Jere Junttila

Www-sovellusten ylläpidettävyyden ja tietoturvan kehittäminen hyödyntämällä MVC– ja Template metodi –suunnittelumalleja

Tietotekniikan pro gradu -tutkielma

5. marraskuuta 2018

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Jere Junttila

Yhteystiedot: Kukkumäentie 24 A43, 40620 Jyväskylä

Ohjaaja: Antti-Juhani Kaijanaho

Työn nimi: Www-sovellusten ylläpidettävyyden ja tietoturvan kehittäminen hyödyntämällä MVC– ja Template metodi –suunnittelumalleja

Title in English: Improving www-application maintainability and security with MVC and Template method design patterns.

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Sivumäärä: 49+18

Tiivistelmä: Www-sovellukset ovat nykyinfrastruktuurille tärkeitä sovelluksia, jotka ovat saatavilla ympäri maapalloa. Www-sovellukset käsittelevät paljon tietoa käyttäjistä, sisältäen myös arkaluonteista tietoa. Suuresta roolista ja kehityshuomiosta johtuen www-sovellusten taustalla olevat teknologiat ja sovellusten kehitystä ajavat ihanteet ovat jatkuvassa murroksessa. Vastatakseen näihin haasteisiin, www-sovellusten tulee olla ylläpidettäviä ja tietoturvallisia.

Tässä pro gradu -tutkielmassa perehdytään siihen, miten suunnittelumalleja hyödyntämällä voidaan helpottaa www-sovelluksia kohtaaman tämän haasteen ratkaisemista. Suunnittelumalleista syvempään tarkasteluun otetaan MVC ja template metodi -suunnittelumallit. Suunnittelumallien tuomia hyötyjä tarkastellaan tekemällä kaksi samankaltaista protoa: yhden jonka toteutuksessa ei hyödynnetä suunnittelumalleja ja toisen jonka toteutuksessa hyödynnetään suunnittelumalleja. Kehitettyjen protojen ylläpidettävyyttä ja tietoturvaa tarkastelemalla havaittiin, että suunnittelumallien hyödyntämisellä oli positiivisia vaikutuksia kehitettyyn protoon, etenkin ylläpidettävyyden kannalta.

Avainsanat: Www-sovellus, ylläpidettävyys, tietoturva, suunnittelumalli, MVC, Template metodi

Abstract: Www-applications are important applications to modern infrastructure that are available around the globe. Www-applications handle a lot of user information, including sensitive information. Technologies and ideals behind www-application development are changing constantly due to important role of www-applications and the attention towards www-application development. Www-applications must be maintainable and secure to respond to this constant challenge.

This thesis gets acquainted with tackling this challenge towards www-application development by using software design patterns. MVC and template method design patterns have been selected. Benefits of these design patterns are observed by developing two similar prototype applications. One of the prototypes is built with assist of design patterns and the other without the help of design patterns. There were positive effects observable in the developed prototype with the used maintainability and information security metrics, especially towards maintainability.

Keywords: Www-application, maintainability, information security, design pattern, MVC, Template method

Esipuhe

Gradun kirjoittaminen on ollut pitkä prosessi. Pääosan gradun kirjoittamisen ajasta olen kulkenut töissä osa-aikaisena noin kaksi päivää viikossa. Lisäksi graduprosessia sotki kirjoittamisen keskellä suoritettu ohjelmistoprojekti, jonka vaatima työaika yhdistettynä osa-aikaiseen työhön, teki graduun todellisen pysäyksen useaksi kuukaudeksi. Lopuksi viimeistely vielä venyi kokopäiväiseen työhön siirtymisen johdosta. Kiitokset kuitenkin lähipiirille ja ohjaajalle, jotka aina valoivat uskoa tekemiseen, vaikka itse en siihen aina pystynyt.

Alkuun koko työ vaikutti mahdottomalta, eikä varmuutta löytynyt edes aiheen valintaan. Aivan liian kauan aiheen parissa aikaa tuhlattuani löysin kuitenkin ensimmäisen vaihteen ja graduprosessi alkoi oikeasti. Melko pian aloituksen jälkeen tuli kuitenkin aiemmin jo mainittu sovellusprojekti, jonka johdosta graduprosessin sai aloittaa ikään kuin uudestaan. Tällä kertaa kuitenkin hieman viisastuneena. Alkuun gradu tuntu seisovan paikallaan useita kuukausia, vaikka gradun eteen yritti todella tehdä töitä. Palanen kerrallaan kokonaisuus kuitenkin alkoi hahmottua ja jossain vaiheessa vain huomasi, että suuri osa päätöksistä on tehty.

Paras päätös graduprosessissa oli kuitenkin takarajan asettaminen itselle. Asetin gradulle käytössä olevan aikani reilun kuukauden päähän siitä, kun perusasiat olivat tiedossa. Paljon pitkiä iltoja ja viikonloppuja myöhemmin gradu oli valmis. Vielä työn ohella muutamia hiomisia, mutta ei mitään suurta. Paras gradu on valmis gradu.

Jyväskylässä 4.11.2018

Jere Junttila

Termiluettelo

Www-sovellus	Internetin kautta web-selaimella käytettäviä ohjelmistoja, joiden käyttöliittymän tarjoaa käytetty web-selain. Www-sovellusta ei tule sekoittaa www-sivustoon, joka on www-sovellusta yksinkertaisempi, ja tarjoaa usein hyvin staattista tietoa.
Intraweb	Organisaation sisäisessä verkossa (intranet) toimiva web-resurssi.
Intranet	Toimii samaan tapaan kuin internet, mutta siihen pääsy on rajoitettu vain organisaation sisältä. Intranet voi olla yhdistettynä myös internettiin.
DNS	(Domain Name System) Toimii internetin nimipalvelujärjestelmänä, joka yhdistää verkkotunnukset niitä vastaaviin IP-osoitteisiin.
GoF-suunnittelumallit	Vaikutusvaltainen ohjelmistotekniikan kirja (Gamma, ym. 1995), joka esittelee suunnittelumalleja. Kirjan tekijöihin viitataan usein nimellä ”Gang of Four” (GoF) ja kirjan esittelemiin suunnittelumalleihin nimellä GoF-suunnittelumallit (GoF patterns)
Formalismi	Esittää tarkastelutavan, joka painottaa muotoa sisällön tai merkityksen sijaan. Ohjelmistokehityksessä tuttu esimerkki formalismista on UML-mallinnus.

Kuviot

Taulukot

Sisältö

1 Johdanto

Internetin ja www-sovellusten suosio kasvaa jatkuvasti ja tahot aina yksittäisistä henkilöistä kokonaisiin valtioihin ovat nykyään hyvin riippuvaisia www-sovellusten tarjoamista mahdollisuuksista (Fonseca;Vieira ja Madeira 2014). Nykyään lähes kaikki on tallennettu, saatavilla, tai vaihdettavissa internetin avulla (Fonseca;Vieira ja Madeira 2014). Useat www-sovellukset ovat saatavilla ympäri maapalloa paikoissa, joista saa yhteyden www-sovellusta suorittavaan palvelimeen. Www-sovellukset ovat tärkeä osa nykymaailman infrastruktuuria ja käsittelevät paljon käyttäjien kirjautumistietoja, maksutietoja ja muita tärkeitä tietoja (Fonseca;Vieira ja Madeira 2014). Hyviä esimerkkejä internetin ja www-sovellusten tärkeästä osuudesta nykymaailman infrastruktuuriin ovat pankkien onlinepalvelut (ja elektroninen rahaliikenne), verkkokaupat ja sähköposti (Chong, ym. 2007). Internetin tärkeyttä kuvaavat myös viime aikoina esille tulleet suuret nettihyökkäykset, niiden aiheuttamat vahingot ja myös niiden potentiaali saada aikaan suurempaakin tuhoa.

Suuresta huomiosta johtuen www-sovellusten toiminta ja samalla myös toimintaympäristö kehittyvät ja monimutkaistuvat jatkuvasti (Gupta ja Dhir 2016). Www-sovelluksissa käsiteltävät teknologiat ovat jatkuvassa murroksessa ja niihin kehitetään lisää uusiutuvaa trendiä tukevia ominaisuuksia (Gupta ja Dhir 2016). Www-sovellusten ominaisuuksia kehitetään tällä hetkellä nopealla tahdilla eri suuntiin (Gupta ja Dhir 2016). Lisäksi Fetterlyn, Manassen, Najorkin ja Wienerin (2003) suorittaman tutkimuksen mukaan suuremmat sivustot muuttuivat enemmän kuin pienet. Www-sovellusten tulee sopeutua uusien teknologioiden lisäksi myös moniin muihin ympäristön asettamiin haasteisiin (Fetterly, ym. 2003). Vastaavasti www-sovellusten kehityksen ja ylläpidon tulee kyetä sopeutumaan niihin liittyvien innovaatioiden asettamiin haasteisiin (Gupta ja Dhir 2016). Asioiden monimutkaistuminen ja laajeneminen tuovat mukanaan ongelmia, joita ei ole aina ollut yhtä tärkeää huomioida www-sovelluskehityksessä.

Internetin toiminnan avoimuudesta johtuen www-sovellukset ovat alttiina hyökkäyksille ympäri maapalloa (Fonseca;Vieira ja Madeira 2014). Www-sovelluksia vastaan kohdistuva uhkakuva onkin erittäin laaja (Fonseca;Vieira ja Madeira 2014). Lisäksi on mahdollista, että

nopeasti kehittyvästä yleisessä käytössä olevasta teknologiasta löydetään tietoturvaongelma, jonka korjaaminen vaatii laajoja muutoksia teknologiaa hyödyntävään www-sovellukseen. Kun monimutkaisuus yhdistetään hyvin laajaan uhkakuvaan, sekä muuttuviin tiedon käsittelyn lakeihin, niin ylläpidettävyys ja tietoturva nousevat erittäin tärkeiksi huomioitaviksi tekijöiksi www-sovelluksen kehityksessä jo hyvin aikaisessa vaiheessa. Ylläpidettävyyttä ja tietoturvaa on hyvä kehittää järjestelmällisesti heti kehitysprosessin alusta alkaen. Www-sovelluksissa käsitellyn datan turvallisuus on erittäin tärkeää, silti siihen ei panosteta paljoa (Chong, ym. 2007).

