OULUN AMMATTIKORKEAKOULU

Janne Tuhkanen

SONARQUBEN KÄYTTÖÖNOTTO

Staattinen analyysi osana laadunvarmistusta

SONARQUBEN KÄYTTÖÖNOTTO

Staattinen analyysi osana laadunvarmistusta

Janne Tuhkanen
Opinnäytetyö
Kevät 2020
Tietojenkäsittely

Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma, Web-sovelluskehitys

Tekijät(t): Janne Tuhkanen
Opinnäytetyön nimi: SonarQuben käyttöönotto
Työn ohjaaja: Teppo Räisänen
Työn valmistumislukukausi ja -vuosi: Kevät 2020 Sivumäärä:21

Sain opinnäytetyöni Laavu Solutionsilta ollessani heillä työharjoittelussa. Tiedonhallinnan työkaluna käytetään M-Filesiä, joka on virtuaalinen ympäristö dokumenteille. M-Files ratkaisuihin pystytään myös kirjoittamaan mukautettuja sovelluksia, joilla voidaan ratkaista työkalun konfiguroinnilla ratkaisemattomia ongelmia. Laavu Solutions halusi löytää staattisen analyysin työkalu, jolla pystyttäisiin varmistamaan koodin laatua automaattisesti analysoimalla uusimman version lähdekoodia. Työkalu siis kiinnitettäisiin kehityskulkuun, jossa tuotteen testauksen jälkeen varmistetaan koodin laatu. Analyysistä tulisi saada tulos, jossa osoitettaisiin mahdollisia ongelmia koodissa.

SonarQube osoittautui työkaluksi joka vastasi Laavu Solutionsin tarvetta. SonarQube on ilmainen ja sen pystyy yhdistämään versiohallintaan niin, että koodi analysoidaan uuden version puskettua versionhallintaan. Uuden version kohdalla, SonarQube ajaa staattisen analyysin koodia vasten ja antaa tuloksen kehittäjälle, jossa ilmenee ongelmat ja mahdolliset tavat ratkaista kyseinen ongelma.

Tutustuin myös mukautettujen sääntöjen kirjoittamiseen, sillä SonarQube pystyy käsittelemään koodia yleisellä tasolla, joten se ei ota huomioon M-Files kirjaston ominaisuuksia. Mukautettuja sääntöjä käytetään tunnettuihin ongelmiin M-Files kirjastossa, jotka kehittäjät Laavulla tiesivät.

Avainsanat: Ohjelmointi, Staattinen analyysi, SonarQube, Koodin katselmointi, Roslyn API

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Web application development

Author(s): Janne Tuhkanen
Title of thesis: Commissioning of SonarQube
Supervisor(s): Teppo Räisänen
Term and year when the thesis was submitted: Spring 2020 Number of pages: 21

I got my thesis topic from Laavu Solutions while doing my internship there. Laavu Solutions is a company, which does information- and document management solutions for customers. As a document management tool, we use M-Files, which is virtual environment for documents. For M-Files solutions, we can create custom application that solved the problems, that default M-Files configuration could not solve. Laavu Solutions wanted to find static analysis tool, that could automatically assure the quality of the source code in M-Files solutions. Tool is attached to the development workflow, where after testing of the solution the code quality could be analyzed. Tool should also give the results of the analysis where possible problems are pointed.

SonarQube turned out that fulfills the needs Laavu Solutions has. SonarQube is free and it can be attached to the solution development workflow, that it analyzes the code when a new version is pushed to the version control. When this happens, SonarQube runs a static analysis on the code and sends the results to the developer, where all possible problems are indicated with possible ways to fix the problem.

I also studied how to write custom rules for SonarQube because the tool only analyzes code on general level, so it does not understand M-Files library. Custom rules needed to be used for common problems with M-Files library, that developers at Laavu knew.

Keywords: Programming, Static analysis, SonarQube, Code review, Roslyn API

SISÄLLYS

1 JOHDANTO

Koodin katselmointi on laadunvarmistus tapa, jossa ohjelmoija, joka ei ole ollut mukana kehittämässä koodia, antaa mahdollista negatiivista ja positiivista palautetta ohjelmoijalle koskien katselmoitavaa koodia (SmartBear Software. 2019). Koodin katselmointi on osoittautunut nopeuttavaksi ja tehostavaksi käytännöksi ohjelmistokehityksen kannalta. Katselmoitu ja korjattu koodi on helposti ymmärrettävää, siistittyä ja sisältää vähemmän bugeja. Katselmointiin kannattaa varata resursseja, sillä yleensä katselmoija osoittaa mahdolliset viat koodista, jotka sitten tulee korjata ennen seuraavaa katselmointia. Koodi katselmoidaan niin kauan, kunnes se hyväksytään ja koodi viedään tuotantoon.

