluun voi olla useita syitä, mutta esimerkeissä esille nousevat muun muassa virheiden vähentäminen ja niiden havaitseminen aikaisemmassa vaiheessa, korjaustarpeiden vähentäminen tuotannossa olevaan ohjelmakoodiin sekä ymmärrettävämmän ohjelmakoodin kirjoittaminen. Miten koodikatselmointi parantaa ohjelmakoodin laatua? Ensinnäkin kehittäjä tulee nopeasti sokeaksi omille virheilleen, eikä välttämättä havaitse kaikkia virheitä kirjoittamassaan ohjelmakoodissa. Syntaksivirheet on helppo huomata, sillä ne eivät mene kääntäjästä läpi, mutta muunlaisia virheitä, kuten kirjoitusvirheitä, virheellistä toiminnallisuutta tai väärinymmärrettyä tehtävää ei tekijän ole itse yhtä helppo huomata. Koodikatselmointi pakottaa muidenkin kuin tekijän itse käymään läpi ohjelmakoodia, minkä ansiosta virheitä havaitaan todennäköisemmin ja näin ollen myös ohjelmakoodin laatu paranee. Sen lisäksi koodikatselmointi edesauttaa kehittäjää kiinnittämään entistä enemmän huomiota valitsemiinsa ratkaisuihin ja pohtimaan niitä tarkemmin, sillä koodikatselmoinnin yhteydessä voi tulla tilanne, jossa valittuja ratkaisuja haluaa perustella muille kehittäjille, jotka toimivat katselmoijina. (Bodner 2018; McConnell 2004, 481; Torlak 2015)

Koodikatselmointi tarjoaa tilaisuuden jakaa osaamista, tietoa ja taitoja kehittäjien välillä. Se on vahvasti vuorovaikutteista, minkä vuoksi kehittäjät pääsevät tutustumaan tarkemmin toistensa kirjoittamaan ohjelmakoodin, tyyliin kirjoittaa ohjelmakoodia, käyttämiin tekniikoihin ja jopa toistensa ajattelutapaan. Tällöin tulee väistämättä esiin eroja omaan tekemiseen ja niitä vertailemalla lopputulos on opettavainen kaikille osapuolille, kun niitä on mahdollista käydä yhdessä läpi. Kehittäjien tuleekin tiedostaa, että koodikatselmointia harjoitetaan myös tiedon jakamisen vuoksi. Sen tiedostamalla he voivat omaksua uusia oppeja toisilta. Kun tietoja ja taitoja jaetaan enemmän tiimin kesken, myös ohjelmointityyli voi tulla yhtenäisemmäksi. Tilanteet, joissa on yhdessä katselmoijan kanssa mietitty parasta mahdollista ratkaisua ongelmaan, jäävät kummankin kehittäjän mieleen ja voivat ohjata heitä valinnoissa, joita he tulevaisuudessa tekevät. Täysin yhtenäistä ohjelmointityylistä ei tule, eikä se ole tarkoituskaan, mutta tällä tavoin on mahdollista päästä eroon esimerkiksi vanhoista tavoista, kun asian

on kertaalleen oppinut tekemään yhdellä tavalla, joka ei välttämättä ole nykyään enää tehokkain tapa ratkaista asia. (Bodner 2018; Zihler 2017)

Tiimin kehittäjät voivat myös vaihtua henkilöstömuutoksien vuoksi ja mahdollisesti käytettävien konsulttien takia. Tällöin katselmointia voidaan hyödyntää osana työtehtäviin perehdytystä. Olemassa olevat ohjelmakoodit tulevat luonnollisemmin tutuksi, kun kehittäjätiimin uudet jäsenet käyvät läpi ohjelmakoodimuutoksia ja tehtäviä. Yhteinen vastuuntunto ohjelmakoodista kasvaa koodikatselmoinnin avulla, sillä jo pelkästään se, että muut kehittäjät ovat nähneet toisen tekemän ohjelmakoodin aiemmin, madaltaa kynnystä tehdä muutoksia siihen tarpeen tullen. Yksikään koodinpätkä ei ole olemassa ilman, että vähintään yksi muista kehittäjistä olisi katselmoinut sen eli yksikään rivi ohjelmakoodia ei ole täysin tuntematonta kaikille. Tällöin saavutetaan askel lähemmäksi ihannetilannetta, jolloin yksikään työtehtävä ei olisi vain yhden kehittäjän vastuulla, vaan esimerkiksi projektin eri tehtäviä voitaisiin jakaa eri kehittäjille aina tilanteen mukaan. Myöskään poissaoloilla ei olisi tällöin niin suurta vaikutusta kehitys- tai ylläpitotyöhön. Yhteisvastuu tehtävistä parantaa myös kehittäjätiimin kykyä arvioida työmääriä ja sen myötä myös helpottaa töiden suunnittelua. Erityisesti ketterässä kehityksessä käytetään yksittäisten työtehtävien työmäärien arvioimista, mikä on entistä helpompaa, mitä paremmin tuntee muiden tiimiläisten työskentelytavat, vahvuudet ja heikkoudet. Näin myös suunnittelu on helpompaa ja johdolle on mahdollista antaa tarkempaa tietoa esimerkiksi työn suorittamiseen tarvittavista resursseista. (Bodner 2018; McCon-nell 2004, 482)

4.4 Haasteet

Koodikatselmoinnin haasteita voidaan tarkastella eri näkökulmista. Esimerkiksi, miksi jokaisessa organisaatiossa ei tehdä koodikatselmointia tai mitkä asiat estävät koodikatselmoinnin onnistumisen? Edellä on mainittu kaksi eri näkökulmaa. Haasteet, jotka estävät koodikatselmoinnin tekemisen kokonaan ja haasteet, jotka estävät onnistuneen koodikatselmoinnin. On selvää, että koodikatselmointia kuitenkin kannattaa tehdä, joten miksi haasteet käyvät joissain tapauksissa liian suuriksi?

Kehittäjien asenteella koodikatselmointia kohtaan on suuri vaikutus. Asenteet vaikuttavat siihen, katselmoidaanko koodia lainkaan, tai jos katselmoidaan, kuinka hyvin siinä onnistutaan. Sillä on siis merkitystä, mitä kehittäjät koodikatselmoinnista ajattelevat ja negatiiviset asenteet voivat näkyä esimerkiksi ennakkoluuloina ja merkityksen vähättelynä. Negatiiviset asenteet ja mielipiteet ovat kuitenkin vain yksi haasteista. Organisaation suhtautumisella voi olla jopa suurempi merkitys sille, kuinka koodikatselmointi onnistuu. Ensinnäkin organisaatio tarjoaa puitteet koodikatselmoinnin toteutukselle. Toisekseen organisaation suhtautumisella on vaikutus yleiseen ilmapiiriin. Molemmat edellä mainituista vaikuttavat tietenkin myös kehittäjien asenteisiin ja mielipiteisiin. (Wiegers 2002, 20-22)

Asenteiden, mielipiteiden ja suhtautumisen lisäksi koodikatselmoinnin sudenkuoppia tiimin näkökulmasta ovat strukturoimaton toiminta, tietämättömyys, resurssien puute sekä subjektiivisuus. Strukturoimaton toiminta aiheuttaa nopeasti sovituista toimenpiteistä lipsumista. Kun koodikatselmointiprosessi puuttuu, kehittäjiltä puuttuu runko, jonka mukaan pitäisi toimia, jolloin työvaihe saattaa jäädä toteutumatta puolihuolimattomasti. Tietämättömyys tai ymmärryksen puute koodikatselmoinnin tarkoituksesta, yhteisistä tavoitteista ja toimintatavoista estää kehittäjiä toimimaan yhdessä tiiminä ja aiheuttaa eroja asenteisiin ja suhtautumiseen. Kehittäjätiimillä tulee olla riittävästi resursseja suorittaakseen riittävällä tasolla katselmointia. Resurssien tarpeeseen vaikuttaa kehityskohteen koko, kehittäjätiimin jäsenet ja sen koko sekä aikataulut. Resursseista päättää kuitenkin aina organisaatio. Subjektiivisuus on ikuinen haaste tiimeille, sillä jokainen jäsen on aina yksilö, eivätkä toimintatavat ole koskaan täysin yhdenmukaisia. Haasteena on päästä tasolle, jossa koodikatselmointi suoritetaan tavoitteiden mukaisesti huolimatta siitä, kuka kehittäjistä toimii katselmoijana. Tiimiltä vaaditaan myös keskinäistä luottamusta, kunnioitusta sekä avointa ilmapiiriä. (Bacchelli & Bird 2013; Wiegers 2002, 16-20 & 22-25)

Käytännöllisemmästä näkökulmasta koodikatselmoinnin haasteita ovat katselmoitavan ohjelmakoodin määrä, muutoksen ymmärtäminen, hyödyllisen palautteen antaminen sekä oikeisiin asioihin keskittyminen. Katselmoijan työlle on suuri merkitys sillä, onko ohjelmakoodia läpikäytävänä kymmeniä vai satoja rivejä. Mitä suurempi kokonaisuus on katselmoitavana, sitä helpommin virheet ja kehityskohteet jäävät huomaamatta. Lisäksi muutoksen ymmärtäminen voi olla helpompaa, kun käsiteltävä kohde ei ole liian suuri. Toisaalta kokonaisuuden ymmärtämisestä on merkittävä hyöty katselmoijalle. Sekä katselmoijan, että tekijän tulisi artikuloida selvästi niin suullisesti kuin kirjallisesti. Heillä tulee olla yhteisymmärrys huomioista, jotta esimerkiksi niiden perusteella tehtävät korjaukset tehdään oikein. Katselmoijalta tulevan palautteen tulisi olla hyödyllistä ja työtä edistävää. Kaikenlaista saivartelua tulisi välttää ja pitää keskittyminen asiassa riittävän yleisellä tasolla. Saivartelua tulisi erityisesti välttää, kun kyseessä on muodollinen tarkastus, johon saattaa osallistua useampia kehittäjiä ja tilanne voi helposti ajautua jonkin yhden yksittäisen ratkaisun löytämiseen kokonaisuuden sijaan. (Bacchelli & Bird 2013)