Tässä tutkimuksessa www-sovellusten ylläpidettävyyden ja tietoturvan sisältämiin haasteisiin haetaan ratkaisua suunnittelumallien avulla. Suunnittelumallit tarjoavat tavan hyödyntää olemassa olevaa tietoa ja taitoa ominaisuuksien kehityksessä (Koskimies ja Mikkonen 2005). Aiempaa kokemusta hyödyntämällä voidaan välttää virheitä, jotka on jo tehty. Suunnittelumalleista tähän tutkimukseen valittiin MVC- ja template metodi -suunnittelumallit.

Lähteissä esiintyvästä normista (web-sovellus, web-ohjelma, web-sivusto) poiketen tässä tutkimuksessa käytetään termiä www-sovellus. Www-sovellus termillä halutaan korostaa, että tarkastelu painottuu sovelluksiin, jotka ovat esillä internetissä maailman laajuisesti, eikä esimerkiksi rajoitetummin saatavilla oleviin intraweb-sovelluksiin. Lisäksi termillä halutaan painottaa, että tarkastelussa ovat sovellus -tyyppiset kokonaisuudet, joiden skaala on sen verran laaja, ettei niitä voida sekoittaa vain yksittäisiin www-sivuihin. Kyseisillä valinnoilla tarkennetaan tutkimuksen fokusta sellaisiin ohjelmistoihin, jotka eivät ole todella yksinkertaisia, ja näin ollen suunnittelumallien tarjoama hyöty tulee helpommin esille. Avoimesti esillä oleviin sovelluksiin keskitytään siksi, että sovelluksen tietoturva tulisi tärkeämmäksi osaksi itse sovelluksen kehitystä, eikä sitä kokonaisuudessaan ole helppo hoitaa sovelluksen ympäristössä.

Luvussa 2 esitellään ylläpidettävyys ja tietoturva omina kokonaisuuksinaan ja tarkastellaan miten ne näkyvät www-sovellusten kontekstissa. Luvussa 2 tutustutaan esittelyjen yhteydessä myös esiteltyjen ominaisuuksien mittaamiseen yleisellä tasolla. Luvussa 3 pohjustetaan suunnittelumalleja yleisesti ja perustellaan miksi suunnittelumallit voivat tarjota apua

www-sovellusten kohtaamaan haasteeseen. Luvussa 4 esitellään tässä tutkimuksessa hyödynnettävät suunnittelumallit ja perustellaan niiden valinta. Luvussa 5 esitellään tässä tutkimuksessa käytetty metodi, määritellään ominaisuuksien tarkastelussa käytetyt tarkemmat metriikat, sekä esitellään tutkimuksessa hyödynnetyt protot. Luvussa 6 esitellään metriikoiden perusteella saadut tulokset. Lopuksi luvussa 7 tehdään johtopäätöksiä siitä, oliko valituista suunnittelumalleista hyötyä www-sovellusten kehityksessä esimerkki protoissa.

2 Ylläpidettävyys ja tietoturva

Tässä luvussa esitellään ylläpidettävyys ja tietoturva ensin omina kokonaisuuksinaan ja sen jälkeen liitetään käsitteet yhteen etsien niiden välisiä riippuvaisuuksia ja yhtäläisyyksiä. Ohjelmistotuotteiden ominaisuuksia tarkastellaan yleisesti toiminnallisten ja laadullisten (ei-toiminnallisten) attribuuttien avulla. Toiminnalliset ominaisuudet kuvaavat mitä järjestelmän tulee tehdä, kun taas laadulliset attribuutit kuvaavat kuinka hyvin toiminnallisuus tulee toteuttaa tiettyjen ominaisuuksien näkökulmasta. Ylläpidettävyys ja tietoturva ovat molemmat hyvin keskeisiä laadullisia attribuutteja. Ylläpidettävyyden ja tietoturvan rooli on helppo unohtaa kehityksen aikaisessa vaiheessa. Ylläpidettävyyden ja tietoturvan laiminlyömisellä kehityksen alkuvaiheessa voi olla suuria vaikutuksia koko ohjelmiston elinkaareen.

2.1 Ylläpidettävyyden määrittely

Ylläpidettävyys on laadullinen attribuutti, joka määritellään ISO 25010 -standardissa yhdeksi kahdeksasta tärkeimmästä laatuattribuutista (ISO/IEC 25010 2011). Ylläpidettävyydellä tarkoitetaan ohjelmiston muokkaamisen helppoutta (Riaz;Mendes ja Tempero 2009). Standardin mukaan ylläpidettävyys kuvaa vaikuttavuutta ja tehokkuutta, jolla järjestelmää voidaan muokata ja ylläpitää. Toisaalta standardi kuvaa ylläpidettävyyttä ilmiönä, jota voidaan tarkastella joko luontaisena kyvykkyytenä helpottaa ylläpidon toimintaa, tai ylläpitäjien käyttäjäkokemuksena ylläpidon tavoitteen saavuttamisessa (ISO/IEC 25010 2011). Heitlagerin, Kuipersin ja Visserin (2007) mukaan ohjelmiston ylläpitoon tarvittavat resurssit riippuvat ohjelmiston ohjelmakoodin teknisestä laadusta.

ISO 25010 (2011) määrittelee ylläpidettävyyden sisältävän viisi eri ominaisuutta (property): modulaarisuus, uudelleenkäytettävyys, analysoitavuus, muokattavuus, sekä testattavuus.

Ominaisuus	**Selite**
Modulaarisuus	Kuinka suurelta osin koostuu irrallisista vaihdettavissa olevista komponenteista.
Uudelleenkäytettävyys	Kuinka hyvin osia voidaan uudelleen käyttää.
Analysoitavuus	Kuinka hyvin järjestelmä tarjoaa tietoa sen tilamuutoksista ja virheistä.
Muokattavuus	Kuinka paljon voidaan tehokkaasti muokata vaikuttamatta olennaisesti tuotteen laatuun
Testattavuus	Kuinka hyvin järjestelmälle voidaan asettaa tarkasteltavissa olevia kriteereitä.

Taulukko 1. Ylläpidettävyyden sisältämät aliominaisuudet

Modulaarisuus kuvaa, kuinka suurelta osin järjestelmä koostuu irrallisista komponenteista, jotka voidaan vaihtaa ilman suurta vaikutusta muihin komponentteihin (ISO/IEC 25010 2011). Uudelleenkäytettävyys kuvaa kuinka hyvin resursseja voidaan käyttää useammassa järjestelmässä (ISO/IEC 25010 2011). Analysoitavuus kuvaa kuinka hyvin järjestelmästä saadaan tietoa muutoksen vaikutuksista, epäonnistumisen syistä, sekä muokkausta vaativista komponenteista (ISO/IEC 25010 2011). Muokattavuus kuvaa kuinka paljon järjestelmää voidaan tehokkaasti muokata ilman että se vaikuttaa olemassa olevaan tuotteen laatuun (ISO/IEC 25010 2011). Testattavuus kuvaa kuinka hyvin järjestelmälle voidaan asettaa kriteereitä, joiden täyttymistä voidaan tarkastella (ISO/IEC 25010 2011). Tiivistetysti ylläpidettävä ohjelma on järkevästi paloiteltu kokonaisuuksiin, jotka sisältävät vähän toistensa välisiä riippuvuussuhteita ja joita on helppo tarkastella analyysin ja testien keinoin.

Ylläpito on pitkä ohjelmiston elinkaaren vaihe, jonka kustannukset kattavat suuren osan ohjelmistoon kuluvista resursseista (Riaz;Mendes ja Tempero 2009). Ylläpidettävyyden kustannuksia voidaan pienentää miettimällä ohjelman ylläpidettävyyttä kehityksen alusta lähtien (Riaz;Mendes ja Tempero 2009). Ylläpidettävyyden huomiointia on helppo pitää ohjelmiston elinkaaren alussa vain ylimääräistä tekemistä, mutta onnistunut ylläpidettävyyden

suunnittelu ja toteutus tuovat suurta hyötyä pitkässä juoksussa. Ylläpidettävyyden hyödyt tulevat esille ohjelmiston pitkän elinkaaren aikana sallimalla helpomman virheiden korjaamisen, sekä sopeutumisen uusiin vaatimuksiin.

2.1.1 Ylläpidettävyyden mittaaminen

Ylläpidettävyyden mittaamiseen voidaan käyttää kolmea eri tyyppisiä mittoja (Heitlager;Kuipers ja Visser 2007):

1. Ulkoinen mitta
2. Sisäinen mitta
3. Arvostelu

Ulkoisilla mitoilla tarkoitetaan jonkin tahon suorittamien tehtävien tehokkuutta (Heitlager;Kuipers ja Visser 2007). Esimerkiksi muokattavuutta voitaisiin mitata ulkoisella mitalla tarkastelemalla aikaa, joka menee tietyn muutoksen toteuttamiseen (Heitlager;Kuipers ja Visser 2007). Sisäisillä mittareilla puolestaan tarkoitetaan olemassa olevien toteutusten vastaavuutta tavoiteltavaan tilanteeseen (Heitlager;Kuipers ja Visser 2007). Esimerkiksi analysoitavuutta voidaan mitata tarkastelemalla mille kaikille ominaisuuksille lokitus on toteutettu suhteessa ominaisuuksiin, joiden määrittely vaatii lokituksen (Heitlager;Kuipers ja Visser 2007). Arvostelulla tarkoitetaan ohjelmiston vuorovaikutusta ympäristönsä kanssa (Heitlager;Kuipers ja Visser 2007). Arvostelut eivät ole suoria havaintoja ohjelmasta, vaan ohjelman kanssa vuorovaikuttavien tahojen havaintoja ohjelmasta (Heitlager;Kuipers ja Visser 2007).