Koodia pystyy myös tarkastamaan automaattisesti. Näitä kutsutaan staattisiksi- ja dynaamisiksi analyyseiksi. Staattinen analyysi on koodin validointia sellaisenaan ilman sen ajamista ja suoritetaan yleensä seuraavaksi koodin kirjoittamisen jälkeen (Ghahrai, A. 2018). Tällä tavalla, voidaan löytää ilmeisiä virheitä ennen testausta. Dynaaminen analyysi on yleensä yksikkö testausta. Koodia siis ajetaan ja tällä tavalla yritetään löytää virheitä, jotka tulevat ilmi vasta koodin ajon aikana. Automaattisella katselmoinnilla varmistetaan, että yrityksen standardi on käytössä kehitetyssä ohjelmistossa. Yhteistä kuitenkin staattisella- ja dynaamisella analyysilla on se, että siihen ei voida sokeasti luottaa, sillä automaattiset katselmoinnin tulokset saattavat antaa valheellista positiivista ja -negatiivista palautetta. Valheellisella positiivisella tarkoitetaan sitä, että annetaan ymmärtää väärästä tavasta, että se on oikein ja valheellisella negatiivisella palautteella tarkoitetaan, että hyvästä käytännöstä annetaan negatiivista palautetta. On siis tärkeää, että myös ihminen katselmoi koodin, sillä automaattinen analyysi ei ota huomioon esimerkiksi asiakkaan vaatimuksia.

Opinnäytetyö on tehty Laavu Solutions nimiselle oululaiselle yritykselle. Laavu on asiakaslähtöinen asiantuntijuutta tarjoava yritys. Yritys tekee esimerkiksi tiedonhallinnan ratkaisuja ja it-ympäristön kartoittamista. Laavu on ottanut SonarQuben käyttöön kehitystyönkulkuunsa. Olemme todenneet, että SonarQube nostaa koodin laatua ja nopeuttaa kehitystyötä, mukaanlukien koodin katselmointia.

2 OHJELMISTON LAADUNVARMISTUS

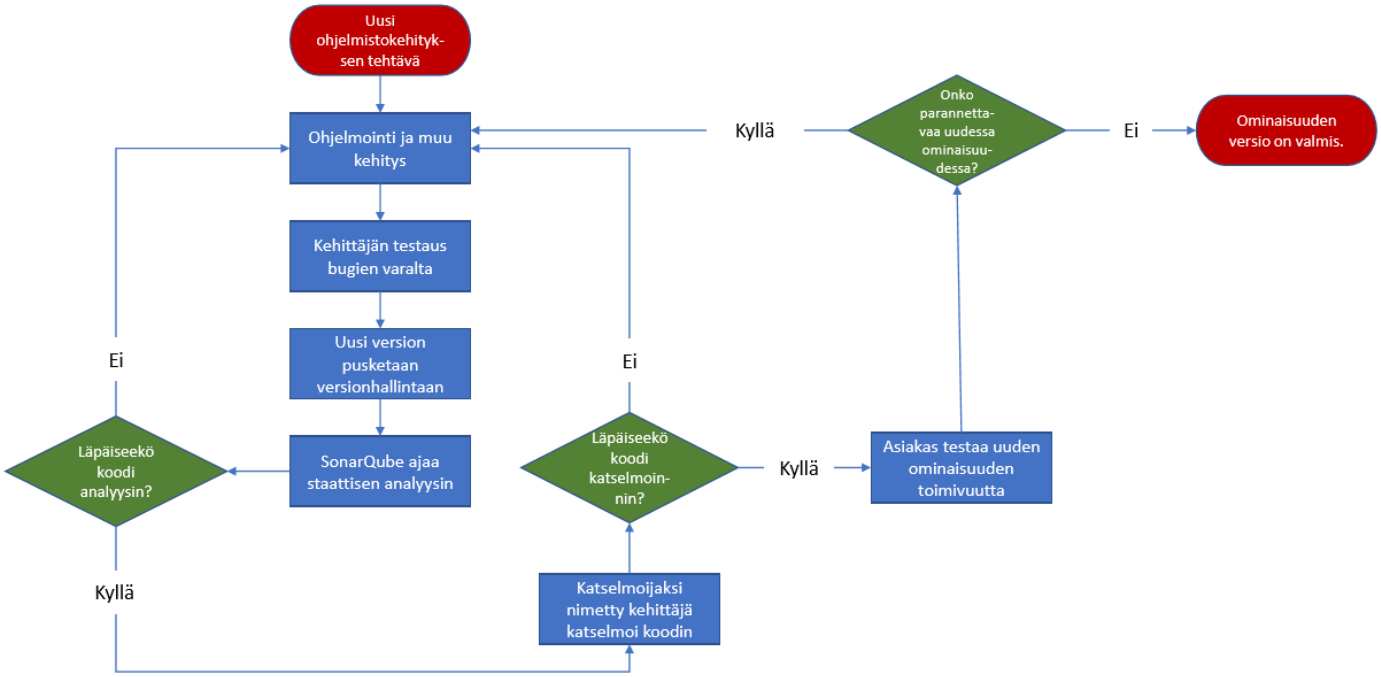
Ohjelmiston laadulla tarkoitetaan ohjelmistotuotteen kykyä täyttää käyttäjänsä kohtuulliset toiveet ja odotukset, täten ohjelmiston laatu käsitteenä on riippuva käyttäjästä ja käyttöympäristöstä. Laadulla yleensä tarkoitetaan toiminnalla mitattavia ohjelmiston ominaisuuksia. Ohjelmiston laatuun vaikutetaan toiminnan hyvällä laadulla, eli kehitysvaiheessa voidaan vaikuttaa tuotteen laatuun esim. laadunvarmistuksella. Toiminnan hyvällä laadulla tarkoitetaan lopputuotteeseen positiivisesti vaikuttavia toimintatapoja. (Hailaka, I. & Märijärvi. J. 2004.)