4.5 Miten onnistua koodikatselmoinnissa?

Koodikatselmoinnissa onnistumiseen ei ole yhtä oikeaa tapaa, jonka avulla siinä voi poikkeuksetta onnistua. Koska tavat katselmoida ohjelmakoodia muovautuvat kehittäjätiimien mukaan kullekin sopivaksi, myös keinot onnistua ovat riippuvaisia kustakin tiimistä ja juuri heille sopivista tavoista. Jokaisen organisaation on löydettävä omiin tavoitteisiin ja tapoihin sopiva tapa saada mahdollisimman paljon hyötyä koodikatselmoinnista ja näin ollen onnistua siinä. Koodikatselmoinnissa onnistuminen ei ole myöskään absoluuttinen tila, joka säilyisi loputtomiin sen kerran saavutettuaan. Onnistunut koodikatselmointiprosessi elää kehittäjätiimin mukana. Sitä

pitää aika ajoin kehittää ja tarkastella kriittisesti, erityisesti silloin, kun muihin kehitystyön vaiheisiin tulee muutoksia. Tällaisia muutoksia voivat olla esimerkiksi ohjelmoinnissa käytettävien työvälineiden vaihtuminen, kehitystyön kohteet tai merkittävät henkilöstömuutokset. Myös käytössä olevat muut resurssit, kuten aika, vaikuttaa koodikatselmointiprosessiin ja erityisesti sen odotuksiin ja vaatimuksiin.

Palautteen vastaanottaminen on taito, jota lähes jokaisen on harjoiteltava ja erityisesti sitä, miten saadun palautteen avulla voi kehittyä. Koodikatselmointiprosessin onnistumisen kannalta on äärimmäisen tärkeää, että palautetta ja kritiikkiä osataan antaa ja vastaanottaa rakentavasti. Kehittäjän on helpompi ottaa vastaan kritiikkiä tehdystä työstään, kun luottamus ja arvostus kehittäjien välillä on kohdallaan. Tällöin kehittäjä luottaa toisen arvostelukykyyn. Rakentavaa kritiikkiä annetaan arvostaen toisen ammattitaitoa sekä tehtyä työmäärää. Sanavalinnoilla on suuri merkitys, joten ne tulisi aina valita huolella keskittyen aina katselmoitavaan ohjelmakoodiin. Egottomalla ohjelmoinnilla tarkoitetaan työskentelytapaa, jossa pyritään minimoimaan työn henkilökohtaisuus. Ajatuksena on, että kehittäjät pääsisivät eroon ajatuksesta, että heidän työllään olisi merkitystä omaan ihmisarvoon. Sen ansiosta kehittäjä pystyisi tarkastelemaan tuotostaan etäämmältä ja näin ollen vastaanottamaan paremmin korjauskehotuksia ja kehitysideoita muilta. Äärimmäisenä ajatuksena on, että kehittäjätiimi tekisi kehitystyötä yhdessä ryhmänä ja päätökset tehtäisiin aina yhdessä, jolloin vastuu, onnistumiset ja epäonnistumiset eivät olisi henkilökohtaisia. Kehitystyön tulisi olla hyvin rajoittamatonta, jotta äärimmäiset ajatukset egottomasta ohjelmoinnista toteutuisi, mutta ydinajatus toimii monissa ryhmätöissä, ei ainoastaan ohjelmointityössä. Organisaation johto voi tukea egotonta ohjelmointia rohkaisemalla yhteistyöhön ja avoimeen ilmapiiriin sekä palkitsemalla tiimiä, ei sen yksittäisiä jäseniä. (Greiler 2019 & Wiegers 2002, 13-22)

Prosessiomistajan valitseminen koodikatselmointiprosessille koetaan tuovan lisäarvoa, sillä hänen ensisijaisena vastuunaan on huolehtia, että prosessi kehittyy ja tarvittavat muutokset tulee tehdyksi. Ilman nimettyä prosessiomistajaa on vaarana, että koodikatselmointiprosessi jää sellaiseksi kuin se ensimmäistä kertaa käyttöön otettaessa oli. Koodikatselmointi kehittyy siinä missä muutkin ohjelmistokehityksen osa-alueet eli saavuttaakseen optimaalisen koodikatselmointiprosessin, on sen muututtava esimerkiksi uusien tekniikoiden tai kehittäjien näkemysten mukana. Vastuuta koko prosessin kehityksestä ei ole suositeltavaa asettaa kehittäjätiimille, sillä vastuunkanto ryhmänä on usein epämääräistä ja muiden töiden ohessa jää todennäköisemmin hoitamatta. Vaikka vastuu annettaisiin ryhmälle, usein yksi ryhmän jäsenistä joutuu joka tapauksessa ottamaan ohjat, jotta asioita saadaan etenemään. Suositeltavaa on, että koodikatselmointiprosessin omistaja on esimies, sillä hänellä on vahvemmat yhteydet organisaation muihin päätäntäelimiin turvatakseen koodikatselmointiprosessin jatkuvuuden. Koodikatselmointiprosessin omistajana voi myös toimia pitkän kokemuksen omaava kehittäjä, jolla on kokemusta organisaation toimintatavoista, ja joka on kykenevä johtamaan prosessia

asiantuntijuudellaan. Tärkeää on, että koodikatselmointi-prosessin omistajalla ja kehittäjätiimillä on avoin keskustelusuhde, yhteinen pyrkimys luomaan onnistunut koodikatselmointi sekä kunnioittava asenne kaikkia osapuolia kohtaan. (Wiegers 2002, 143-145)

5 Menetelmät

Tutkimukset jaetaan usein laadullisiin tai määrällisiin, vaikka todellisuudessa molempia tutkimusmenetelmiä on mahdollista käyttää yhtäaikaisesti samassa tutkimuksessa. Kyseessä ei siis ole täysin vastakohdat, vaan tosiaan osittain muistuttavat ja toisistaan osittain erottuvat tutkimusmenetelmät. Määrällisessä tutkimuksessa kohdetta kuvataan ja tulkitaan tilastoista ja numeroista, kun laadullisessa tutkimuksessa kohde kuvataan kokonaisvaltaiseesti ymmärtäen sen laatua, ominaisuuksia ja merkityksiä. (Tuomi & Sarajärvi 2018, 18-25)

Laadulliselle tutkimukselle ei ole yhtä oikeaa määritelmää. Se omaa piirteitä toisesta tutkimusstrategiasta, tapaustutkimuksesta, sillä siinä analysoidaan jotain tapausta, joka pyritään argumentoimaan myös muuten kuin vain määrällisesti. Lisäksi havainnot perustuvat tutkimuksessa ilmenevään tulkintaan. Tiedonkeruu laadullisessa tutkimuksessa tapahtuu usein erilaisin kyselyin tai havainnoimalla. Havainnointi voi olla esimerkiksi tilanteiden tai dokumenttien tulkitsemista. Laadullisessa tutkimuksessa keskitytään aineiston määrän sijasta sen kattavuuteen, jotta siitä tehtävä tulkinta olisi mahdollisimman pätevä. (Tuomi & Sarajärvi 2018, 18-25)

5.1 Tutkimus- ja kehittämismenetelmät

Opinnäytetyön teoreettisen osuuden tutkimusstrategiana oli laadullinen tutkimus. Kehittämiskohdetta tutkittiin vapaasti ja osallistuvasti havainnoimalla siihen vaikuttavia tekijöitä, kuten Handelsbanken .NET-kehittäjätiimin työskentelytapoja sekä ryhmädynamiikkaa. Havainnoinnin tarkoituksena oli ymmärtää, miten koodikatselmointi todellisuudessa hoidetaan, ja eroavatko nykyiset koodikatselmointitavat organisaation vaatimuksista tai yleisestä käsityksestä, miten koodikatselmoinnin annetaan ymmärtää toteutuvan. Havainnointi tapahtui osallistuvana, mutta siinä pyrittiin pitäytymään yliosallistumisesta ja keskittymään todellisen kuvan muodostamiseen niin, että havainnoijan näkökulma oli vain yksi osa kokonaiskuvasta. Havainnoinnin tulokset dokumentoitiin muistiinpanoina, joita hyödynnettiin myöhemmin tutkimuskirjallisuuden läpikäynnissä. Aineiston kokoamiseksi käytettiin sisällön analyysia. Aineiston keräämiseksi myös tutkimuskirjallisuuden läpikäynnissä hyödynnettiin sisällön analyysia, jotta tutkimuskohdetta voitiin kuvailla käsitteellisesti. Sisällön analyysin menetelmällä pyrittiin luomaan yhtenäinen ja selkeä kuvaus tutkimuskohteesta. Havainnoinnin tulokset jaettiin teemoihin aineistolähtöisesti. Teemat luotiin pääosin havainnoinnissa ilmenneiden huomioiden perusteella, mutta osin myös lähdekirjallisuudessa esille tulleiden osa-alueiden mukaan. Teemoja olivat resurssien hallinta, tiimityöskentely, yksilöllinen ohjelmointitapa, yhteinen näke-

mys hyvästä ohjelmointitavasta, oppiminen sekä sääntöjen noudattaminen. Tutkimuskirjallisuudessa kiinnitettiin huomioita lähteiden monipuolisuuteen, eri näkökulmien esilletuomiseen sekä uskottavuuteen. Tutkimuskirjallisuutta valittaessa arvostettiin enemmän lähdekirjallisuuden laatua kuin sen määrää, sillä määrä ei olisi tuonut huomattavasti lisäarvoa tutkimuskohteen teoreettisessa esittämisessä. (Kvalitatiivinen sisällönanalyysi; Tuomi & Sarajärvi 2018, 87-91; Tutkimusstrategiat 2014)