Heitlager, Kuipers ja Visser (2007) määrittelevät mallin ylläpidettävyyden mittaamiselle. Heidän esittelemänsä malli perustuu ISO 9126 (2001) standardiin, joka on myöhemmin korvattu jo aiemmin tässä tutkimuksessa mainitulla ISO 25010 (2011) standardilla. Standardien välillä on eroja, mutta Heitlagerin, Kuipersin ja Visserin (2007) määrittelemä malli antaa mallin molemmista standardeista löytyvien analysoitavuuden, muokattavuuden ja testattavuuden mittaamiselle. Taulukko 2 liittää ylläpidettävyyden tarkastelun alla olevat aliominaisuudet varsinaisiin mitattavissa oleviin ominaisuuksiin.

Ominaisuus	**Määrittävät ominaisuudet**
Analysoitavuus	Volyymi, koodin toisto, yksikköjen koko, yksikkötestit
Muokattavuus	Yksikköjen monimutkaisuus, koodin toisto
Testattavuus	Yksikköjen monimutkaisuus, yksikköjen koko, yksikkötestit

Taulukko 2. Ylläpidettävyyttä määrittäviä ohjelmakoodin ominaisuuksia

Rychkova, ym. (2017) määrittävät mallin ylläpidettävyyden arvioimiselle mallinnustyökalulla luoduille malleille. Rychkova, ym. (2017) mukaan ohjelmistotekniikan hyväksytyt metriikat voidaan jakaa seuraaviin kategorioihin:

- kokometriikat,
- leksikaaliset metriikat (lexical metrics),
- metriikat, jotka perustuvat kontrollin kulkua seuraaviin kaavioihin (control flow graphs)
- ja muut metriikat.

Rychkova, ym. (2017) suorittaman tutkimuksen mukaan ylläpidettävyyttä onnistuneesti ennustavat Halsteadin metriikat, McCabe:n monimutkaisuus ja kokoon liittyvät metriikat. Lisäksi ylläpidettävyyttä mitataan yhdistelmämetriikalla: ylläpidettävyys indeksillä (maintainability index) (Rychkova, ym. 2017) (Heitlager;Kuipers ja Visser 2007). Ylläpidettävyys indeksi voi tarjota arvokasta tietoa kokonaisuuden ylläpidettävyydestä pelkästään ohjelmakoodin perusteella (Heitlager;Kuipers ja Visser 2007). Ylläpidettävyys indeksin ongelmana on kuitenkin sen yleisyys (Heitlager;Kuipers ja Visser 2007). Ylläpidettävyys indeksin avulla ei saada tietoa esimerkiksi siitä mihin mahdolliset ylläpidettävyyden ongelmat liittyvät (Heitlager;Kuipers ja Visser 2007). Rychkova, ym. (2017) valitsivat selvityksensä perusteella metriikoista käyttöönsä koodirivien määrän (LOC), relatiiviset muutokset (RLOC), syklomaattinen monimutkaisuus (CC), Halsteadin metriikat ja ylläpidettävyys indeksin.

Ylläpidettävyyden mittaamiseen on esitelty monia erilaisia tapoja (Riaz;Mendes ja Tempero 2009). Ylläpidettävyyttä voidaan mitata pelkästään ohjelmakoodin perusteella, mutta myös erilaisten ohjelmistolle suoritettavien toimintojen kautta (Heitlager;Kuipers ja Visser 2007).

Ylläpidettävyyttä voidaan mitata laskennallisilla kokonaisuutta mittaavilla metriikoilla, sekä hyvin pientä ylläpidettävyyden ominaisuutta mittaavien metriikoiden kautta (Heitlager;Kuipers ja Visser 2007). Yhdistelemällä erilaisia metriikoita, voidaan saavuttaa tietoa ylläpidettävyydestä niin yleisellä tasolla, kuin tietyn aliominaisuuden tasolla (Riaz;Mendes ja Tempero 2009). Koodin monimutkaisuuteen ja kokoon liittyvät metriikat esittävät yleensä tietoa tasaisesti kaikista ylläpidettävyyden aliominaisuuksista (Riaz;Mendes ja Tempero 2009).

2.2 Tietoturvan määrittely

Tietoturva tarkoittaa jonkin halutun palvelun tai tavoitteen toteuttamista tavoitetta vastustavan tahon läsnä ollessa (Zeldovich 2014). Tietoturva on kokonaisuus, joka rakennetaan kolmesta pääasiallisesta osasta:

1. toimintaperiaatteesta (policy),
2. uhkamallista (threat model),
3. sekä mekanismista (mechanism) (Zeldovich 2014).

Toimintaperiaatteella tarkoitetaan tavoitetta, eli tapaa, jolla ohjelman halutaan toimivan (Zeldovich 2014). Uhkamallilla puolestaan tarkoitetaan oletuksia, joita hyökkääjän mahdollisuuksista tehdään (Zeldovich 2014). Mekanismilla tarkoitetaan säätimiä ja osia, jotka järjestelmä tarjoaa politiikan toteuttamiseen (Zeldovich 2014). Tietoturvan eri osien kokonaisuuden määrittely rakentaa periaatteen tietoturvan pohjalle, vastustajan rajoitteen, jolle suojaus perustuu sekä teknologian, jolla se toteutetaan. Kiteytetysti voisi sanoa, että uhkamallissa esitellyt mahdollisuudet rakentuvat esitellyn politiikan ympärille ja määrittelevät mekanismin luonteen. Tavoitteena on saavuttaa tilanne, jossa vastustaja ei pysty uhkamallin puitteissa uhkaamaan ohjelman toimintaperiaatetta (Zeldovich 2014).

Tietoturvan taso määräytyy aina heikoimman lenkin mukaan (Zeldovich 2014). Tietoturva voidaan rikkoa jokaisessa mallin osassa (Zeldovich 2014). Mikäli toimintaperiaate on typerästi rakennettu, niin vastustaja voi käyttää hyväkseen periaatteen heikkoutta ja näin rikkoa tietoturvan (Zeldovich 2014). Jos uhkamallissa ei oteta huomioon kaikkia realistisia tilanteita, niin vastustaja voi rikkoa tietoturvan, vaikka kaikki toimisi suunnitellusti (Zeldovich

2014). Toisaalta jos vastustaja määritellään kaikkivoivaksi, niin tietoturvaa ei ole mahdollista edes rakentaa (Zeldovich 2014). On myös mahdollista, että toimintaperiaate on hyvä ja uhkamalli on rakennettu järkevästi, mutta toteuttava mekanismi on toteutettu puutteellisesti ja näin hyökkääjä voi rikkoa turvallisuuden toteutukseen liittyvää vikaa hyödyntäen (Zeldovich 2014).

Www-sovellusten tietoturva ja sen suurimmat uhkakuvat ovat kokeneet suuria muutoksia viime vuosien aikana (Mickens, Lecture 8: Web Security Model 2014). Nykyisellään www-sovellukset ovat hyvin alttiita hyökkäyksille (Fonseca;Vieira ja Madeira 2014). Ennen www-sovellukset olivat yksinkertaisia asiakas - palvelin arkkitehtuuriin perustuvia kokonaisuuksia, joissa palvelin suorittaa monimutkaisen toiminnan ja asiakas vain esittää tuloksen (Mickens, Lecture 8: Web Security Model 2014). Nykyään www-sovellusten uhkakuva on aiempaa laajempi, sillä selaimet kykenevät muokkaamaan näkymää ja ajamaan useita eri ohjelmointikieliä käyttäjän laitteessa (Mickens, Lecture 8: Web Security Model 2014). Selaimet ovat nykyään hyvin monimutkaisia kokonaisuuksia ja niihin lisätään kovan kilpailun johdosta paljon uusia ominaisuuksia nopealla tahdilla, ilman että varmistetaan niiden tietoturvaa kunnolla (Mickens, Lecture 8: Web Security Model 2014).

Suuri osa selainten tietoturvaa on kutsujen alkuperän tarkastelu (same-origin policy), jonka tarkoituksena on varmistaa, että verkkosivut eivät voi vaikuttaa toistensa sisältöön, ellei sitä erikseen sallita (Mickens, Lecture 8: Web Security Model 2014). Haasteita käsittelyyn aiheuttavat erilaiset kehykset ja ikkunat, joiden sisällön alkuperä vaatii poikkeusten käsittelyä (Mickens, Lecture 8: Web Security Model 2014). Alkuperän tarkastelun kannalta suuri uhka on se, että perustuessaan kutsujen alkuperään, tarkastelu perustuu DNS-järjestelmän infrastruktuuriin. Kutsujen alkuperään perustuva tietoturva ei siis pysty suojautumaan DNS-järjestelmän heikkouksiin ja altistaa selaimet hyökkäyksille kuten DNS uudelleensitominen (DNS rebind attack) (Mickens, Lecture 8: Web Security Model 2014).

Selaimen lisäksi tietoturvaongelmia on myös selaimen ja palvelimen välissä. Yleisiä ongelmia ovat odottamattomasti rakennetut kutsut, joiden yhteisvaikutusta ei ole huomioitu, käyttäjän sessiot ja autentikointi, sekä kutsuissa käytetyn protokollan heikkoudet (Mickens, Lecture 9: Securing Web Applications 2014). Tunnettuja esimerkkejä tällaisista ongelmista

ovat esimerkiksi sivustojen välinen skriptaus (Cross Site Scripting, XSS), shell shock ja Gifar-hyökkäys (Mickens, Lecture 9: Securing Web Applications 2014).

Varsinaisten selaimeen ja kutsuihin perustuvien ongelmien lisäksi www-sovellusten tietoturvassa tulee myös huomioida yleiset palvelimia koskevat tietoturvaongelmat. Www-sovellukset tyypillisesti tallentavat jotain käyttäjän syöttämää dataa ja esittävät sitä sallituille käyttäjille. Mikäli kaikkia www-sovelluksen kautta suoritettavia toimintoja ei tarkasteta kunnolla, niin voidaan altistua hyökkäyksille, kuten SQL injektio (SQLi) (Fonseca;Vieira ja Madeira 2014).