Tuotteen laadunvarmistuksella yritetään estää virheiden pääsemistä tuotteeseen ja auttaa löytämään tehdyt virheet mahdollisimman aikaisin. Koodin analyysin raportti saattaa olla jopa opettavainen, sillä esimerkiksi SonarQubessa, säännöissä on esimerkki, miten ei tulisi tehdä ja vaihtoehtoinen esimerkki, miten tulisi tehdä. Tällä tavalla voidaan parantaa toiminnan laatua jo ohjelmointi vaiheessa. Pahimmassa tapauksessa huono koodi saattaa hidastaa uuden ominaisuuden kehittämistä tai jopa estää sitä. Kun huono koodi estää jonkin ominaisuuden kehittämistä, saatetaan päätyä nk. ”purkkaratkaisuun”.

Kirjassa ”Clean Code”, kirjoittaja kertoo eräästä yrityksestä 80-luvulta, joka kirjoitti loisto sovelluksen. Ammattilaiset pitivät siitä ja ostivat kyseisen sovelluksen. Myöhemmin kuitenkin sovelluksen päivitykset viivästyivät ja bugeja ei ikinä korjattu ja projekti ajettiin alas. Yritys lopetti toimintansa lyhyen ajan päästä projektin lopettamisesta. Kirjoittaja kertoo tavanneensa 20 vuotta myöhemmin erään työntekijän samasta yrityksestä ja pohtivat että mikä meni vikaan. Vastauksena oli se, että projekti yritettiin saada nopeasti tuotantoon ja kiireen tuloksena koodi oli sotkuista. Uusien ja aina uusien ominaisuuksien jälkeen, sotkuinen koodi jatkoi kasvamista, kunnes kukaan ei ottanut siitä selvää. *Huono koodi* ajoi yrityksen konkurssiin.

2.1 Laavun uusi ohjelmistokehitysprosessi

Laavu Solutionsin ohjelmiston laadunvarmistus ennen SonarQuben käyttöönottoa oli perinteinen ohjelmiston katselmointi, eli ulkopuolinen katselmoijaksi nimetty kehittäjä tarkasti koodin ja hyväksyi sen oman ammattimaisen näkemyksen perusteella tai palautti sen korjattavaksi, kertoen mitä tulisi muuttaa.
Kun SonarQube otettiin käyttöön, aikaa mitä käytetään katselmointiin, on lyhyempi ja koodin virheet mitä katselmoija ei välttämättä huomaa, on kitketty työkalun antaman raportin avulla. Hyöty minkä Laavu saa automaattisesta katselmoinnista on se, että ohjelmoitu logiikka ei omaa ihmismäisiä luonteita, toimii aina samalla tavalla, muistaa aina samat käytännöt ja huomauttaa niistä joka kerta.



KUVIO1: Prosessi kaavio uudesta kehitysprosessista.

Kuviossa 1 nähdään prosessikaavio siitä, miten ohjelmistokehityksen sykli menee Laavulla, kun siihen on lisätty staattinen analyysi. Huomaa että ihminen tarkastaa vielä koodin staattisen analyysin jälkeen siltä varalta, että SonarQube ei ole saanut kiinni kaikkia virheitä.