Kehittämismenetelmistä hyödynnettiin pääasiassa SWOT-analyysia. SWOT-analyysissä kohde kuvataan nelikenttäisessä taulukossa, joissa positiiviset asiat eli vahvuudet ja mahdollisuudet kuvataan taulukon vasemmassa sarakkeessa ja negatiiviset asiat eli heikkoudet ja uhat taulukon oikeassa sarakkeessa. Vahvuudet ja heikkoudet kuvaavat kohteen nykytilaa ja sisäisiä asioita. Vastaavasti mahdollisuudet ja uhat kuvaavat tulevaisuutta ja ulkoisia asioita. SWOT-analyysin avulla asiat saatiin esitettyä eri näkökulmista ja mahdollisimman monipuolisesti. SWOT-analyysin avulla pyrittiin saavuttamaan opinnäytetyön tavoite ja antamaan vastaus sen tutkimuskysymyksiin. (Eloranta, Hautala, Kinos & Salonen 2017, 90)

5.2 Validiteetti ja reliabiliteetti

Tutkimuksessa käytettyjen tutkimusmenetelmien luotettavuutta tarkastellaan kahdesta eri näkökulmasta, validiteetti ja reliabiliteetti. Validiteetti määrittyy sen mukaan, onko tutkimuksessa tutkittu sitä, mitä siinä on ollut tarkoitus tutkia. Reliabiliteetti määrittyy sen mukaan, kuinka toistettavia tutkimustulokset ovat. Validiteetin ja reliabiliteetin merkitys määrällisessä tutkimuksessa on suurempi, mutta myös laadullisen tutkimuksen menetelmiä on mahdollista tarkastella validiteetin ja reliabiliteetin näkökulmasta.

Kehittämistyötä varten kerätyn aineiston valinnassa kiinnitettiin huomiota monipuolisuuteen, laatuun ja luotettavuuteen. Havainnoinnin ja sisältöanalyysin avulla aineistosta pyrittiin nostamaan esille olennaiset asiat kattavasti, jotta tutkimuskohde saatiin esitettyä mahdollisimman todenmukaisesti. Opinnäytetyö on pyritty tekemään johdonmukaiseksi ja tiedonkeruussa huomioitiin hyvä tutkimustapa ja käytetyt lähteet on merkitty asianmukaisesti. Opinnäytteessä julkaisemattomia lähteitä käytettiin kehittämiskohteelle asetettujen tavoitteiden, tarpeiden ja lähtökohtien kuvaamiseen, sillä julkisia lähteitä sitä varten ei ollut saatavilla. Kehittämiskohteen tavoitteilla ja lähtökohdilla on vaikutusta tutkimustuloksiin, mutta ei tutkimustulosten toistettavuuteen eli vaikutusta opinnäytteen reliabiliteettiin ei ole. Opinnäytetyö on tutkimuksellinen kehitystyö, jossa tekijä tekee sekä tutkimukseen, että kehittämiseen liittyvät työt, mikä voi lisätä tekijän puolueellisuutta. Opinnäytteessä tekijän pyrkimyksenä on ollut tarkastella aihetta mahdollisimman objektiivisesti. Subjektiivisuutta saattaa kuitenkin esiintyä kehittämistyössä, sillä opinnäytetyön tekijä kuuluu kehittämistyössä osallisena olevaan .NET-kehittäjätiimiin. (Tuomi & Sarajärvi 2018, 110-111 & 118-125)

6 Koodikatselmointisuunnitelman kehittäminen

Kehittämistyö aloitettiin selvittämällä Handelsbankenin nykyisiä koodikatselmointitapoja seuraten havainnoimalla, miten kukin tiimissä hoitaa sitä, käymällä läpi konsernin sisäisiä ohjeistuksia ja vertailemalla niitä keskenään sekä tutkimalla, saavutetaanko nykyisillä toimintatavoilla vaadittu taso koodikatselmoinnin suhteen. Näin ollen saatiin luotua myös kuva siitä, kuinka yhtenäisiä kehittäjien toimintatavat koodikatselmoinnin suhteen ovat olleet. Tämän lisäksi selvitettiin, minkälaisia työvälineitä kehittäjillä on ollut käytössä ja minkälaisia työvälineitä kehittäjien olisi mahdollista hyödyntää konsernin omien ohjeiden ja suositusten perusteella. Konsernitasolla asetettuja vaatimuksia ja ohjeistuksia koodikatselmoinnille selvitettiin käymällä läpi niihin liittyvät dokumentaatiot. Kyseiset dokumentaatiot on tarkoitettu sisäiseen käyttöön, minkä vuoksi niissä esitettyjä kohtia ei tarkemmin avattu tässä opinnäytetyössä. Vaatimusten ja ohjeistusten pohjalta luotiin kuitenkin suuntaviivat tutkimukselle ja niiden perusteella selvitettiin erilaisia toimintamalleja, joiden avulla ohjelmakoodin laatu paranisi, konsernitason vaatimukset täyttyisivät, minimoitaisiin työtehtävien henkilöityminen, mahdollistettaisiin kanavat, joissa voidaan jakaa tietoja ja osaamista sekä luotaisiin .NET-kehittäjätiimille yhtenäinen tapa toimia.

Havainnoinnin tuloksista oli mahdollista tehdä päätelmiä, että kehittäjillä oli jokseenkin erilainen, omanlainen näkemys koodikatselmoinnin merkityksestä ja toteutuksesta. Osa koki koodikatselmoinnin enemmän työn lopputuloksen teknisenä arvosteluna ja pakollisena työvaiheena, eikä niin paljon nähnyt sitä tilaisuutena jakaa osaamista ja kehittää omaa osaamistaan. Toisille koodikatselmointi oli enemmänkin opetustilaisuus, jossa on mahdollista ammentaa osaamista muilta kehittäjiltä. Erityisesti niille, jotka olivat työskennelleet vasta vähemmän aikaa kehittäjinä, koodikatselmoinnin tuoma mahdollisuus oppia toisilta korostui. Yhteisiä, selkeitä toimintatapoja katselmoida ei ollut, vaan pikemmin sitä hoidettiin aina tilanteen mukaan. Tämä aiheutti tilanteita, joissa kehittäjillä kului työaikaa ensin miettimiseen ja sopimiseen, miten katselmointi suoritetaan. Koodikatselmointia tutkittaessa nousi esille, että mielipiteisiin koodikatselmoinnista voidaan vaikuttaa esimerkiksi tarjoamalla kehittäjille koulutusta koodikatselmoinnista, jotta jokaisella olisi riittävä tuntemus siihen liittyvistä asioista, mahdollisuuksista ja uhista. Nämä seikat huomioiden koodikatselmointisuunnitelmaa lähdettiin kehittämään pohjaksi yhteistä koodikatselmointiprosessin suunnittelua varten.

Opinnäytetyön teoreettisessa osuudessa ilmeni ja korostui kehittäjätiimin yhteisymmärrys koodikatselmoinnin onnistumiseksi, minkä vuoksi koodikatselmointisuunnitelmassa ei sanella, mitä toimia tehdään, vaan annetaan runko, jonka pohjalta kehittäjätiimi ja organisaation johto määrittelee Handelsbankenin .NET-kehittäjätiimille sopivan koodikatselmointiprosessin. Suunnitelman perustana käytettiin Handelsbankenissa entuudestaan käytössä olevia työvälineitä, toimintatapoja ja vaatimuksia eli se täyttää niiltä osin sille asetetut kriteerit. Ennalta kokonaan päätetyn koodikatselmointisuunnitelman implementointia .NET-kehittäjätiimin

käyttöön kannatti välttää siksi, että yhdessä sopiminen sitouttaa noudattamaan suunnitelmaa paremmin kuin ylhäältä alas ojennettua tapaa, jonka toimivuuteen kehittäjillä ei ole ollut mahdollisuutta vaikuttaa.

SWOT-analyysi mahdollisti koodikatselmointityyppien vertailun, jonka perusteella voitiin valita Handelsbankenin .NET-kehittäjätiimille sopivimmat koodikatselmointitavat. Tein SWOT-analyysit opinnäytetyön teoreettisessa osiossa esille nousseiden tietojen perusteella. Kuviossa 6 esitetään tarkastuksen SWOT-analyysi, josta kävi ilmi se, miten tarkoin ennakkoon suunniteltu katselmointitilaisuus toi paljon vahvuuksia, mutta kasvatti myös heikkouksien määrää. Tarkastus sopii erityisesti suurempien kokonaisuuksien katselmointiin, esimerkiksi kokonaisten järjestelmien tai prosessien tarkasteluun ja virheiden sekä puutteiden etsintään.