Käyttäjän virheet ja huolimattomuus muodostavat osion, johon ei juuri voida vaikuttaa. Tietoturvan ylläpitämiseksi suoritetut toimenpiteet ovat monesti käyttäjän rikottavissa (Mickens, Lecture 8: Web Security Model 2014). Rakennettu tietoturva rikkoutuu, kun käyttäjä päättää klikata jotain vastustajan linkkiä, jonka avulla vastustaja pääsee suojausten ohi (Mickens, Lecture 8: Web Security Model 2014). Käyttäjä voi myös vahingossa tai epähuomiossa siirtyä johonkin vastustajan osoitteeseen, mikäli ei tarkista DNS-osoitteen oikeellisuutta (Mickens, Lecture 8: Web Security Model 2014).

Kokonaisuutena www-sovellusten tietoturva on siis hyvin monimutkainen ja kompleksinen kokonaisuus, jonka perusteita on nyt hieman avattu. Www-sovellusten tietoturva voidaan rikkoa selaimessa, palvelimessa, selaimen ja palvelimen välillä suoritettavilla kutsuilla, käyttäjän suorittamasta virheestä tai eri osioiden yhteisvaikutuksella (Mickens 2014).

2.2.1 Tietoturvan mittaaminen

Tietoturvan mittaaminen ohjelmakoodista on hyvin monimutkainen prosessi ja jättää helposti tarkastamattomia aukkoja. Olemassa olevan www-sovelluksen tietoturvaa voidaan tarkastella altistamalla kehitetty sovellus www-sovelluksiin kohdistuville haavoittuvuuksille ja hyökkäyksille (Fonseca;Vieira ja Madeira 2014). Haavoittuvuuksia ja hyökkäyksiä hyväksikäyttäen tarkastellaan, saadaanko kohteena oleva www-sovellus siirrettyä virhetilanteeseen (Fonseca;Vieira ja Madeira 2014). Käytännössä haavoittuvuudella tarkoitetaan heikkoutta, jota hyödyntämällä saadaan aikaan harmia, mutta jonka läsnäolo ei itsessään aiheuta

ongelmia (Fonseca;Vieira ja Madeira 2014). Kyseistä metodia voidaan käyttää monentyyppisten haavoittuvuuksien etsimiseen, mutta pääasiassa www-sovelluksille sitä hyödynnetään SQL injektion (SQLi) ja Cross site scripting (XSS) tarkasteluun (Fonseca;Vieira ja Madeira 2014).

Tietoturvan tarkastelun helpottamiseksi on luotu monia erilaisia sovelluksia, joiden tarkoituksena on tarkastella ohjelmistoa tietoturvan näkökulmasta. Tarkastelu voidaan suorittaa suoraan ohjelmakoodia tarkastelemalla, www-sovellukselle kutsuja lähettämällä tai jopa hakukoneita hyödyntäen (OWASP 2017). Ohjelmakoodiin perustuvaa tarkastelua kutsutaan yleisesti staattiseksi analyysiksi, ohjelman ajoon perustuvaa tarkastelua puolestaan dynaamiseksi analyysiksi (Fonseca;Vieira ja Madeira 2014).

Staattinen analyysi vaatii työkalulta tietämystä käytetystä ohjelmointikielestä, tarkasteltavista heikkouksista sekä käytetystä kehitysympäristöstä (OWASP 2017). Staattisen analyysin vahvuus on, että sen avulla voidaan tarkastella tietoturvaan liittyviä perusasioita helposti ja kehittäjille saadaan suuntaa potentiaalisista ongelmakohdista jo aikaisessa vaiheessa (OWASP 2017). Staattisen analyysin heikkoutena on se, että monia hyökkäyksiä on vaikea tunnistaa suoraan ohjelmakoodista (OWASP 2017) (Fonseca;Vieira ja Madeira 2014). Staattisen analyysin työkalut antavat paljon virheellisiä ongelmatulkintoja ja niillä on vaikea todistaa, että löydetty ongelma on oikeasti haavoittuvuus (OWASP 2017).

Dynaamisen analyysin avulla voidaan tarkastella hallitussa ympäristössä, miten kehitetty sovellus käyttäytyy tunnettujen hyökkäysten kohteena (Fonseca;Vieira ja Madeira 2014). Ajoaikaisen monitoroinnin ja tietokannan tarkastelun avulla voidaan selvittää, olivatko hyökkäykset onnistuneita, vai onnistuiko sovellus selviytymään hyökkäyksestä ilman virhetilanteita ja tietovuotoja (Fonseca;Vieira ja Madeira 2014).

Paras lopputulos www-sovelluksen tietoturvan tarkasteluun saadaan hyödyntämällä sekä staattista analyysia, että dynaamista analyysia (Fonseca;Vieira ja Madeira 2014). Näin mahdollistetaan tietoturvan potentiaalisten ongelmien tarkastelu alkaen jo aikaisessa vaiheessa kuitenkaan unohtamatta kokonaisuuden testausta.

2.3 Ylläpidettävyyden ja tietoturvan yhteys

Www-sovellusten ylläpidettävyys ja tietoturva rakentuvat monen eri tekijän yhteisvaikutuksesta. Tietoturvaa esiteltäessä tietoturvan katsottiin rakentuvan kolmesta eri palasesta: toimintaperiaatteesta, uhkamallista ja mekanismista (Zeldovich 2014). Toisaalta ylläpidettävyyttä määriteltäessä ylläpidettävyys kuvattiin vaikuttavuutena ja tehokkuutena, jolla järjestelmää voidaan muokata ja ylläpitää (ISO/IEC 25010 2011). Kuten johdannossa todettiin, www-sovellusten toimintaperiaatteet, toimintaympäristö ja www-sovelluksia ajavat teknologiat kehittyvät jatkuvasti (Gupta ja Dhir 2016). Nämä muutokset viestivät, että www-sovellusten toimintaperiaatteet ja mekanismit kehittyvät ja muuttuvat jatkuvasti. Www-sovellusten tietoturvan ylläpidon täytyy pystyä muuttumaan ympäristön asettamien vaatimusten mukana.

Lisäksi on tärkeää huomioida, että tietoturva määräytyy heikoimman lenkin mukaan (Zeldovich 2014). Ylläpidettävyys sisältää aliominaisuudet: uudelleenkäytettävyys, analysoitavuus ja testattavuus (ISO/IEC 25010 2011). Kun ohjelmisto on helposti analysoitavissa ja eri osa-alueet ovat helposti testattavissa, niin ohjelmiston muutosten yhteydessä on helpompaa pitää huoli, ettei ohjelmistoon esitellä uusia heikkoja lenkkejä. Lisäksi joustava koodin uudelleenkäyttö mahdollistaa tietoturvalliseksi todetun koodin hyödyntäminen laajemmin ohjelmistossa.

Www-sovellusten tietoturvan ylläpitäminen on iteratiivinen prosessi, joka kehittyy järjestelmän elinkaaren aikana ympäristön asettamien uusien haasteiden, sekä järjestelmästä löydettyjen haasteiden kautta. Mikäli järjestelmän ylläpidettävyys on kunnossa, niin tietoturvan ylläpitämiseen kuluu vähemmän resursseja ja mahdollisesti voidaan myös vähentää ylläpidon virheherkkyyttä (Riaz;Mendes ja Tempero 2009).

3 Suunnittelumallien hyödyntäminen

Tässä kappaleessa perehdytään suunnittelumallien ominaisuuksiin. Mitä ovat suunnittelumallit ja miksi ne ovat tärkeitä nykypäivän www-sovelluksille. Suunnittelumallit ovat jo pitkään vaikuttaneet ohjelmistosuunnitteluun ja ne ovat osoittaneet hyödyllisyytensä teorian lisäksi myös todellisessa elämässä (Koskimies ja Mikkonen 2005).

3.1 Mitä ovat suunnittelumallit

Suunnittelumallit keskittyvät tiettyyn ohjelmistokehityksen suunnitteluhaasteeseen tai ongelmaan (Gamma, ym. 1995). Malli kuvaa milloin sitä voidaan soveltaa, millaisten suunnittelurajoitteiden kanssa se on sovellettavissa, sekä sen käytön seuraukset (Gamma, ym. 1995). Kukin malli esittelee ne attribuutit, joiden osalta se tuo parannuksia ja toisaalta myös ne attribuutit, jotka jäävät heikommaksi mallia hyödynnettäessä (Gamma, ym. 1995).

Pohjimmiltaan suunnittelumallit ovat keino jakaa jo olemassa olevaa kokemusta ja ammattilaisten mielipiteitä (Zhang ja Budgen 2012) (Koskimies ja Mikkonen 2005). Mestarien töistä parhaiden puolien hyödyntäminen muiden työssä on vanhempi ajatus kuin tietotekniset suunnittelumallit (Koskimies ja Mikkonen 2005). Suunnittelumallit ovat tiettyyn tilanteeseen soveltuvia hyväksi havaittuja ratkaisujen kuvauksia ohjelmiston suunnittelua koskeviin ongelmiin (Koskimies ja Mikkonen 2005). Suunnittelumallien käyttöönotto ei edellytä tiettyä teknologiaa, suunnittelumenetelmää tai ohjelmointikieltä (Koskimies ja Mikkonen 2005).

Suunnittelumallien idea on siis hyötyä jo aiemmin tehdyistä projekteista ja ottaa oppia niissä tehdyistä ratkaisuista, sen sijaan että tutkittaisiin empiirisesti vastaavia tekijöitä aineiston perusteella (Koskimies ja Mikkonen 2005). Monesta suuresta ohjelmistosta on todella vaikeaa tai mahdotonta tehdä empiiristä tutkimusta, koska rakenteita ja prosessia ei ole kuvattu kunnolla, tai yritykset eivät halua julkistaa varsinaista toteutusta. Lisäksi ohjelmistojen suunnittelu (design) on huonosti jäsennetty ongelma, jolle monitulkintaiset määrittelyt ovat luonteenomaisia (Zhang ja Budgen 2012). Suunnittelumallien avulla voidaan hyödyntää hyviä ratkaisuja ilman, että alkuperäiseen työhön perehtyminen olisi mahdollista (Koskimies ja

Mikkonen 2005). Lisäksi suunnittelumallien avulla voidaan todeta ratkaisu toimivaksi käytännön sovelluksessa, vaikka empiirisen tutkimuksen monitulkintaista ominaisuutta ei vältämättä voitaisi määritellä ratkaistuksi ja tavoitetta saavutetuksi.