Prosessista puuttuu dynaaminen analyysi, joka tulisi staattisen analyysin jälkeen, mutta Laavulla sitä ei ole vielä implementoitu ohjelmistokehityksen prosessiin. Lopuksi asiakas testaa käytettävyyttä loppukäyttäjän näkökulmasta.

Jos asiakas toteaa, että jonkin ei toimi tai toimii väärällä tavalla, joudutaan koko sykli aloittamaan alusta ongelman kohdalla. Tällainen ilmiö maksaa rahaa ja mahdollisesti pahimmassa tapauksessa tulevan asiakkaan. Tätä pystytään välttämään sillä, että prosessi käydään tiheämmin läpi. Tarkoittaa siis sitä, että pusketaan uusia versioita vähemmillä muutoksilla, jolloin löydettyjä virheitä pystytään korjaamaan aikaisemmin.

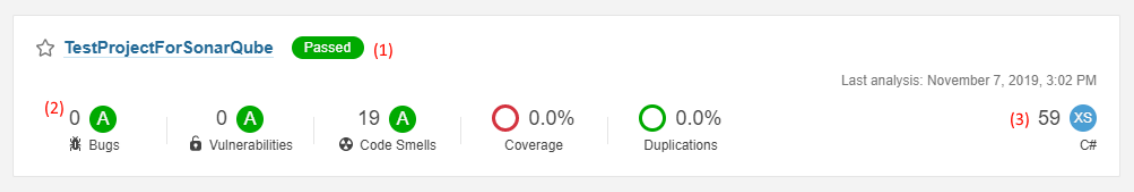
3 SONARQUBE

SonarQube on staattisen analyysin työkalu, jolla voi hakea ohjelmointivirheitä mm. bugeja ja huonoja ohjelmointitapoja ja työkalun voi helposti lisätä ohjelmistokehityksen työnkulkuun (SonarSource. 2019). SonarQubella voidaan analysoida yli 20 ohjelmointikieltä, ja useimpaan niistä voidaan kirjoittaa omia sääntöjä. Alustana se sopii myös yrityksille, sillä SonarQuben voi liittää versionhallintaan niin, että kun uusi versio pusketaan versionhallintaan, SonarQube ajaa automaattisesti analyysin sille. Laavun tapauksessa, SonarQubea käytetään C#- ja JavaScript projektien analysointiin. SonarQube on palvelimelle (myös paikallisesti) asennettava ohjelmisto, jota ajetaan komennoilla. Laavulla, näitä komentoja ajetaan versionhallinnassa automaattisesti skriptillä, kun versiosta pusketaan uusi versio.

Tässä kappaleessa käyn läpi SonarQuben käyttöliittymää. Käyttöliittymässä pystytään hallitsemaan ja tarkastelemaan esimerkiksi projekteja ja sääntöjä. Projekteissa pystytään tarkastelemaan virheilmoituksia, joita työkalu on laittanut sääntöjen perusteella. Säännöissä pystytään tarkastelemaan voimassa olevia ja käytöstä poistettuja sääntöjä. Kaikki säännöt eivät välttämättä ole yrityksen standardin mukaisia, jolloin niitä voidaan poistaa laadunvarmistuksesta helposti tai niitä voidaan kirjoittaa itse ja lisätä palvelimelle.

3.1 Projects

Projekti näkymässä voi tarkastella kaikkia analysoituja projekteja. Kielestä riippumatta, kaikki projektit tulevat samaan paikkaan, mutta näkymää voidaan suodattaa vasemmalla näkyvästä Filters-näkymästä. Se miten projektit näkyvät, voidaan säätää projektilistauksen yllä näkyvässä kojelaudalla.



KUVIO 2. Yksittäinen projekti SonarQubessa Projektit-näkymässä.

Kuviossa 2 nähdään, miten yksittäinen projekti näkyy SonarQuben projekteissa.

- (1) viittaa projektin nimeen ja pääsikö se laadunvarmistuksesta läpi.
- (2) viittaa muihin projektin ominaisuuksiin. Tarkemmin niitä pääsee tarkastelemaan kun avaa projektin.
- (3) viittaa projektin kokoon. Luku koodi merkitsee koodiriveihin ja kirjaimet koon luokitusta.

Klikkaamalla projektin nimestä, päästään tarkastelemaan projektia tarkemmin.

KUVIO 3: Projektin aktiivisuus raportissa nähdään nykyisten virheiden raportit verrattuna edellisen versioiden raportteihin.