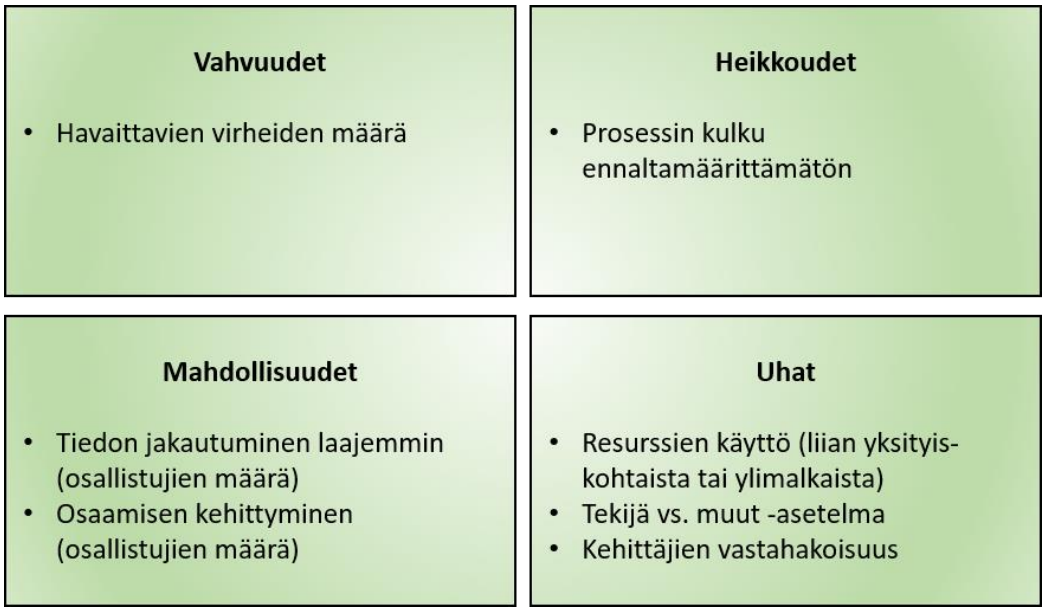
Kuvio 6: Tarkastuksen SWOT-analyysi

Ryhmäkatselmoinnin SWOT-analyysi esitetty kuviossa 7. Ryhmäkatselmoinnin vahvuuksiksi nousivat havaittavien virheiden määrä, pitkälle suunniteltu katselmointiprosessi, dokumentaatio sekä virheiden havaitsemisen seuranta. Koska ryhmäkatselmoinnissa katselmointitilaisuus on suunniteltu etukäteen, osallistujien roolit ovat selkeät ja kaikki tarvittavat asiat käydään varmasti läpi. Dokumentaation ansiosta jatkotoimenpiteet olivat selkeät ja niitä on helpompi seurata. Myös virheiden havaitsemista on mahdollista seurata esimiehen ja johdon toimesta, minkä ansiosta prosessia on mahdollista kehittää. Heikkouksina ovat vaaditut resurssit sekä toteutuksen työläys. Ryhmäkatselmointi vaatii vähintään neljä osallistujaa, joiden tulee ehtiä valmistautua ennen tilaisuuteen osallistumista. Myös tilaisuuden järjestäminen vaatii huomattavia ponnisteluja esimerkiksi yhteisen ajan löytämiseksi. Mahdollisuuksia ovat tiedon ja osaamisen jakautuminen laajemmin, kun katselmointiin osallistuu enemmän kehittäjiä. Uh-

kana ovat aiheen rönsyily, osallistujien tekijä vastaan muut -asetelma sekä kehittäjien vastahakoisuus osallistua tilaisuuteen. Ryhmäkatselmoinnissa valvojana toimivalla henkilöllä on oltava kyky pitää keskustelu riittävän tarkalla tasolla, jotta vältytään keskustelu liian yksityiskohtaisista asioista. Keskustelu ei kuitenkaan saa olla liian ylimalkainenkaan. Katselmoinnin muodollisuus voi aiheuttaa epämiellyttävän tekijä vastaan muut -asetelman, mikä voi aiheuttaa vastahakoisuutta kehittäjillä osallistua tilaisuuteen.

Kuvio 7: Ryhmäkatselmoinnin SWOT-analyysi

Kuviossa 8 esitetään läpikäynnin vahvuudet, heikkoudet, uhat ja mahdollisuudet. Koska osallistujia on useita, virheiden havaitsemisen todennäköisyys kasvaa, mutta suurimpana heikkoutena on katselmoinnin eteneminen. Katselmointitilaisuutta ei ole ennalta suunniteltu, joten eteneminen riippuu osallistujista. Keskustelu saattaa olla hyvin vähäistä tai äityä pitkiksi analyyseiksi. Läpikäynti on kuitenkin erinomainen ympäristö tiedon jakamiselle osallistujien määrän ja vapaamuotoisuuden vuoksi.

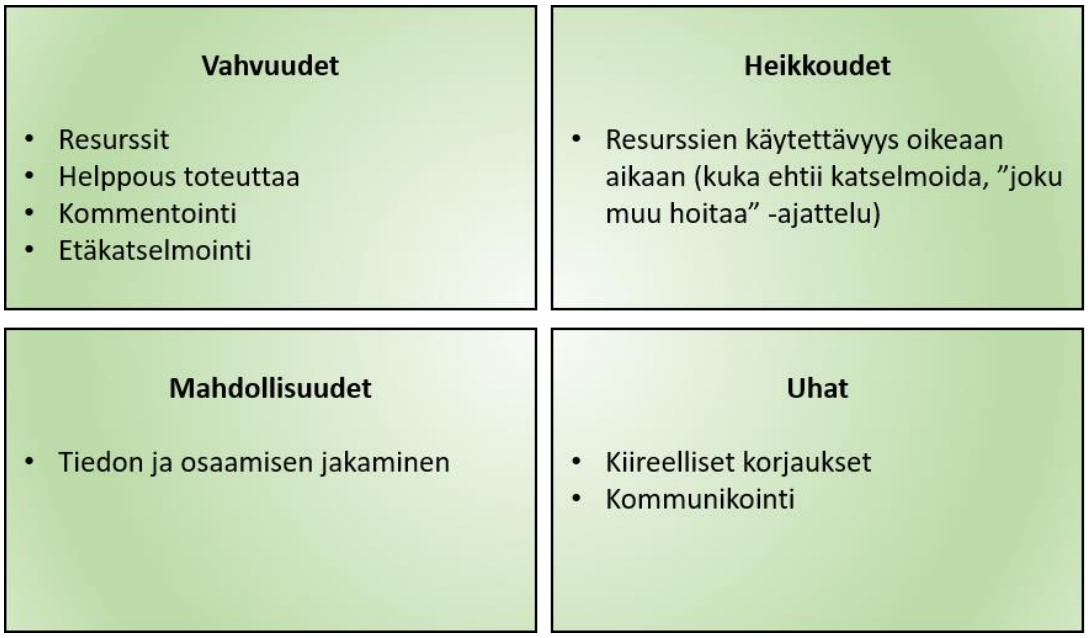


Kuvio 8: Läpikäynnin SWOT-analyysi

Pariohjelmoinnin SWOT-analyysi on esitetty kuviossa 9. Sen vahvuuksia ovat sen vaatimat suhteellisen vähäiset resurssit havaittuihin virheisiin nähden, ongelmien ratkaisutehokkuus sekä virheisiin puuttuminen heti. Lisäksi voidaan varmistua, että kehittäjätiimissä on aina kaksi henkilöä, joilla on lähtökohtaisesti yhtäläinen ymmärrys ja osaaminen kehityskohteesta. Heikkouksina on dokumentaation puute sekä resurssit. Riittävien resurssien mahdollistaminen vaatii esimieheltä työn uudelleenorganisoimista. Mahdollisuuksia pariohjelmoinnissa ovat tiedon ja osaamisen jakaminen. Keskusteluja on helpompi käydä kuin suuremmassa ryhmässä ja ratkaisuista sekä valinnoista keskustelu kehittää molempien osapuolien taitoja. Uhiksi voidaan laskea yhteinen sokeus virheille sekä vastahakoisuus osallistua pariohjelmointiin. Kun kaksi kehittäjää työskentelee yhdessä, he saattavat tulla sokeiksi omille virheilleen yhtä lailla kuin kehittäjä yksin. Tämä voi olla tulosta, jos rooleja ei vaihdeta riittävän usein. Lisäksi osa kehittäjistä voi kokea pariohjelmoinnin tai jommassakummassa roolissa toimimisen tehottomaksi, eikä siksi mielellään työskentele parin kanssa.

Kuvio 9: Pariohjelmoinnin SWOT-analyysi

Kuviossa 10 kuvataan pull requestin SWOT-analyysi. Pull requestien vahvuudet ovat vaadittujen resurssien vähyydessä, toteutuksen vaivattomuudessa sekä edes jonkin tasoisessa dokumentaatiossa, eikä se vaadi tekijää ja katselmoijaa olemaan samassa tilassa eli katselmointi onnistuu myös etänä. Heikkoutena on suunnitelmattomuus eli katselmoinnista ei sovita etukäteen katselmoijan kanssa. Mahdollisuutena on tiedon sekä osaamisen jakaminen ja uhkia ovat kiireelliset korjaukset sekä kommunikointi. Kiireellisissä korjauksissa voi olla vaikea löytää katselmoijaa, jolla on aikaa perehtyä tehtävään ja katselmoitavaan ohjelmointikoodiin. Sähköisissä viestimissä kommunikointi voi aiheuttaa epäselvyyksiä.



Kuvio 10: Pull requestin SWOT-analyysi

Parin kanssa tehdyn tarkastuksen SWOT-analyysi kuviossa 11. Parin kanssa tehdyn tarkastuksen vahvuuksia ovat resurssien vähäinen tarve sekä toteuttamisen helppous. Katselmoinnin järjestämiseen tarvitaan toinen kehittäjä, joka toimii katselmoijana eli ainoastaan työaikaa kahdelta kehittäjältä. Heikkoutena ovat yhtäkkiä tapahtuvat katselmoinnit, joita varten resursseja ei välttämättä ole sekä dokumentaation puute. Mahdollisuutena on tiedon sekä osaamisen jakaminen ja ongelmien ratkaisutehokkuuden paraneminen, kun ongelmat on mahdollista ratkaista ja virheet korjata katselmoitaessa. Uhkina ovat kiireellisten korjausten viivästyminen sekä virheiden korjaamatta jääminen.