Koskimiehen ja Mikkosen (2005) mukaan suunnittelumallit koostuvat kolmesta olennaisesta osasta:

1. Ongelmasta, joka on yleinen suunnitteluongelma ja ilmenee useissa järjestelmissä.
2. Ongelmayhteydestä, joka kertoo, millaisissa tilanteissa suunnittelumalli on sovellettavissa sekä mahdollisesti ratkaisulle asetetuista vaatimuksista.
3. ja ratkaisusta, joka on kuvattu yleisesti ja on kuvattavissa yleisesti tunnetuilla formalismeilla.

Suunnittelumalleista on hyvä huomata, että vaikka tarkoituksena on esitellä yleisellä tasolla ratkaisu johonkin aiemmin ratkaistuun ongelmaan, niin ongelman laajuuteen ja ongelmayhteyden yleisyyteen ei oteta kantaa. Suunnittelumallien yleisyys- ja koetelluusvaatimusta ei tarvitse määritellä tiukasti, vaan voidaan koota esimerkiksi yrityksen omia hyviksi osoittautuneita ratkaisuja suunnittelumallien muotoon (Koskimies ja Mikkonen 2005). Tästä piirteestä johtuen käytännön työhön voidaan hakea apua yleisistä yhteisössä hyviksi todetuista, sekä mahdollisista yritysten sisäisistä suunnittelumalleista.

Eräs määritelmä suunnittelumalleille: *”Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”* (Zhang ja Budgen 2012) (Gamma, ym. 1995). Suunnittelumallien tarkoituksena on siis ratkaista toistuvia ongelmia siten, että varsinaista toteutusta ei anneta, vaan annetaan keinoja oikeantyyppisen ratkaisun tekemiseen kullekin omassa kontekstissaan. Suunnittelumalli voi esimerkiksi kertoa järjestelmälle kokonaisarkkitehtuurin tukipylväät ja esittää tavan jakaa asioita, mutta ei ota kantaa siihen kuinka kyseinen arkkitehtuuri rakennetaan.

Suunnittelumallien esittelytapa on tärkeä, sillä ideana on tarjota tietoa kehittäjälle, jonka tietotaito kyseisellä alueella on vähäisempää kuin suunnittelumallin tekijällä (Zhang ja Budgen 2012). Oikein tehtynä suunnittelumallit sisältävät raportin ja analyysin suunnittelumallin

käytöstä, eikä vain suunnittelumallin tunnistamista (identification) ja dokumentointia (Zhang ja Budgen 2012).

Vaikka suunnittelumalleilla viitataan jollakin tasolla olemassa olevan tiedon tai ammattitaidon hyödyntämiseen, niin siihen liittyvässä tutkimuksessa on löydettävissä painotus ja vivahde eroja. Muun muassa Zhang ja Budgen (2012) painottavat suunnittelumallien uudelleenkäytettävyyttä ja konkreettisen ratkaisun puuttumista malleista. Sen sijaan Koskimies ja Mikkonen (2005) painottavat sitä, että kyseessä on tiettyyn tilanteeseen jo käytännössä hyväksi todettu ratkaisu, joka esitetään tietyssä muodossa.

3.1.1 Ohjelmistoarkkitehtuurit

Suunnittelumallit tukevat ohjelmistoarkkitehtuuria (Koskimies ja Mikkonen 2005). Ymmärtääkseen mitä hyötyä edellä mainitulla tuella on ohjelmistolle, täytyy tietää mitä ohjelmistoarkkitehtuuri tarkoittaa.

Koskimiehen ja Mikkosen (2005) mukaan IEEE:n arkkitehtuurien kuvaamista koskeva standardi määrittelee ohjelmistoarkkitehtuurin järjestelmän perusorganisaatioksi, joka sisältää järjestelmän osat, niiden keskinäiset suhteet ja niiden suhteet ympäristöön sekä periaatteet, jotka ohjaavat järjestelmän suunnittelua ja evoluutiota. Ohjelmistoarkkitehtuuri sisältää tietoa sekä ohjelmiston rakenteesta, että ohjelmiston käyttäytymisestä (Koskimies ja Mikkonen 2005). Staattisen rakenteen lisäksi ohjelmistoarkkitehtuuri liittyy myös suorituksen aikaisiin rakenteisiin, kuten dynaamiset oliorakenteet (Koskimies ja Mikkonen 2005). Yksinkertaistettuna voinee sanoa, että ohjelmistoarkkitehtuurilla tarkoitetaan ohjenuoraa, joka tarjoaa abstraktin näkymän osiin, joista ohjelmisto koostuu ja tavan, joilla osat kommunikoivat toisilleen ja mahdollisille muille ohjelmistoille, sekä tiedon siitä miten ohjelmiston osia suunnitellaan ja kehitetään.

Ohjelmistoarkkitehtuuri määrittelee ohjelmiston ytimen, joka ei olennaisesti muutu ohjelmiston elinkaaren aikana (Koskimies ja Mikkonen 2005). Ohjelmiston ydin on hyvin monimuotoinen asia, joten sitä tarkastellaan useista eri näkökulmista erillisissä dokumenteissa. Arkkitehtuuria esittelevien dokumenttien tarkoituksena on esitellä kuvaus ja perustelu jollekin arkkitehtuurin osakokonaisuudelle (Koskimies ja Mikkonen 2005). Ilman kunnollista

arkkitehtuurin dokumentointia eri ihmiset voivat tehdä erilaisia tulkintoja siitä mikä arkkitehtuuri on ja mitä siihen kuuluu (Koskimies ja Mikkonen 2005).

Ohjelmistoarkkitehtuurin tehtävänä on tarjota näkymiä, joiden avulla eri sidosryhmät voidaan saattaa samalle aaltopituudelle kyseessä olevasta järjestelmästä (Koskimies ja Mikkonen 2005). Arkkitehtuuria määrittelevien dokumenttien avulla järjestelmästä voidaan puhua kullekin sidosryhmälle sopivalla abstraktiotasolla ja näin välttää eri tahojen välisiä väärinkäsityksiä.

3.2 Miksi suunnittelumallit

Www-sovellusten yleinen trendi on jo pitkään siirtynyt kohti monimutkaisempia työpöytäsovelluksia muistuttavaan suuntaan (Gupta ja Dhir 2016). Tästä johtuen www-sovellusten toteutuksessa käytettävät työkalut ja teknologiat kehittyvät nopeaa tahtia. Suunnittelumalleista voi olla hyötyä kehityksen mukana pysymiseen, sillä esimerkiksi Zhangin ja Budgenin (2012) suorittama tutkimus tukee näkemystä, jonka mukaan suunnittelumallit voivat tukea ohjelmiston ylläpitoa, mikäli suunnittelumalli on niin dokumentoitu. Lisäksi (Hegedűs, ym. 2012) tekemän tutkimuksen mukaan tarkastelluilla suunnittelumalleilla oli positiivisia vaikutuksia kohteena olleen järjestelmän ylläpidettävyyteen.

Suunnittelumallien käyttö tarjoaa välineet, joiden avulla voidaan hyödyntää jo aiemmin hyväksi todettuja ratkaisuja omassa projektissa (Koskimies ja Mikkonen 2005). Näin ollen suunnittelumallien käyttö luo turvaa tietyn tyyppisen ratkaisun luomiseksi omassa projektissa. Lisäksi suunnittelumallit tarjoavat etenkin vähemmän kokeneelle kehittäjälle keinoja pohtia oikeita ratkaisuja arkkitehtuuria suunniteltaessa (Koskimies ja Mikkonen 2005). Suunnittelumalleista on hyötyä myös kokeneelle kehittäjälle, sillä ne tarjoavat ohjelmistotekniikan kehitystrendin mukaisia uusia korkean tason rakenneabstraktioita (Koskimies ja Mikkonen 2005). Lisäksi suunnittelumallit tarjoavat kehityksessä toistuville rakenteille nimen ja esitysmuodon (Koskimies ja Mikkonen 2005). Hyvin valittuna suunnittelumallit antavat keinon lähteä liikkeelle ratkaisuilla, jotka on jo aiemmin todettu toimivaksi oman tyylisessä projektissa, joka pyrki vastaavantyyppiseen tavoitteeseen.

Monet kokeneet kehittäjät arvostavat suunnittelumalleja toistuvien ongelmien ja ratkaisujen kategorisoimiseen ja ratkaisemiseen (Zhang ja Budgen 2012). Suunnittelumallit tarjoavat myös keinon kommunikoida monimutkaisistakin kokonaisuuksista eri suunnittelijoiden välillä (Koskimies ja Mikkonen 2005). Kokonaisuuden selittämisen sijaan suunnittelijat voivat välittää suuren määrän informaatiota (rakenne, ratkaisut, perustelut) pelkästään viittaamalla yleisesti tunnettuun suunnittelumalliin (Koskimies ja Mikkonen 2005).

Suunnittelumallit ovat huomattavassa asemassa ohjelmiston kehityksessä, mutta niistä ei ole tehty kuin hyvin rajoittuneita tutkimuksia (Zhang ja Budgen 2012). Tutkimusten perusteella suunnittelumallit kuitenkin auttavat kehittämään seuraavia asioita (Zhang ja Budgen 2012):

1. Ohjelmoijan tuottavuus ja tuotettavan ohjelmiston laatu.
2. Aloittelijoiden design taidot.
3. Rohkaisevat kehittämään parhaiden käytänteiden mukaisesti.
4. Parantaa kommunikaatiota.