3.2 Issues

Issues-näkymässä nähdään kaikki virheilmoitukset kielestä riippumatta. Sivulla pystyy tarkastelemaan eri tasoisia rikkeitä kaikista projekteista ja näistä rikkeistä voi tarkastella sääntöä tarkemmin. Säännössä ilmenee tarkemman kuvauksen lisäksi huono- ja toimiva esimerkki. On myös hyvä huomioida, että rikkeet eivät aina ole vakavia ja ne voidaan ottaa pois käytöstä, jos yrityksen ohjelmointikäytännöissä on eroja tällaisten sääntöjen kohdalla.

3.3 Rules

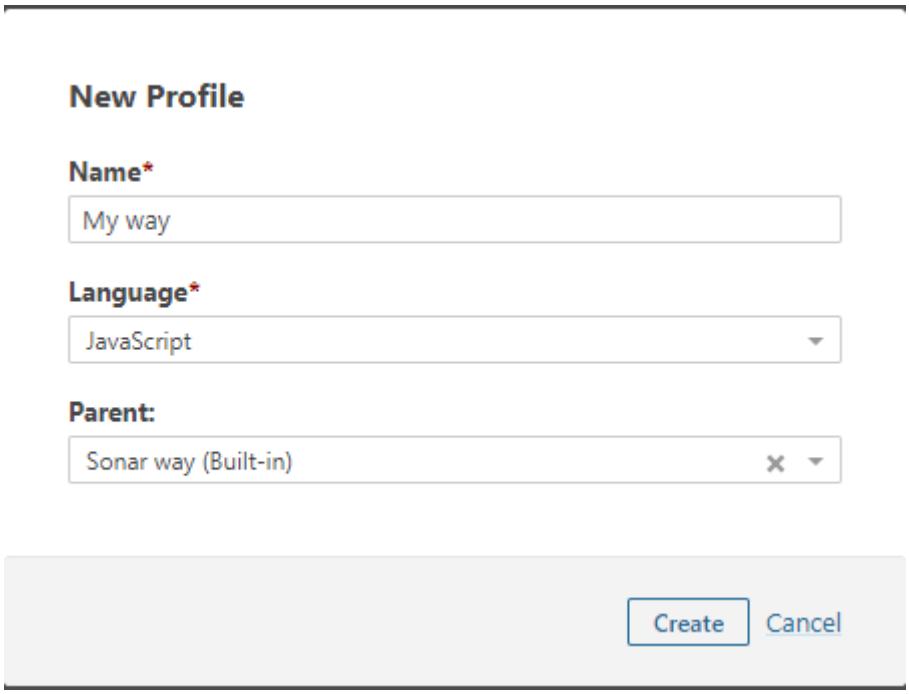
Rules sivulla pystyy tarkastelemaan nimenmukaisesti kaikkia SonarQubessa ajettavia sääntöjä. Tällä sivulla pystyy tarkastelemaan sääntöjä paremmin ja sisään kirjautuneena ylläpitäjän oikeuksilla, joitain sääntöjen tietoja pystyy muokkaamaan, kunhan sääntö on käyttäjän luomassa laatuprofiilissa (Quality profile). Sääntöjä pystyy myös suodattamaan erilaisilla suodattimilla.

Language: Ohjelmointikielen mukaan.
Type: Minkälaista ohjelmointivirhettä sääntö edustaa. Esim. Bugia tai tietoturvariskiä.
Tag: Tagit ovat tapoja joilla voidaan paikantaa tiettyjä sääntöjä helpommin.
Repository: Mistä analysoijasta sääntö tulee SonarQubeen.
Default Severity: Kuinka vakava rikkomus sääntö on.
Status: Säännön tila elämänkaaressaan.
Avaible Since: Säännön julkaisupäivä. Hyvä suodatin jos haluaa tarkastaa uuden version uudet säännöt.
Template: Pohjia helposti luotaville säännöille.
Quality Profile: Laatuprofiilien mukaan.

Säännöissä on yleensä myös esimerkkejä, mistä voi tarkastella tapaa mistä työkalu antaa virheilmoituksen ja oikeaoppista tapaa. Tämä on osoittautunut opettavaiseksi ohjelmistokehittäjän kannalta, sillä säännöissä ilmenee oikeaoppinen tai oikea tapa.