Kuvio 11: Parin kanssa tehdyn tarkastuksen SWOT-analyysi

Ad hoc -katselmoinnin vahvuudet, heikkoudet, uhat ja mahdollisuudet on esitetty kuviossa 12. Vahvuuksia ovat vaivattomuus ja resurssien tehokas käyttö. Heikkouksina ovat yhtäkkiset katselmointitilanteet, jolloin resursseja voi olla vaikea saada käyttöön. Lisäksi tilanne on katselmoijalle haastava, kun ad hoc -katselmointitapa vaatii kokonaisuuden nopeaa ymmärtämistä. Katselmoitavana on usein liian pieniä osia ohjelmakoodista, että katselmoijan on lähes mahdotonta saada kokonaiskuvaa ja havaita esimerkiksi arkkitehtuurisia ongelmia, mikä on myös uhka. Minkäänlaista dokumentaatiota ei myöskään synny. Ad hoc -katselmointi tarjoaa mahdollisuuden tiedon ja osaamisen jakamiseen sekä ongelmat ratkeavat todennäköisesti tehokkaammin.

Kuvio 12: Ad hoc -katselmoinnin SWOT-analyysi

6.1 Valitut toimintatavat

Handelsbankenin .NET-kehittäjätiimille osoitetut kehitystyöt vaihtelevat sisällön, koon ja kriittisyyden mukaan, joten samaa koodikatselmointityyppiä ei ollut kannattavaa suositella käytettäväksi kaikkiin kehitystöihin. Myös koodikatselmointityyppien vertailu osoitti, että yhtä oikeaa katselmointitapaa ei kannata valita, vaan eniten hyödytään silloin, kun valitaan katselmointitapa kehityskohteen mukaan. Mitä kriittisempi kehitystyö on, sitä muodollisempaa koodikatselmointityyppiä ehdotettiin käytettäväksi. Näiden perusteella koodikatselmointisuunnitelmassa suositeltiin, että kehittäjätiimi ja organisaation johto sopivat yhdessä, miten tehtävän kriittisyys määritellään, kenen toimesta sekä miten kehittäjätiimi saa siitä tiedon. Missään tilanteessa tehtävän kriittisyyden arviointi ei tule olla yksin kehittäjän vastuulla.

Koodikatselmointisuunnitelmassa kehitystyöt jaettiin kolmeen eri kategoriaan niiden koon, sisällön ja kriittisyyden mukaan. Ylläpitotyöksi laskettiin sellainen kehitys, jossa tehdään pienehköjä muutoksia olemassa olevaan ohjelmakoodin. Pienkehitystyö käsitti työt, jotka ovat kooltaan pienehköjä tai keskisuuria, ja joiden kriittisyys on matala. Kehitystyöksi laskettiin kaikki muu kehittäminen. Ylläpitotyölle koettiin riittäväksi, että katselmointi suoritetaan käymällä läpi pull request, sillä muutos on yleensä suhteellisen pieni kokonaisuus, kuten vian korjaus, joka tehdään olemassa olevaan ohjelmakoodiin, eikä sen vuoksi vaadi muodollisempaa katselmointia. Pienkehitystyölle suositeltiin käytettäväksi pull requestien lisäksi joko pariohjelmointia tai läpikäyntiä, koska pienkehitystyö luokiteltiin uuden kehittämiseksi, jolloin on erityisen tärkeää varmistua, että ohjelmakoodi täyttää hyvien ohjelmointikäytänteiden kriteerit. Kehitystöiden katselmointitavaksi ehdotettiin pull requesteja, läpikäyntiä tai ryhmäkatselmointia. Läpikäynti tai ryhmäkatselmointi suositeltiin järjestettäväksi vähintään kerran

kehitysprosessin aikana. Lisäksi suositeltiin, että pull requesteja käytetään aina sekä tarpeen ja mahdollisuuksien mukaan myös pariohjelmointia. ReSharper ja SonarQube ehdotettiin otettavaksi käyttöön kehittäjän tueksi jokaiseen kehitystehtävään. Koodikatselmointityypeistä parin kanssa tehtävää tarkastusta ei nostettu suunnitelmaan, sillä sen hyödyt verrattuna esimerkiksi pull requestin tai pariohjelmoinnin käyttöön jäävät vähäisiksi.

Tarkistus olisi liian raskas toteuttaa, eikä sovi toimeksiannon määrittelemiin rajoihin. Tarkoituksena oli selvittää koodikatselmointitapoja .NET-kehittäjätiimille, eikä resursseja ole riittävästi, että tarkistuksia olisi mahdollista järjestää tarpeeksi usein. Ad hoc -katselmointia käytettiin jo entuudestaan paljon kehittäjien kesken. Vapaamuotoisuutensa vuoksi se ei kuitenkaan ollut sellainen katselmointityyppi, joka vastaisi toimeksiantajan tarpeita, sillä niistä puuttuu struktuuri, eikä näin ollen mahdollista seurantaa tai ennakointia. Vaikka ad hoc -katselmointia ei nostettu suunnitelmaan yhdeksi katselmointitavaksi, sitä voidaan siitä huolimatta käyttää koodikatselmoinnissa lisänä tarpeen tullen, mutta sillä ei korvata suunnitelmassa esitettyjä tapoja.

Koodikatselmoinnissa painotettiin, että kehittäjän vastuuseen kuuluu huolehtia ohjelmakoodin asianmukainen katselmointi. Kehittäjien tueksi koodikatselmoinnin järjestämisessä ehdotettiin koodikatselmointiprosessin omistajaa, mutta ensisijainen vastuu on aina kehittäjällä itsellään. Suunnitelmassa huomautettiin, että koodikatselmointi tulee järjestää ajoissa, eikä kehittäjän tulisi koskaan odottaa, että ohjelmakoodi on hänen mielestään täysin valmis. Tämä siksi, että jos kehittäjä kokee ohjelmakoodinsa olevan niin valmis kuin mahdollista, hän on yleensä vähemmän vastaanottavainen muutoksille ja suurempien korjausten tekeminen on työläämpää. Lisäksi koodikatselmointisuunnitelmassa suositeltiin, että katselmoitava ohjelmakoodin määrä pidetään kohtuullisena, sillä liian suuren kokonaisuuden katselmointi veisi turhan paljon aikaa sekä virheiden ja kehityskohteiden havaitseminen vaikeutuisi. Suunnitelmassa myös muistutettiin, että kehittäjien on huomioitava, ettei tavoiteltavaa ole katselmointitilaisuus, jossa ei löydy mitään korjattavaa. Sen sijaan tavoiteltavaa on, että ratkaisuista voidaan käydä avointa keskustelua, josta laadukkaamman ohjelmakoodin lisäksi saavutetaan osaamisen kasvattamista sekä kehittäjän, että katselmoijan kannalta. Tällöin tulisi pikemminkin kyseenalaista koodikatselmoinnin onnistuminen, jos sen seurauksena ei ole löytynyt yhtään huomioita. Jos katselmoitavaa ohjelmakoodia on paljon, koodikatselmointisuunnitelmassa suositeltiin harkitsemaan uuden katselmointitilaisuuden järjestämistä. Lisäksi muistutettiin, että esimiehen vastuulla on tarjota sellainen työskentely-ympäristö, jossa koodikatselmointia on mahdollista toteuttaa. Tämä tarkoittaa, että ohjelmistokehitysprosessia tulee jäsennellä uudelleen niin, että koodikatselmointia varten on varattu riittävä määrä resursseja. Koodikatselmointisuunnitelmassa ei otettu kantaa resurssien määrään, mutta painotettiin kehittäjien työajan uudelleen järjestämistä. Suunnitelman mukaisesti ehittäjätiimin jäsenet ovat vastuussa yhteisestä työilmapiiristä, mutta viime kädessä esimiehellä on velvollisuus puuttua tilanteeseen, jos työilmapiiri ei pysy avoimena.

Kun koodikatselmointi on osa kehitysprosessia, on hyödyllistä, jos virheiden ja kehityskohteiden havaitsemisista pidettäisiin kirjaa, jotta prosessia olisi mahdollista kehittää. Tilastointia varten ehdotettiin kerättäväksi tiedot siitä, minkälaisia virheitä tai kehityskohteita havaittiin, mitä katselmointityyppiä käyttämällä ja kuinka suuresta huomiosta on kyse. Esimiesten ja kehittäjätiimin on mahdollista seurata muun muassa sitä, minkä tyyppisiä virheitä esiintyy eniten ja minkälaisia katselmointityyppiä käyttämällä niitä havaitaan. Koodikatselmoinnin tarkistuslistaa suositeltiin käytettäväksi kaikissa katselmoinnissa ainakin ensimmäisen puolen vuoden ajan, jotta niitä voidaan seurata. Organisaation johdon on tärkeää huolehtia riittävät resurssit myös tilastojen seurantaan, sillä ilman seurantaa, tarkistuslistan täyttäminen on turhaa työtä ja kehittäjien sekä koodikatselmointiprosessin kehittäminen hankaloituu.