Suunnittelumallien avulla luodun ohjelmiston arkkitehtuurin dokumentointi tarjoaa korkean abstraktiotason näkymän ohjelmiston rakenteeseen (Koskimies ja Mikkonen 2005). Näkymien avulla voidaan perehdyttää uusia tekijöitä, esitellä kokonaisuuksia eri sidosryhmille ja yhtenäistää ohjelman kehitystä koko elinkaaren ajan.

3.3 Suunnittelumallien kritiikkiä

Tutkimuksessa keskitytään hyvin tiettyihin laatuominaisuuksiin ja suunnittelumalleilla pyritään juuri näiden laatuominaisuuksien kehittämiseen. On kuitenkin tärkeää huomata, että suunnittelumallit keskittyvät aina jonkin ohjelmiston laatuominaisuuden edistämiseen (Koskimies ja Mikkonen 2005). Tavallisesti tämä johtaa jonkin muun laatuominaisuuden heikkenemiseen (Koskimies ja Mikkonen 2005). Esimerkiksi tyypilliset GoF-suunnittelumallit edistävät ohjelmiston muunneltavuutta, mutta heikentävät ohjelmiston suorituskykyä (Koskimies ja Mikkonen 2005). Usein suorituskyvyn heikkeneminen on kuitenkin hyvin marginaalista, eikä sillä ole suurempaa merkitystä, ellei kyseessä ole kriittinen reaaliaikajärjestelmä, tai sulautettu järjestelmä (Koskimies ja Mikkonen 2005). Lisäksi yleensä suunnittelumallit monimutkaistavat tuotetun ohjelman arkkitehtuuria lisäämällä komponenttien,

luokkien ja rajapintojen määrää (Koskimies ja Mikkonen 2005). Ennen suunnittelumallien hyödyntämistä onkin hyvä miettiä, onko projekti tarpeeksi kompleksinen, jotta suunnittelumalleja kannattaisi hyödyntää. Lisäksi suunnittelumalleja hyödyntäessä tulee tarkastella myös ominaisuuksia, joihin suunnittelumalli saattaa vaikuttaa negatiivisesti (Koskimies ja Mikkonen 2005).

4 Hyödynnettävät suunnittelumallit

Suunnittelumallit ohjaavat kehitystä, mutta yksi suunnittelumalli on monessa tapauksessa riittämätön. Esimerkiksi MVC-suunnittelumalli on hyvin yleisesti www-sovellusten kehityksessä käytetty malli. MVC-suunnittelumalli ohjaa kutsun kulkua erillisiin loogisiin palasiin, mutta ei ota kantaa siihen, miten kukin looginen palanen rakentuu. Tästä syystä MVC-suunnittelumallin lisäksi on hyvä hyödyntää myös muita suunnittelumalleja.

Tässä tutkielmassa hyödynnetään MVC-suunnittelumallia, jotta loogiset kokonaisuudet saadaan jaettua www-sovellukselle sopivalla tavalla. MVC-suunnittelumallin lisäksi hyödynnetään template metodi -suunnittelumallia ratkaisemaan malliluokkien toteutuksen monimutkaisuutta ja pyritään pienentämään niiden toteutuksen virheherkkyyttä.

4.1 MVC-suunnittelumalli

MVC-malli tulee sanoista malli (model), näkymä (view), ja ohjain (controller). Mallit ovat ohjelman objekteja, näkymät ovat tapa esittää tieto ruudulla ja ohjain määrittelee miten käyttäjä rajapinta reagoi käyttäjän syötteisiin (Gamma, ym. 1995). MVC-mallin tarkoituksena on erotella nämä osiot parantaakseen joustavuutta ja uudelleenkäytettävyyttä (Gamma, ym. 1995). Alkuperäisen MVC-mallin mukaan näkymät olivat suoraan yhteydessä malleihin (Gamma, ym. 1995). MVC-mallista on myöhemmin kehittynyt erillinen versio, jossa mallien ja näkymien välinen yhteys estetään uudelleenkäytettävyyden ja modulaarisuuden parantamiseksi.

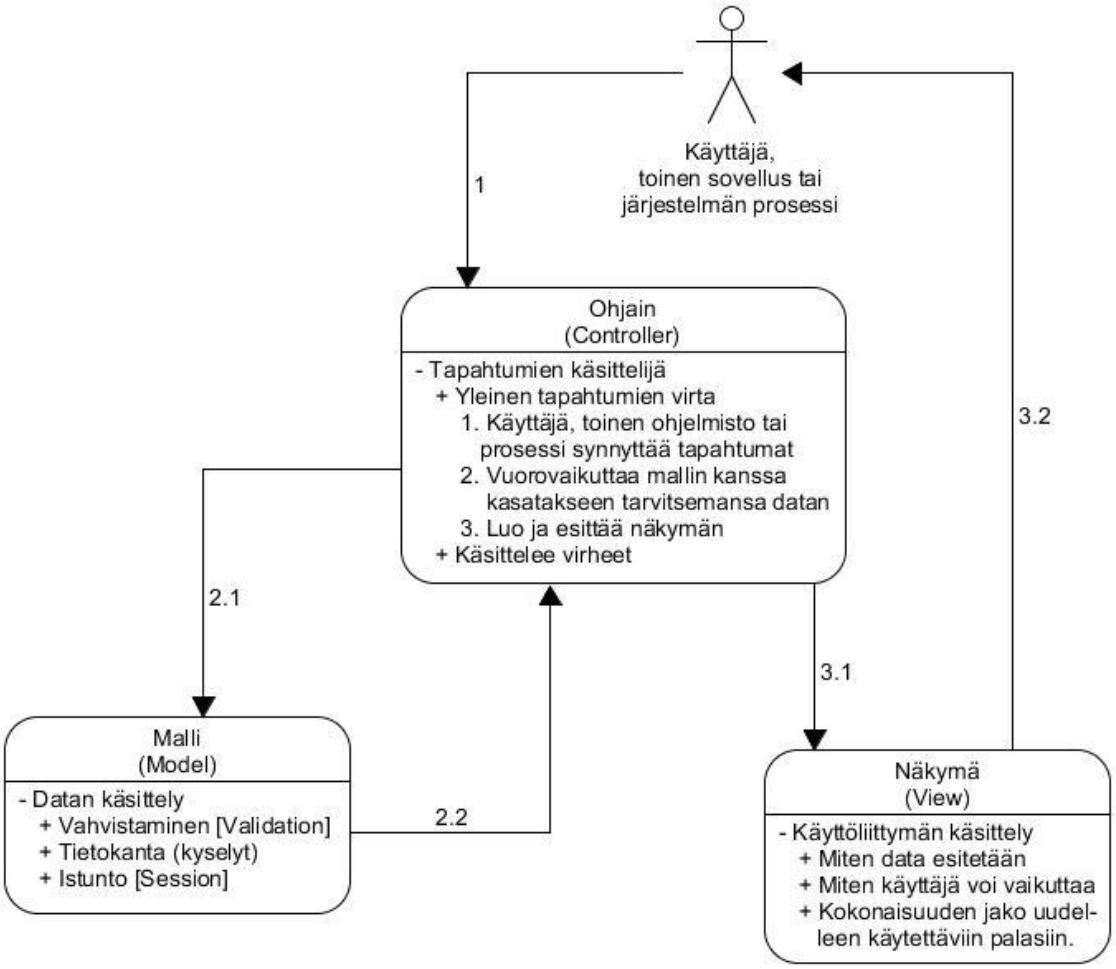
MVC-malli on hyvin vanha konsepti. Alun perin kyseessä oli vain ajatus luoda silta käyttäjän ajatusmallin ja digitaalisen mallin väliin (Pop ja Altar 2014). Myöhemmin ajatukseen tuli mukaan ajatus modulaarisuudesta ja hyödyistä, jotka saavutetaan, kun eri osat eristetään toisistaan (Pop ja Altar 2014). Tässä vaiheessa malli jaettiin kolmeen kategoriaan: pääohjelmaan, datan esitykseen ja käyttäjän interaktioon (Pop ja Altar 2014).

Suunnittelumallin pääajatus on säilynyt samankaltaisena, mutta mallin pääpaino on muuttunut (Pop ja Altar 2014). Nyky-ajatuksen mukaan MVC-suunnittelumallin ratkaisemana ongelmana on tarve erottaa datan esitys sen tallentamisesta ja niitä käsittelevästä logiikasta

(Pop ja Altar 2014). Ongelmayhteys ovat sovellukset, joissa käsitellään datan tallentamista, esittämistä ja jonkinlaista logiikkaa, sellaisella tavalla, jolloin niiden erottamisesta voidaan saada hyötyä. Yleisemmällä tasolla MVC- suunnittelumallia kannattaa hyödyntää, kun loogisten osasten palasia halutaan käyttää uudelleen tai kehittää erillisinä kokonaisuuksinaan.

MVC-suunnittelumallin avulla voidaan siis vähentää koodin sisäisiä riippuvaisuuksia (Pop ja Altar 2014). Sisäisten riippuvuuksien väheneminen puolestaan mahdollistaa www-sovelluksen eri osioiden rinnakkaisen kehittämisen, sekä tehostaa koodin uudelleenkäyttöä (Pop ja Altar 2014).

Kuvio 1 sisältää MVC-suunnittelumallin loogisten kokonaisuuksien erottelun, niiden välisen kommunikoinnin sekä osoittaa perusosasten sisältämät vastuut kokonaisuuden hallinnasta. Esimerkiksi datan vahvistamisen voisi toteuttaa missä tahansa MVC-mallin osasessa, mutta MVC-suunnittelumallin mukaan se kuuluu mallin tehtäväksi (Pop ja Altar 2014). Ohjain siirtää mahdollisen virhetiedon näkymälle ja näkymä puolestaan esittää virheen ottamatta kantaa siihen miksi data ei kelpaa.



Kuvio 1. Kuvaus MVC-suunnittelumallin periaatteista.