3.4 Quality Profiles

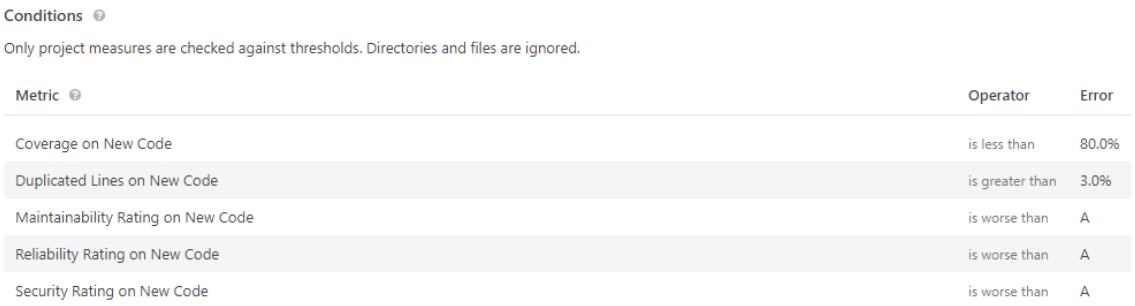
Quality Profiles sivulla näkyy kaikki laatuprofiili. Laatuprofiili on sääntöjen kokoelma, mitä voidaan käyttää projektien analysointiin. Sisäänrakennettuja laatuprofiileja ei pysty muokkaamaan, mutta halutessaan voi tehdä oman laatuprofiilin, johon voi periyttää sisäänrakennetun laatuprofiilin. Oman laatuprofiilin voi luoda klikkaamalla ”Create”-painiketta oikealla ylhäällä SonarQuben käyttöliittymässä.



KUVIO 4: Uusi JavaScript laatuprofiili jossa periytetään olemassa oleva SonarQuben ”Sonar way” laatuprofiili JavaScriptille.

3.5 Quality Gates

Quality Gates eli tässä asianyhteydessä laatuvaatimukset ovat totuusarvoja sisältäviä vaatimuksia, jotka tulee olla tosi, jotta projekti voidaan hyväksyttyä. Tämä on kätevä työkalu projektien karsintaan, jotka eivät yleisesti täytä ryhmän tai yrityksen standardeja.



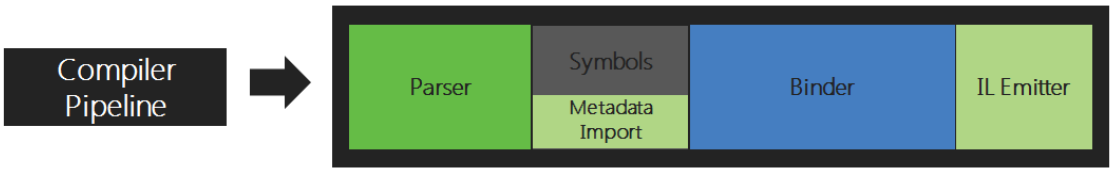
KUVIO 5: Sisäänrakennettu "Sonar way" laatu vaatimukset.

3.6 Administration

Administration eli ylläpitäjän tai järjestelmänhallitsijan sivu on nähtävillä, jos olet kirjautunut sisään käyttäjänä, joka on ylläpitäjäryhmässä. Täällä on laajemmin työkaluja, jolla muokata SonarQuben asetuksia ja sen työkaluja.

4 .NET COMPILER PLATFORM SDK

.NET Compiler Platform SDK, joka myös nimellä ”Roslyn”, on sovelluskehitystyökalu, joka sisältää avoimenlähdekoodin kääntäjiä ja koodinanalysointi ohjelmistorajapinnan (Wikipedia. 2019. Roslyn (compiler)). Analyysit kirjoitetaan C#-ohjelmointikielellä ja analyysit toimivat C#- ja Visual Basic .NET ohjelmointikielille.



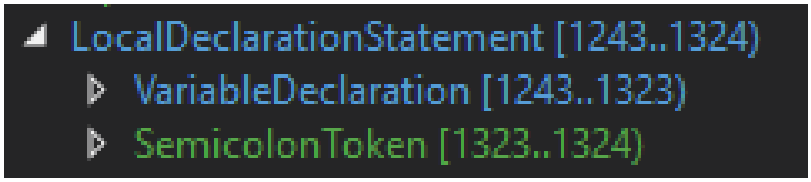
KUVIO 6. Kuva käännösprosessi (Microsoft 2017, viitattu 5.11.2019)

Kääntäjät rakentavat yksityiskohtaisen mallin sovelluksen koodista, kun ne validoivat sen syntaksia ja semantiikkaa. Ne käyttävät tätä mallia rakentaakseen ajettavan tuotoksen lähde koodista (Microsoft. 2019. The .NET Compiler Platform SDK). Kuviossa 6 nähdään kääntäjän malli, miten esim. C#- ja Visual Basic kääntäjä kääntää koodin toteuttamiskelpoiseksi tuotokseksi. Jokainen vaihe käännösprosessissa on oma erillinen komponentti. Esimerkiksi, ”Parser”-vaiheessa lähdeteksti jäsennetään ohjelmointikielen syntaksiin (Microsoft. 2017).