6.2 Koodikatselmointisuunnitelma

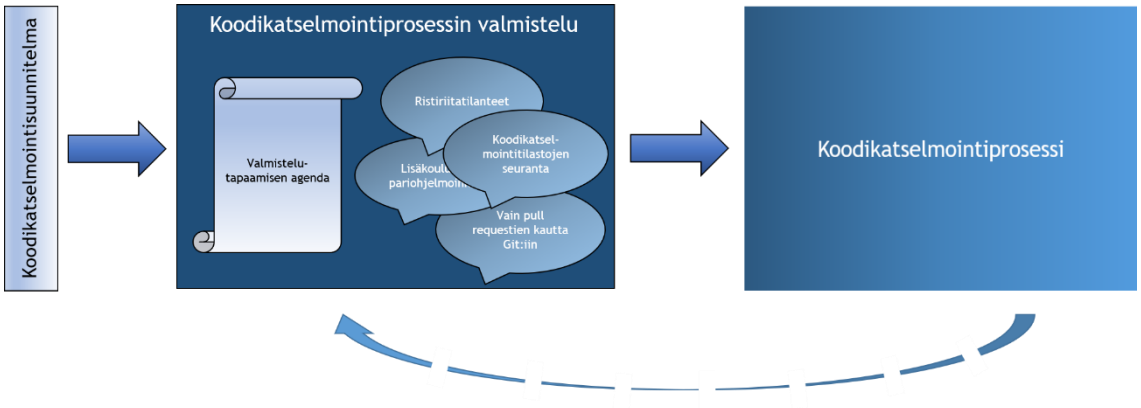
Koodikatselmointisuunnitelma sisälsi työtehtävien luokittelun, ehdotuksen kulloinkin käytettävistä koodikatselmointitavoista, agendan suunnitelman läpikäyntiä varten sekä tarkistuslistan katselmoinnin tueksi ja havaittujen virheiden tilastoimiseksi. Yhden koodikatselmointitavan valitseminen käytettäväksi kaikissa kehitystöissä ei ollut perusteltua, sillä koodikatselmointisuunnitelman päätavoitteina oli varmistaa konsernin sisäisten vaatimusten täyttyminen, ohjelmakoodin laadun parantaminen ja osaamisen jakaminen olemassa olevilla resursseilla. Ei ole kannattavaa käyttää massiivisia, paljon resursseja vieviä koodikatselmointitapoja tehtävissä, jotka ovat kooltaan suhteellisen pieniä. Koodin laadun kannalta on taas uhka, jos suuria kehitystöitä viedään tuotantoon ilman strukturoidumpaa koodikatselmointia, jossa virheiden havaitseminen on todennäköisempää ja osaamista on mahdollista jakaa laajemmin. Tästä syystä koodikatselmointisuunnitelmassa suositeltiin käytettäväksi työtehtävien luokittelua kolmeen eri kategoriaan, joiden perusteella valikoituu kehitystehtävässä käytettävä koodikatselmointitapa, mikä on kuvattu taulukossa 1. Näin kehittäjillä on käytössään valmiit ohjeistukset, miten toimia koodikatselmoinnin osalta, voidaan maksimoida havaittujen virheiden määrä sekä mahdollistaa osaamisen jakaminen niin, että kokonaisuus on riittävän iso, että ohjelmakoodin katselmointi tiedon ja osaamisen jakamisen näkökulmasta on mielekästä ja hyödyllistä.

Kehitystehtävä	Koodikatselmointitapa (minimi)
Ylläpitotyö	Pull request
Pienkehitystyö	Pull request Pariohjelmointi ja/tai Läpikäynti
Kehitystyö	Pull request Läpikäynti ja/tai Ryhmäkatselmointi

Taulukko 1: Koodikatselmointitavat kehitystehtäväkategorioittain

Koodikatselmointiprosessin omistajaksi suositeltiin .NET-kehittäjätiimin esimiestä, sillä prosessin omistajalla tulisi olla vahva ymmärrys koodikatselmoinnin tarpeesta ja riittävästi auktoriteettia sen mukaan ottamiseksi osaksi kehitysprosessia. Suunnitelmassa kuitenkin mainittiin, että koodikatselmointiprosessiin voidaan myöhemmin valita prosessin omistajaksi joku .NET-kehittäjistä, sillä heillä on potentiaalisesti paremmat lähtökohdat koodikatselmointiprosessin kehittämiseen niin kokemuksen kuin myös teknisten seikkojen kannalta. Koodikatselmointisuunnitelmaa toimeenpannessa on tärkeää, että esimies edustaa niin organisaation näkemystä kuin myös kehittäjien mielipiteitä, minkä vuoksi on hyödyllistä, että prosessiomistajana toimii esimies. Kun koodikatselmointi on osa kehitysprosessia ja kaikilla on yhteinen näkemys sen toteutustavoista ja tavoitteista, voi omistajaksi vaihtaa kokeneen kehittäjän, jolla on konkreettisempi näkemys työssä käytettävistä tekniikoista ja tehtävistä ja näin ollen todennäköisesti enemmän hyötyä prosessin ylläpitotehtävissä.

Koodikatselmointisuunnitelman toteutus on visualisoitu kuviossa 13. Koodikatselmointisuunnitelma toimii pohjana valmistelutapaamiselle, jossa keskustellaan ja päätetään valmistelutapaamisen agendalla olevista asioista ja muista esille nousevista seikoista. Valmistelutapaamisen perusteella on syntynyt koodikatselmointiprosessi, joka voidaan ottaa käyttöön. Koodikatselmointiprosessi on jatkuva, joka kehittyy ja mahdollisesti muuttuu ajan myötä. Ehdotukset muutoksiin voivat tulla kehittäjiltä tai koodikatselmointiprosessin omistajalta. Jos tarvittavat muutokset ovat merkittäviä, koodikatselmointisuunnitelmassa suositeltiin palaamaan valmisteluvaiheeseen, jossa avoin keskustelu ja mielipiteiden jakaminen on mahdollista ja kaikilla osallisilla on ollut mahdollisuus vaikuttaa prosessiin. Pienemmissä muutoksissa valmistelutapaamisen järjestäminen ei ole välttämätöntä.



Kuvio 13: Koodikatselmointisuunnitelman toteutus

6.2.1 Valmistelu

Valmistelutapaaminen ehdotettiin järjestettäväksi, sillä yksi koodikatselmoinnin onnistumisen avaintekijöistä on kehittäjien sitoutuminen siihen. Kun kehittäjät pääsevät vaikuttamaan sisältöön ja kertomaan näkemyksiä tämän hetkisestä tilanteesta verraten tulevaisuuteen, luodaan todennäköisemmin kestävämpi prosessi kuin silloin, kun jokainen kohta on ennalta määrätty. Valmistelevan tapaamisen tarkoituksena on myös herätellä kiinnostusta koodikatselmointia kohtaan ja ylläpitää avointa työilmapiiriä. Valmisteluvaiheessa kehittäjätiimi käy läpi koodikatselmoinnin tavoitteet organisaation johdon kanssa. Sen perusteella määrittyy yhteiset toimintatavat, joilla päämäärä saavutetaan. Kehittäjätiimin yhteinen missio ja suunnitelma sitouttaa kehittäjät koodikatselmointiprosessiin. Yhteisistä säännöistä sopiminen ja organisaation johdolta saatu tuki luovat tervettä työilmapiiriä, jossa koodikatselmointi ei ole ainoastaan välttämätön paha. Sitä varten järjestetään tapaaminen, jonka agendalla on keskustella ja sopia seuraavista kohdista:

- Kaikissa projekteissa käytetään ReSharperia ja SonarQubea.
- Versionhallinnan päähaaraan yhdistetään vain pull requestit eli versionhallintaan ei viedä ohjelmakoodia, jota kukaan muu ei olisi katselmoinut.
- Kuka määrittelee projektin kriittisyyden ja minkä kanavan kautta tieto välittyy kehittäjätiimille?
- Koodikatselmoinnin tavoitteena on turvallisuuden edistäminen, tehokkuuden lisääminen sekä ylläpidettävyys. Lisäksi tavoitteena on vastuun jakaminen kehittäjätii-missä laajemmin sekä tiedon ja taidon välittäminen toisilta ja toisille.
- Tarvitaanko koulutusta koodikatselmoinnista yleisesti?

- Miten toimitaan, jos kehittäjillä vastahakoisuutta toimia suunnitelman mukaisesti?
- Kuka seuraa havaittujen virheiden tilastoa?
- Kuka järjestää tapaamiset koodikatselmointiprosessin toimivuuden seuraamiseksi, jossa käydään läpi muun muassa kehitysideoita ja edistymistä ja haasteita?

Koodikatselmointisuunnitelmassa suositeltiin, että kaikki suunnittelupalaverissa käydyt asiat kirjataan ylös ja suunnitelmassa avoinna olevista asioista sovitaan ja lopputulokset vahvistetaan koodikatselmointisuunnitelmaan. Suunnitelman mukaisesti vastuu koodikatselmointisuunnitelman ylläpidosta on oltava koodikatselmointiprosessin omistajalla.