4.1.1 Miksi MVC-suunnittelumalli

Monimutkaistuessaan www-sovellukset tulevat yhä vaikeammaksi hallita (Pop ja Altar 2014). Kun muodostetaan dynaamisia sivuja ja kokonaisuus on hyvin laaja, niin datan esitystä suunniteltaessa ja toteutettaessa on vaikeaa huomioida sen tekemiseen liittyvää logiikkaa ja sitä, miten dataa oikeasti käsitellään (Pop ja Altar 2014). Yleisesti monimutkaisen järjestelmän osioita kehittävät myös useat ihmiset, jolloin on tärkeää sopia loogisista kokonaisuuksista ja vastuista, mitkä kullekin osaselle kuuluvat (Pop ja Altar 2014). Esimerkiksi yksi kehittäjä voi olla erikoistunut nykyaikaisen HTML- ja CSS-koodin kirjoituksessa, kun toinen taas PHP-koodauksessa ja kolmas SQL-kielestä. Loogiset osat erittelemällä voidaan

mahdollistaa eri tyyppisen osien kehitys erillisinä kokonaisuuksinaan (Pop ja Altar 2014). Riittää kun määritellään, mille osalle mikäkin vastuu kuuluu ja missä muodossa tieto siirretään osalta toiselle.

Lisäksi MVC-malli sopii todella hyvin www-sovellusten kehitykseen, sillä www-sovelluskehityksessä usein yhdistetään useita teknologioita, jotka voidaan jakaa johonkin MVC-mallin esittelemistä kategorioista. MVC-malli mahdollistaa myös helposti erilaisten näkymien palauttamisen erilaisille kutsun suorittajille.

Tämän tutkimuksen protojen kehityksessä hyödynnetään ohjelmistokehystä. Ohjelmistokehyksen hyödyntäminen mahdollistaa sen, että suuri osa työajasta ei mene reititykseen ja pyyntöjen käsittelyjen toteutukseen (Gupta ja Dhir 2016). Useat web-ohjelmistokehykset toteuttavat MVC-mallin. Näitä ohjelmistokehyksiä ovat mm. ASP.NET MVC (https://www.asp.net/mvc), Codeigniter (https://codeigniter.com/), AngularJS (https://angularjs.org/), Rails (http://rubyonrails.org/) ja Laravel (https://laravel.com/). Monissa (luetelluissa) web-ohjelmistokehyksissä MVC-malli on hyvin suuressa osassa kehitystä. Luetellut ohjelmistokehykset on kasattu GitHubin tekemästä listauksesta (Github 2017), sekä google-hakukoneella tehdystä hausta hakusanoilla ”web+framework+popular”.

MVC-malli on yleisyydestään, ongelmaan sopivuudestaan ja ohjelmistokehysten tuesta johtuen valittu toteutettavan rakenteen pohjaksi. Myöhemmin esitettyjä suunnittelumalleja hyödynnetään MVC-mallin mukaisten osien toteutukseen.

4.1.2 MVC-suunnittelumalliin kohdistuvaa kritiikkiä

MVC-suunnittelumalli ja sen periaatteet ovat muuttuneet ajan mukana (Pop ja Altar 2014). MVC-suunnittelumallista on myös liikkeellä monia erilaisia variaatioita, joiden välillä saattaa olla suuriakin eroja tavoissa kehittää eri osaset ja kommunikoida niiden välillä (Pop ja Altar 2014) (Koskimies ja Mikkonen 2005). Yleensä johonkin suunnittelumalliin tutustunut ammattilainen voi ymmärtää paljon sovelluksen toimintaperiaatteista, vain kun kuulee sovelluksen toteuttaman suunnittelumallin nimen. MVC-suunnittelumallia hyödynnettäessä tulee tarkentaa toteutusta ja osasten vastuunjakoa. MVC-suunnittelumalli on eräänlainen

muotisana ja internetistä on löydettävissä runsaasti materiaalia, jonka mukaan MVC-suunnittelumalli ei edes täytä suunnittelumallin vaatimuksia.

MVC-suunnittelumallin avoimuudesta johtuen väärin toteutettuna eri osasten väliin voidaan luoda suuria riippuvuussuhteita (Koskimies ja Mikkonen 2005). Jos näin tapahtuu, niin MVC-suunnittelumalli luo vain ylimääräistä kompleksisuutta ohjelmistoon. Olio-ohjelmointikehityksessä MVC-suunnittelumallin toteuttamisen myötä toteutettujen luokkien määrä kasvaa (Koskimies ja Mikkonen 2005). Hyvin toteutettuna luokkien määrän kasvu ei kuitenkaan kasvata kompleksisuutta, vaan jakaa kokonaisuutta.

4.2 Template metodi -suunnittelumalli

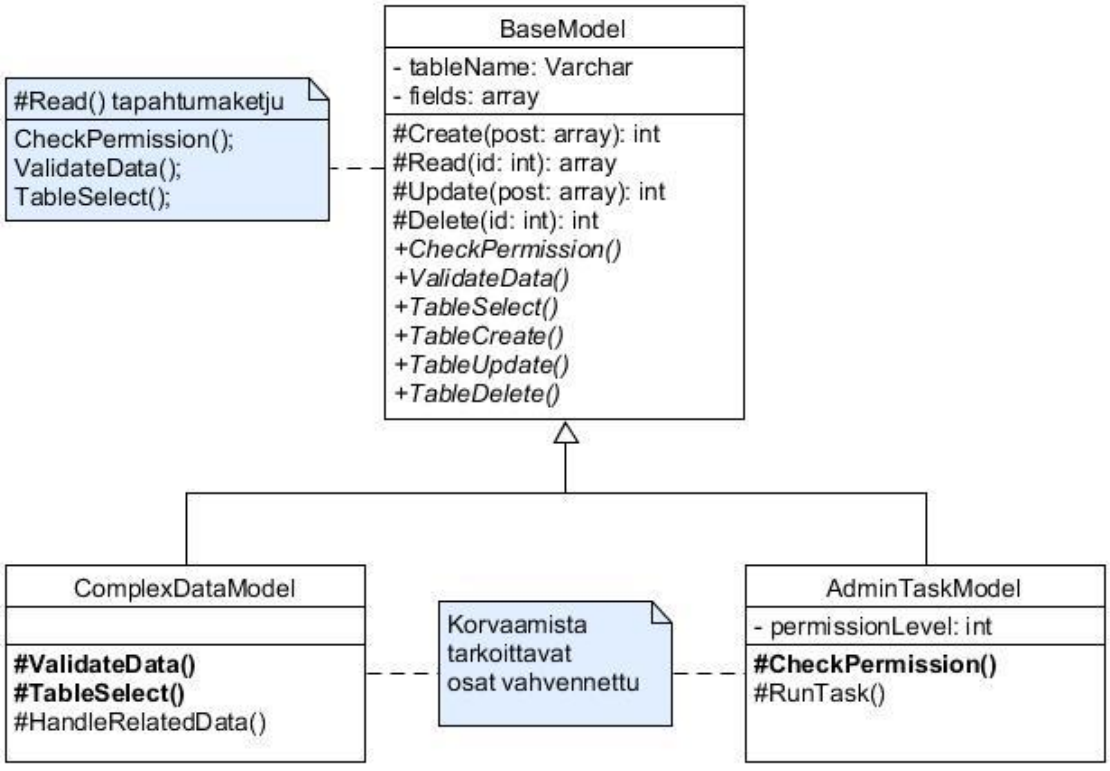
Template metodi keskittyy määrittelemään operaatioon rungon algoritmille (Gamma, ym. 1995). Algoritmin rakenne ja osat määritellään, mutta eri osien toteutusta voidaan muokata perityissä aliluokissa (Gamma, ym. 1995). Tarkoituksena on määritellä runko jollekin yleiselle operaatiolle siten, että eri osien toteutus jätetään aliluokille (Shvets 2015).

Template metodi suunnittelumalli ratkaisee ongelman, jossa kaksi eri komponenttia sisältävät merkittäviä samankaltaisuuksia, mutta eivät ole kokonaisuutena ratkaistavissa tietyn rajapinnan tai toteutuksen avulla (Shvets 2015). Muuttuvien osien ulkoistaminen aliluokille mahdollistaa yhtenäisten osien toteuttamisen vain yhdessä luokassa (Gamma, ym. 1995). Ilman ongelman ratkaisua komponenttien yhteisiin ominaisuuksiin tulevat muutokset tulisi monistaa kaikkien komponenttien toteutukseen (Shvets 2015).

Template metodin avulla voidaan selvittää jokin standardi tai haluttu tapahtumaketju, jota kaikkien samankaltaisten operaatioiden tulisi noudattaa (Shvets 2015). Tapahtumaketjun eri osia voidaan merkitä muuttumattomiksi, tai muunneltaviksi (Shvets 2015). Muunneltavuuden erottelulla luodaan turvaa sille, että tärkeä operaatio tehdään juuri oletetusti, mutta jokin muutoksia mahdollisesti vaativa osuus voidaan silti muuttaa tarvittaessa. Esimerkiksi tietokanta kysely, jossa luodaan uusi rivi tietokanta tauluun. On mahdollista, että ennen varsinaista SQL-kyselyä tulee tarkistaa, onko kutsun kysyjällä oikeus suorittaa operaatiota, sekä ovatko käyttäjän antamat syötteet valideja. Rakenteensa avulla kaikille kyselyille voidaan

määrittää oletusarvoinen tapa tarkastaa käyttäjän oikeudet ja käyttäjän antamat syötteet, mutta erikoisemmissa tilanteissa osia voidaan korvata (Shvets 2015).

Kuvio 2 määrittelee esimerkkirakenteen tietoja käsittelevälle luokalle, joka hyödyntää Template metodi suunnittelumallia. Esimerkissä malliluokalle on tehty pohjaksi tyypilliset tietokanta toimenpiteet (CRUD), joka käsittää yleiset tarvittavat toimenpiteet. Kukin osa sisältää tyypillisen toteutuksen ja tarkistukset. Monimutkaisemmissa tilanteissa voidaan korvata tarvittavilta osin oikeuksien tarkistus, datan validointi ja datan käsittelystä vastaava kutsu.



Kuvio 2. Kuvaus Template metodi -suunnittelumallin periaatteesta.