Roslynin ohjelmistorajapinnalla päästään käsiksi kääntäjien luomaan malliin, tällä tavalla voidaan luoda koodikeskeisiä työkaluja, mm. staattisia koodin analysoijia, jotka voidaan ajaa automaattisesti versionhallinan yhteydessä.

4.1 Roslyn Syntax Visualizer

Roslyn Syntax Visualizer on työkalu, jota käyttämällä voidaan tarkastella koodin syntaksi puuta. Se on tärkeä työkalu, joka auttaa ymmärtämään koodin mallia, jota halutaan analysoida. Työkalun avulla voidaan tarkastella mitä koodin mallien tietyissä osissa on, tällä tavalla, staattisessa analyysissä saadaan helposti tietää, onko koodi asetettujen standardien mukainen.



KUVIO 7: Paikallisen muuttujan esittely syntax visualizer -työkalun hierarkiassa.

Roslynin Syntax Visualizer työkalussa, Siniset merkkaukset syntaksi puussa ovat ”nodeja”, eli solmuja, jotka edustavat loogisia kokonaisuuksia. Solmun sisällä voi olla useampia solmuja, solmuksi lasketaan esimerkiksi muuttujan esittely. Vihreät merkkaukset puussa ovat syntaksi ”tokeneja”, eli merkkejä. Yksittäisiä sanoja, numeroita ja symboleja, joita kääntäjä on löytänyt lukiessaan tiedostoa. Merkit ovat solmujen lapsia, jotka kaiken kaikkiaan muodostavat loogisia solmuja. Punaiset merkinnät puussa tarkoittavat trivioita. Triviat ovat muuta kuin solmuja tai merkkejä. Ne ovat esimerkiksi välilyönnit, kommentit ja rivinvaihdot (Turner, A. 2019). Tämä on tärkeää tietoa staattisessa analyysissä, jos haluaa ottaa kantaa koodin muotoiluun, sillä näihin merkintöihin voidaan suoraan viitata kirjaston tapahtuman käsittelijässä, jota kutsutaan, aina kun SonarQube löytää esimerkiksi vaikkapa muuttujan, mikä kuviossa 7 esiintyy.

```
// Analyse the variable to see if its cool enough.
private void AnalyzeVariable(SyntaxNodeAnalysisContext nodeContext)
{
    // Get the variable declaration node as variable declaration syntax.
    VariableDeclaratorSyntax variable = nodeContext.Node as VariableDeclaratorSyntax;

    string variableName = variable.Identifier.ValueText;
    if (!variableName.ToLower().Contains("mycool"))
    {
        // If variable is not cool enough, report the violation.
        nodeContext.ReportDiagnostic(Diagnostic.Create(Rule, variable.GetLocation()));
    }
}
```

KUVIO 8. Esimerkissäni käytettävä metodi, joka raportoi If-lohkon sisällä rikkeestä.

Kuviossa 8 nähdään esimerkki sääntö, jossa tarkastellaan sitä, että sisältääkö muuttujan nimi ”mycool” merkkijonoa. Sääntö ei tietenkään ole käytännöllinen mutta havainnollistaa hyvin, miten koodia voidaan analysoida staattisesti. Lopuksi, jos muuttuja nimi ei ole säännön mukainen, raportoidaan rikkeestä, muutoin mennään funktion loppuun ja tullaan takaisin uuden muuttujan esittelyn kanssa.

Laavulla on käytössä M-Files ja sen mukana tulee oma ohjelmointirajapinta, jolla pystytään luomaan toiminnallisuuksia, joita ei voi luoda ilman ohjelmointia. Tässä kohtaa tarvittiin M-Files kohtaisia ohjelmointikäytäntöjä tarkastavia sääntöjä. Pääsin kehittämään eräitä sääntöjä jotka koskivat funktiokutsuja, joiden argumentteja ei voitu tarkastaa sisäänrakennetuilla vakiosäännöillä, sillä SonarQube ei ole vakiona tietoinen M-Filesin kirjastosta. Kirjoitin myös katselmointitiimin pyynnöstä sääntöjä, jotka olivat heidän mielestään hyvä lisätä. Syy tähän oli SonarQuben vakioasetusten puutteellisuuden lisäksi se, että rikkeet ovat yrityksen standardin vastaisia.