6.2.2 Tarkistuslista

Koodikatselmointiprosessin kehityksessä ja onnistumisen mittaamisessa on tärkeää, että havaittujen virheiden määrää on mahdollista seurata. Tätä varten koodikatselmointisuunnitelma sisälsi mallin tarkistuslistasta, jossa määritellään ne osa-alueet, joista virheitä tai puutteita etsitään. Tarkistuslistamalli liitettiin koodikatselmointisuunnitelmaan myös siksi, että se toimii lisäksi dokumentaationa, jonka avulla voi seurata, onko koodikatselmoinnissa ilmenneet virheet ja puutteet korjattu. Tarkistuslistalla varmistettiin, että kaikki vaaditut kohdat läpikäydään ja katselmointi on tasalaatuista huolimatta siitä, kuka kehittäjätiimistä toimii katselmoijana. Samaa tarkistuslistaa hyödynnettiin kaikissa koodikatselmointisuunnitelmaan valituissa koodikatselmointitavoissa. Koodikatselmointisuunnitelmassa suositeltiin, että aluksi tarkistuslistaa käytetään kaikissa katselmointitilaisuuksissa, jotta tarkistuslistan käyttö tulee luontevaksi ja sitä on mahdollista kehittää edelleen, mutta myöhemmin sen käyttö ei ole pakollista muissa kuin ryhmäkatselmoinnissa. Seurantamahdollisuuden takaamiseksi ehdotettiin, että kaikki katselmointitilaisuuksissa käytetyt tarkistuslistat tallennetaan kyseisen projektin hakemistoon tai Confluence-tilaan. Taulukossa 2 on kuvattu .NET-kehittäjätiimille tarkoitettu tarkistuslista, jossa on otettu huomioon yleinen, hyvä ohjelmointitapa.

Tarkasteltava osa-alue	OK / -	Jatkotoimenpiteet
Tehtävänannon läpikäynti suoritettu		
ReSharperin analyysi läpi		
Nimeämiskäytäntöjä noudatettu • Ei sisällä akronyymeja, lyhenteitä tai järjestelmien nimiä		
Modulaarisuus / Single Responsible Principle		
Kommenttien käyttö • Ei sisällä kommentoitua ohjelmakoodia • Ei sisällä keskeneräisiä toiminnallisuuksia kommentoituina		
Samaa ohjelmakoodia ei toisteta useissa paikoissa		
Silmukoilla on oikeat lopetusehdot		
Sisältää tarvittavat tarkastukset syötteille		
Sisältää tarvittavat tarkastukset tulosteille		
Sisältää käsittelyn väärien parametrien varalta		
Tehokkuus (esimerkiksi optimaaliset HTTP-kutsut ja tietokantakyselyt)		
Turvallisuus huomioitu • Oikeuksien todentaminen • Käyttöoikeudet • Aineistonvalidointi • Tietojen asianmukainen käsittely • Istunnonhallinta • Lokitus • Virhekäsittely • Kryptaus		
Rajapinnoissa huomioitu • Dokumentaatio • Käyttöoikeudet • Saatavilla vain tarvittava • Johdonmukaisuus		
Testien kattavuus		

Taulukko 2: Tarkistuslistamalli

7 Yhteenveto

Tässä opinnäytetyössä esiteltiin yleisesti koodikatselmoinnin tarkoitus kehitystyössä, tapoja suorittaa koodikatselmointia sekä koodikatselmoinnin vaikutuksia niin ohjelmakoodiin kuin myös sitä harjoittaviin kehittäjiin ja koko organisaatioon. Teoreettisessa osuudessa esitettyjä osa-alueita peilattiin havainnoinnin tuloksiin, minkä pohjalta kehittämistyötä toteutettiin. Lisäksi koodikatselmointia käsiteltiin sekä teknisestä, että kehittäjien näkökulmasta.

Kehittämistyön kohteena oli koodikatselmointisuunnitelman kehittäminen Handelsbankenin .NET-kehittäjätiimille. Kehittämistyön tuloksena syntyi suunnitelma, joka sisälsi ehdotuksen eri koodikatselmointityyppien käytöstä, koodikatselmointiprosessin valmistelevan kokouksen järjestämisestä agendoineen sekä koodikatselmoinnin toteutuksesta ja seurannasta. Suunnitelmassa huomioitiin toimeksiantajan asettamat vähimmäisvaatimukset koodikatselmoinnin suorittamiselle, mutta siihen nostettiin myös teoreettisessa osuudessa esille nousseiden huomioiden perusteella myös muita elementtejä, joiden avulla saavutetaan toimeksiannon muut vaatimukset. Näitä olivat sellaisten koodikatselmointitapojen valitseminen, joilla mahdollistetaan kehittäjien ammattitaidon kehittyminen tiimin sisällä sekä tiimin vastuun projekteista sekä lähdekoodista yksilöiden vastuun sijaan.

7.1 Pohdinta

Koodikatselmointi on aihealueena laaja käsittäen teknisen ja käytännön toteutukset sekä henkilöiden vuorovaikutukseen ja työhyvinvointiin liittyvät piirteet. Aihe herättää mielipiteitä ja toteutuksia on olemassa todennäköisesti yhtä monta kuin on organisaatioita, joissa koodikatselmointi on osa kehitystyötä. Toisaalta koodikatselmointi voidaan nähdä myös kilpailuvalttina, sillä tehokkaammat, virheettömämmät ja kehittäjille hyödyllisemmät katselmointitavat tuovat huomattavasti lisäarvoa. Jos kilpailijalla on käytössä kankea ja hengetön koodikatselmointiprosessi ero näiden kahden välillä on huomattava. Koodikatselmointi kehittyy ja monipuolistuu ajan myötä, kun ohjelmointiympäristöt ja toimintamallit muuttuvat.

Automaattiset staattisen analyysin työvälineet kehittyvät koko ajan ja tuntuu olevan vaikeaa löytää perusteluja olla käyttämättä niitä. Tällä hetkellä niillä ei kuitenkaan voida vielä täysin korvata manuaalisia eli ihmisten toteuttamia katselmointeja, sillä niiden käyttämät algoritmit eivät vielä pysty havaitsemaan esimerkiksi kaikkia rakenteellisia tai arkkitehtuurisia ongelmia, jotka ihmisen on mahdollista havaita ohjelmakoodista. Lisäksi Handelsbankenin käytössä olevista staattisen analyysin työvälineistä puuttuu vielä tällä hetkellä kommunikointimahdollisuus, minkä vuoksi manuaalisia katselmointitapoja ei ole vielä syrjäytetty kokonaan. Tekoälyn kehittyessä myös koodikatselmointiin käytettävä tekniikka kehittyy ja tulevaisuudessa voi olla mahdollista suorittaa koodikatselmointia täysin automaattisesti.

Opinnäytetyön aiheen rajoissa pysyminen oli yksi tärkeistä tekijöistä tämän opinnäytetyön onnistumisen kannalta. Aihe onnistuttiin rajaamaan riittävän selkeästi niin, että opinnäytteen

teoreettinen osio tuki kehittämistyötä. Suurimpia haasteita aiheen rajauksessa oli löytää selkeä linja, kuinka tarkasti koodikatselmoinnin sosiologisia vaikutuksia tutkitaan. Kehittämistyön edetessä niiden merkitys korostui entisestään, mutta aihe oli saatava riittävän kompaktiksi, jotta työ vastaa tutkimusongelmaan. Valitut tutkimus- ja kehittämismenetelmät tukivat sekä teoreettisen osion, että koodikatselmointisuunnitelman luonnissa. Havainnoinnin valitseminen tutkimusmenetelmäksi oli perusteltua, sillä tutkimuskohteesta pyrittiin saamaan mahdollisimman todenmukainen kuva ja osallistuvalla havainnoinnilla siinä onnistuttiin. Osallistuva havainnointi oli tässä tapauksessa toimiva keino, sillä näin kehittämiskohdetta voitiin tutkia aiheelle asetetuissa rajoissa. Toisaalta kehittäessä koodikatselmointisuunnitelmaa kävi ilmi, että haastattelut tutkimusmenetelmänä olisivat voineet tuoda lisäarvoa tutkimukselle ja erilaisen lopputuloksen kehittämistyölle. Haastattelujen avulla olisi ollut mahdollista kerätä julkaistavia seikkoja, joihin kehittämistyö olisi pohjautunut. Lopulta haastatteluja kuitenkin päätettiin olla käyttämättä opinnäytetyön aiheen vuoksi. Kokemukseni on, että useissa tapauksissa kehittäjät ymmärtävät koodikatselmoinnin tärkeyden, mutta siitä huolimatta siihen ei panosteta riittävästi tai sitä jopa vältellään. Tästä syystä koin, että haastattelujen vaarana olisi ollut se, etteivät vastaukset olisi välttämättä olleet todenmukaisia, vaan niitä olisi jossain määrin voitu kaunistella. Tällöin tutkimuskohteesta ei olisi saatu riittävän todenmukaista kuvaa.

Kokonaisuutena tarkastellen opinnäytetyön voitiin nähdä onnistuneen, sillä sille asetettuun tutkimuskysymykseen vastattiin kehittämistyössä esitetyillä ehdotuksilla. Myös muihin tutkimuskysymyksiin vastattiin teoreettisessa osiossa ja näin ollen voitiin todeta, että tutkimusongelma ratkaistiin. Koodikatselmointisuunnitelmaa, joka oli kehittämistyön tulos, ei ole vielä implementoitu Handelsbankenin .NET-kehittäjätiimissä, mutta se esiteltiin alustavasti ja palaute on ollut positiivista. Koska suunnitelmassa painotetaan jokaisen kehittäjän osallistamista lopullisen koodikatselmointiprosessin luontiin, ei siinä esitettyjen toimintamallien toimivuuden mittaaminen ole relevanttia. Koodikatselmointisuunnitelman toimivuutta voidaan kuitenkin mitata sillä, kuinka nopeasti koodikatselmointiprosessi saadaan osaksi kehitystyöprosessia. Kun koodikatselmointisuunnitelmassa on huomioitu kaikki oleelliset osa-alueet ja selvitystyö on tehty perusteellisesti, ei koodikatselmointiprosessin implementointiin mene kauaa.

7.2 Johtopäätökset

Koodikatselmointityyppien erot ovat toisaalta hyvin suuria, mutta toisaalta hyvin samankaltaiset erot nousevat esiin eri koodikatselmointityyppejä vertailtaessa. Suurimpana erona esille nousee resurssit, niiden käyttö ja erityisesti niiden uudelleenorganisoiminen. Organisaatiolle, jossa koodikatselmointi ei ole osa kehitysprosessia, voi olla haastavaa budjetoida koodikatselmoinnin vaatimat resurssit. Työn uudelleenorganisoiminen vaatii panostusta organisaation johdolta, mutta myös kehittäjätiimiltä. Heidän tulee opetella uudet työrutiinit ja tiiviimpää

yhteistyötä kollegojen kesken. Vuorovaikutustaidot ovat ensiarvoisen tärkeät koodikatselmointiprosessin toimivuuden kannalta.

Koodikatselmoinnin onnistuminen on riippuvainen monista eri tekijöistä niin kuin teoreettisessa osassa käy ilmi. Koodikatselmointisuunnitelma onkin vain yksi vaihe. Suunnitelma on oleellinen osa toimivuutta, mutta koodikatselmointiprosessin kehittymisen on jatkuttava ja siinä käytettävien toimintatapojen on kohdattava sille asetetut tarpeet, mikä onnistuu koodikatselmointisuunnitelman säännöllisellä päivittämisellä. Handelsbankenin .NET-kehittäjätiimin koodikatselmointisuunnitelmaan on nostettu ne asiat, jotka SWOT-analyysia hyödyntäen nähtiin suotuisimmiksi.

8 Jatkokehitys

Tutkimusaihe ja kehittämiskohde olivat erittäin mielenkiintoisia, aihetta on käsitelty paljon erilaisissa julkaisuissa ja sen merkitys kehitystyössä on suuri. Tämän ansiosta myös jatkokehitysehdotuksia syntyi jonkin verran. Aihe on myös sinänsä otollinen jatkokehitysehdotuksille, sillä koodikatselmointi itsessään kehittyy koko ajan.

Jatkotyönä olisi mielenkiintoista toteuttaa koodikatselmointisuunnitelman implementointi, jossa sen toimivuus lopullisesti mitataan. Suunnitelman implementointi tuo omat haasteensa ja näkökulmansa aiheeseen, joita opinnäytettä tehdessä ei ole osattu huomioida. Lisäksi koodikatselmointia voisi viedä enemmän automaattiseksi ja näin ollen tutkia markkinoilla tarjottavia työvälineitä, joita olisi mahdollista käyttää koodikatselmoinnissa. Myös havaittujen virheiden seurantaan olisi mahdollista kehittää automatiikkaa. Aihetta olisi hyödyllistä tutkia myös ihmislähtöisemmästä näkökulmasta painottaen enemmän koodikatselmoinnin sosiologista puolta. Tällöin tutkimusta voisi tehdä esimerkiksi haastatteluin, mikä mahdollistaisi syventymisen myös näkökulmien ja mielipiteiden taustalla oleviin syihin. Myös koodikatselmointietiketin kehittäminen olisi hyödyllistä, kun siinä huomioitaisiin hyvä katselmointitapa niin tekijän kuin katselmoijan näkökulmasta.

Lähteet

Painetut

McConnell, S. (2004). Code Complete. 2. painos. Washington: Microsoft Press.

Pressman, R. (2010). Software Engineering: A Practitioner's Approach, Seventh Edition. 7. painos. New York: McGraw-Hill.

Sarajärvi, A. & Tuomi, J. (2018) Uudistettu laitos. Kustannusosakeyhtiö Tammi.

Wiegers, K. (2002). Peer Reviews in Software. 2. painos. Boston: Pearson Education.

Zhu Y. (2016) Software Reading Techniques. New York: Springer Science +Business Media.

Sähköiset

Alasuutari, P. Mitä on laadullinen tutkimus?. Viitattu 29.9.2019. http://wwwedu.oulu.fi/tohtorikoulutus/jarjestettava_opetus/Alasuutari/Mita_laadullinen_tutkimus_on.pdf

Arisholm, E., Dybå, T., Hannay, J. & Sjøberg, D. The effectiveness of pair programming: A meta-analysis. Viitattu 29.8.2019. http://www.ic.unicamp.br/~wainer/outros/systrev/30.pdf

Bacchelli, A. & Bird, C. 2013. Expectations, Outcomes, and Challenges Of Modern Code Review. Viitattu 4.9.2019. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ICSE202013-codereview.pdf

Bodner, H. 2018. 10 Reasons Why Code Reviews Make Better Code and Better Teams. Viitattu 4.9.2019 https://simpleprogrammer.com/why-code-reviews-make-better-code-teams/

Elliott, E. 2016. The Outrageous Cost of Skipping TDD & Code Reviews. Viitattu 17.9.2019. https://medium.com/javascript-scene/the-outrageous-cost-of-skipping-tdd-code-reviews-57887064c412

Eloranta, S., Hautala, T., Kinos, S. & Salonen, K. 2017. Kehittämistoiminta ja kehittämisen menetelmiä ammatillisessa korkeakoulutuksessa. Viitattu 21.9.2019. julkai-sut.turkuamk.fi/isbn9789522166494.pdf

GitHub. 2019. Viitattu 13.8.2019. https://help.github.com/en/articles/about-pull-requests

Git. Viitattu 20.11.2019. https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Greiler, M. 2019. Proven Code Review Best Practices from Microsoft. Viitattu 27.8.2019. https://www.michaelagreiler.com/code-review-best-practices/

Handelsbanken. 2019a. Viitattu 10.9.2019. https://www.handelsbanken.com/en/about-the-group/our-story

Handelsbanken. 2019b. Viitattu 10.9.2019. https://www.handelsbanken.com/en/about-the-group/locations

Handelsbanken. 2019c. Viitattu 10.9.2019. https://www.handelsbanken.com/en/about-the-group/locations/finland

Handelsbanken. 2019d. Viitattu 10.9.2019. https://www.handelsbanken.fi/shb/inet/IStartfi.nsf/Frameset?OpenView&id=Shbfi&navid=X_Tietoa_Handelsbankenista&sa=/shb/inet/icentfi.nsf/default/q66FAA493D3C4A2F2C22570F900327F5B?opendocument&iddef=Tietoapankista

Immonen, J. 2002. Johdatus ohjelmistotuotantoon. Viitattu 8.10.2019. http://cs.joensuu.fi/~jimmonen/jot_moniste/jot_moniste_121.html

ISTQB. 2018. Perustason sertifikaattisisältö. Viitattu 24.11.2019. http://www.fistb.fi/sites/fistb/files/liitteet/CTFL%202018%20Sertifikaattisisa%CC%88lto%CC%88%2020181010-1%20Valmis.pdf

Jetbrains. 2019a. Viitattu 14.8.2019. https://www.jetbrains.com/resharper/documentation/documentation.html

Jetbrains. 2019b. Viitattu 14.8.2019. https://www.jetbrains.com/help/resharper/Sharing_Configuration_Options.html

Jetbrains. 2019c. Viitattu 30.10.2019. https://www.jetbrains.com/resharper/features/code_analysis.html

Kvalitatiivinen sisällönanalyysi. Viitattu 7.10.2019. https://metodix.fi/2014/05/19/seitamaa-hakkarainen-kvalitatiivinen-sisallon-analyysi/

Microsoft. 2015. Viitattu 24.11.2019. https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions

Pressman, R. 2010. Software Engineering A Practitioner's Approach. Viitattu 7.10.2019. http://dinus.ac.id/repository/docs/ajar/RPL-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf

Sommerville, I. 2011. Software engineering. Viitattu 8.10.2019. https://edisciplinas.usp.br/pluginfile.php/2150022/mod_resource/content/1/1429431793.203Software%20Engineering%20by%20Somerville.pdf

SonarQube. 2019a. Viitattu 15.8.2019. https://docs.sonarqube.org/latest/

SonarQube. 2019b. Viitattu 15.8.2019. https://docs.sonarqube.org/latest/analysis/overview/

SonarQube. 2019c. Viitattu 15.8.2019. https://docs.sonarqube.org/latest/extend/adding-coding-rules/

SonarQube. 2019d. Viitattu 15.8.2019. https://docs.sonarqube.org/latest/analysis/pull-request/

SonarQube. 2019e. Viitattu 15.8.2019. https://docs.sonarqube.org/latest/user-guide/metric-definitions/

SonarQube. 2019f. Viitattu 30.10.2019. https://www.sonarqube.org/features/multi-languages/csharp/

Tutkimusstrategiat. 2014. Viitattu 29.9.2019. https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/tutkimusstrategiat/

Torlak, E. 2015. Code Reviews. Viitattu 4.9.2019. https://courses.cs.washington.edu/courses/cse403/15sp/lectures/L15.pdf

van Veenendaal, E. 1999. Practical Quality Assurance for Embedded Software. Viitattu 16.8.2019. http://www.erikvanveenendaal.nl/NL/files/Practical%20Quality%20Assurance%20for%20Embedded%20Software.pdf

Visual Studio. 2019. Viitattu 14.8.2019. https://marketplace.visualstudio.com/items?itemName=JetBrains.ReSharper

Williams, L. 2000. The Costs and Benefits of Pair Programming. Viitattu 29.8.2019. https://collaboration.csc.ncsu.edu/laurie/Papers/dissertation.pdf

Zihler, O. 2017. How to do effective code reviews. Viitattu 17.9.2019. https://www.zuehlke.com/blog/en/effective-code-reviews/

Julkaisemattomat

IT Development Process at Handelsbanken. 2019. Viitattu 23.7.2019.

System Development. 2019. Viitattu 23.7.2019.

Kuviot

Taulukot