4.2 Säännön käyttöönotto

Kun sääntö on valmis, se tulee pakata jar-paketiksi. Jar-pakkaus siirretään palvelimen säännöille osoitettuun kansioon ja palvelin käynnistetään uudestaan. Usein säännöissä on versioita, jonka avulla työkalu tarkastaa, onko sääntö jo palvelimella. Kun sääntöön tulee muutoksia, tulee versionumeroa nostaa, jolloin uusi versio korvaa aikaisemman version säännöstä. Itse sääntöjen kanssa työskennellessäni, vaihdoin versionumeron nuget-paketin muodossa. Nuget-paketilla on säännön nimi ja versio. Tätä pakettia käytetään jar-pakkauksen tekemiseen, joka saa tiedot nuget-paketista. Palvelin pitää käynnistää uudestaan, sillä SonarQube lukee käytössä olevat säännöt vain käynnistyksen yhteydessä.

5 POHDINTA

Mielestäni Laavu ottaa askeleen siihen suuntaan, mihin kaikki ohjelmistoalan yritykset tulisi ottaa. Esimerkiksi, SonarQubea voidaan ajaa palvelimella, jossa sitä voidaan ajaa automaattisesti. Automatiikalla pystytään säästämään rahaa ja aikaa, jota voidaan käyttää johonkin muuhun, mitä ei voida automatisoida tai jotain, joka on kiireellisempää. SonarQube on käytännöllinen työkalu, johon pystytään lisäämään mukautettuja muokkauksia, joilla pystytään luomaan ominaisuuksia omien tarpeiden mukaan. Aluksi Laavulla oli perinteinen koodin katselmointi ennen SonarQubea, jossa katselmoija tuli tarkastamaan koodin ennen tuotantoon vientiä. Tämä on hidasta ja monesta muuttujasta johtuen epätarkkaa. Nyt kun staattinen analyysi on lisätty askeleeksi ennen katselmointia, jää katselmoijalle selkeämpi ja kertaalleen hyväksytty koodi. Jos katselmoija huomaa virheitä joista SonarQuben olisi pitänyt varoittaa, se voidaan lisätä säännöksi palvelimelle.

Laavu Solutions antoi minulle loistavan opinnäytetyön aiheen, sillä siitä oli käytännön hyötyä Laavulle ja opin itse staattisesta analyysistä, johon en ole aikaisemmin törmännyt. Aiheen lisäksi opin myös SonarQubesta. Olen varma, että pystyn suosittelemaan staattisen analyysin ottamista osaksi ohjelmistokehitysprosessia, sillä esimerkiksi SonarQube on ilmainen ja monipuolinen työkalu tätä tarkoitusta varten.

Mielestäni Laavu Solutionsin tulisi ottaa myös huomioon dynaaminen analyysi. Vaikka se ei koskekaan minun opinnäytetyöni aihetta täysin, se silti liittyy ohjelmistokehitykseen ja laadunvarmistukseen. Kun kaikki käsittelemäni asiat otetaan huomioon, mitä kävin läpi opinnäytetyössäni, on ohjelmiston laadunvarmistus mielestäni hyvällä pohjalla. Dynaaminen analyysi kuuluisi mielestäni lisätä Laavun sovelluskehitysprosessiin heti koodin katselmoinnin jälkeen, sillä koodin tulisi olla hyväksyttyä, ennenkuin voidaan siirtyä itse toiminnallisuuteen.

6 LÄHTEET

Ghahrai, A. 2018. Static Analysis vs Dynamic Analysis in Software Testing. Viitattu 31.10.2019, https://www.testingexcellence.com/static-analysis-vs-dynamic-analysis-software-testing/

Microsoft. 2019. The .NET Compiler Platform SDK. Viitattu 1.11.2019, https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/

Microsoft. 2017. Understand the .NET Compiler Platform SDK model. Viitattu 5.11.2019, https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/compiler-api-model

SmartBear Software. 2019. Why Review Code?. Viitattu 30.10.2019, https://smartbear.com/learn/code-review/why-review-code/

Hailaka, I. & Märijärvi. J. 2004. Ohjelmistotuotanto. Helsinki:Talentum.

SonarSource. 2019. Documentation. Viitattu 7.11.2019, https://docs.sonarqube.org/latest/

SonarSource. 2019. Overview. Viitattu 7.11.2019 https://docs.sonarqube.org/latest/analysis/overview/

Turner, A. 2019. C# and Visual Basic – Use Roslyn to Write a Live Code Analyzer for Your API. Viitattu 1.11.2019, https://msdn.microsoft.com/en-us/magazine/dn879356.aspx

Wikipedia. 2019. Roslyn (compiler). Viitattu 1.11.2019, https://en.wikipedia.org/wiki/Roslyn_(compiler)