4.2.1 Miksi template metodi -suunnittelumalli

Yleisesti tietokantaan tehtävät kutsut ovat hyvin samankaltaisia ja eri tietokokonaisuuksien välillä pätee hyvin samankaltaisia lainalaisuuksia. Esimerkiksi käyttäjän syöttämä data pitää tarkastaa SQL-injektioiden varalta, huolimatta siitä, minkälaiseen tauluun tietoa ollaan lisäämässä. Lisäksi käyttäjän oikeus suorittaa haluttu toiminto on usein hyvin samankaltainen läpi ohjelmiston. Tästä syystä on mielekästä määrittää yleisesti käytetty ketju templateksi kullekin operaatiolle, jolloin yksittäisten luokkien toteutus jää vähemmän kunkin ohjelmoijan huolehdittavaksi. Tämän rakenteen ansiosta ohjelmistosta saadaan helpommin ylläpidettävä ja turvallisempi, sillä oikeuksien tarkistusta ja syötteen validointia ei voida vain unohtaa, vaan ne täytyy erikseen poistaa käytöstä, ennen kuin kyseiset toimenpiteet jäävät tekemättä. Joissakin tapauksissa vaaditaan kuitenkin poikkeuksia, joten on tärkeää, että malli mahdollistaa senkin mahdollisimman joustavasti.

MVC-suunnittelumalli jättää tietokantaoperaatioiden jaon erillisiin luokkiin hyvin avonaiseksi (Pop ja Altar 2014). Tätä ongelmaa voidaan helpottaa asettamalla mallien perustaksi yhden selkeän tietokokonaisuuden ja asettamalla mallin pohjaan valmiiksi yleiset tietokanta operaatiot:

- Luominen (**C**reate)
- Lukeminen (**R**ead)
- Päivittäminen (**U**pdate)
- Poistaminen (**D**elete)

Näihin yleisiin operaatioihin viitataan yleensä akronyymillä CRUD. Kullakin näistä operaatioista on omat roolinsa ja ne suoritetaan yleensä hyvin samankaltaisten tarkistusten kera. Monissa järjestelmissä kullekin edellä mainituista operaatioista on omat erilliset käyttöoikeutensa. Lisäksi kussakin tapauksessa tulee tarkistaa käyttäjän antama syöte erilaisten virheellisten tai haitallisten syötteiden varalta. Varsinaisen kyselyn jälkeen on monesti myös mielekästä muotoilla palautettu data muotoon, joka vastaa näkymälle sovittua datan muotoa. Yleisesti kullekin operaatiolle voidaan määritellä esimerkkinä toimivat alioperaatiot:

1. Operaation käyttöoikeuksien tarkastaminen.
2. Käyttäjän antaman syötteen tarkastaminen.
3. SQL-kyselyn suorittaminen.
4. Kyselyn palauttaman datan muotoileminen.

Alioperaatioiden joustava toteutus mahdollistaa koodin uudelleenkäyttöä hyvin laajasti tietokantaoperaatioiden muodostamisessa. Lisäksi alioperaatioista tehdyt mallit kutsuille ja yleisille tietokantaoperaatioille varmistavat, että kaikki tietokantaoperaatiot käyvät läpi tarvitun ketjun ja näin ollen välivaihe ei jää tekemättä, jolloin tietoturva-aukko jää helpommin tekemättä. Mallien pohjaksi voidaan tehdä oletus toteutus eri operaatioille ja yhdistää ne yleisiin operaatioihin, jolloin ideaalissa yksinkertaisessa tilanteessa riittää, kun annetaan tietokanta-taulun nimi ja taulussa olevat tietorivit. Ideaalin tilanteen lisäksi rakenne auttaa monimutkaisten kokonaisuuksien luomista uudelleenkäytettävien ja tarvittaessa vaikka kokonaan ylikirjoitettavien alioperaatioiden ansiosta.

Ilman tarkempaa määrittelyä malliluokkien rajoista, malliluokat voivat kasvaa todella suuriksi, tai muuten epäloogisiksi kokonaisuuksiksi. Template metodi -suunnittelumallin avulla malliluokille voidaan antaa suuntaa siitä, minkälaisen kokonaisuuden kukin luokka toteuttaa. Tästä johtuen mallien sisältämät kokonaisuudet ovat selkeämpiä ja malliluokkien koko pysyy helpommin hallittavana.

4.2.2 Template metodi-suunnittelumalliin kohdistuvaa kritiikkiä

Template metodista löytyy hyvin vähän hyvin asiaa käsitteleviä tieteellisiä artikkeleita. Kritiikkiä on vaikea löytää tieteellisistä lähteistä. Hakua on suoritettu muun muassa hakusanoilla ”template method design pattern” ja ”template method design pattern criti” palveluista Google Scholar ja Springer. Lähtökohtana on yleisemmin kyseisen mallin esittely ja hyödyntäminen. Monipuolisemmin kritiikkiä löytyi google-haun kautta erilaisilta GoF-malleja selventäviltä verkkosivuilta ja niihin liittyvistä keskusteluista. Suunnittelumallia koskevat keskustelut esittävät template metodin heikkoudeksi suuren riippuvuuden, joka eri osasten välille jää. Pohjaluokan kasvaessa yleisesti käytettävien operaatioiden ylläpito voi hankaloitua, kun samaa funktiota on käytetty monessa eri paikassa. Yleiset operaatiot voivat olla

käytössä monessa eri paikassa ja muutokset voivat aiheuttaa ongelmia, jotka tulevat esiin vasta myöhemmin jossain poikkeustilanteessa.

5 Metodi

Konstruktiivisessa osassa tarkastellaan suunnittelumallien tuomia hyötyjä tekemällä kaksi protoa ja tarkastelemalla protojen välisiä eroja erilaisten ylläpidettävyyttä ja tietoturvaa tarkastelevien metriikoiden avulla. Tehtäviin tuotoksiin viitataan prototyyppi -termin sijaan hieman puhekielisemmällä termillä: ”proto”. Tämän termin valinnalla halutaan kuvastaa sitä, että tehdyt tuotokset ovat kokonaisen ja kunnollisen prototyypin sijaan enemmänkin niin sanottu ensimmäinen proto. Ensimmäisen proton tarkoituksena on vain teknisen konseptin toimivuuden tarkastelu, kattavien prototyyppien muodostamisen sijaan.

Ensimmäinen proto tehdään ilman tutustumista kehitystä tukeviin malleihin ja toinen proto tehdään esiteltyjä malleja hyödyntäen. Tehtyjen protojen eroja tarkastelemalla selvitetään mitä hyötyjä ja haittoja suunnittelumallien hyödyntäminen on tuonut kehitettyyn protoon. Hyötyjä ja haittoja tarkastellaan kunkin aiemmin esitellyn ylläpidettävyyden ja tietoturvan osien avulla. Protojen toteutuksesta tarkastellaan siis sitä, miten hyvin teoriataustassa esitellyt kriteerit tietoturvalle ja ylläpidettävyydelle toteutuvat. Tarkastelua varten valitaan teorian perusteella tärkeimpiä tarkasteltavia ominaisuuksia, joihin perehdytään hieman tarkemmin.

5.1 Tutkimuskysymykset

Tämän tutkimuksen tavoitteena on selvittää suunnittelumallien hyödyntämisen vaikutukset www-sovellusten ylläpidettävyyteen ja tietoturvaan. Tarkat tutkimuskysymykset ovat:

1. Millä tavoin suunnittelumallien hyödyntäminen vaikuttaa www-sovellusten ylläpidettävyyteen ja onko vaikutus positiivista vai negatiivista?
2. Millä tavoin suunnittelumallien hyödyntäminen vaikuttaa www-sovellusten tietoturvaan ja onko vaikutus positiivista vai negatiivista?

5.2 Metodin rakentaminen

Teoriataustassa esiteltiin www-sovelluksia ja niiden erityistarpeita erityisesti tietoturvan ja ylläpidettävyyden näkökulmasta. Erityistarpeiden tarkempi määrittely mahdollistaa niiden

tarkastelun kehitetyissä protoissa. Teoriataustassa ylläpidettävyyttä ja tietoturvaa käsiteltiin laajemmin, mutta tarkemmin tutustuttiin vain olennaisimmiksi katsottuihin aspekteihin. Tarkempaan tarkasteluun valittiin tietoturvasta ja käytettävyydestä ne ominaisuudet, jotka ovat tärkeytensä lisäksi tutkimuksen mittakaavassa mitattavissa ja järkeviä.

Metodi on rakennettu tutustumalla samankaltaisesti protoja tarkastelleisiin lähteisiin. Tässä tutkimuksessa on otettu mallia protojen hyödyntämiseen ja metriikoiden määrittelemiseen, sekä niiden hyödyntämiseen Hoffman ja Eugster (2008) tutkimuksesta. Tutkimuksessaan he tarkastelevat saavutettuja hyötyjä refaktoroimalla neljä ohjelmaa ja tarkastelemalla miten eri mittarien arvot ovat muuttuneet alkuperäiseen nähden. Tässä tutkielmassa hyödynnetään vastaavasti mittareita eri versioiden erojen tarkastelemiseen, mutta sen sijaan että tarkastellaan eroja alkuperäisen ja refaktoroidun ohjelman välillä, tarkastellaan eroja kahden erikseen kehitetyn proton välillä.

Hoffman ja Eugster (2008) tavoin määrittelemme tekniikat protojen kehittämiselle, määrittelemme kohteena olevan proton, sekä käytetyt metriikat. Määrittelyn jälkeen tarkastelemme metriikoiden antamat tulokset ja lopuksi tarkastelemme johtopäätöksiä taustan ja tulosten pohjalta.

Tässä tutkimuksessa protojen kehitys tekniikoilla tarkoitetaan aiemmin esiteltyjä suunnittelumalleja. Tekniikoita hyödyntävän proton lisäksi tehdään myös toinen tekniikoita hyödyntämätön proto. Tekniikoita hyödyntämätön proto tarjoaa vertailukohdan, johon vertaamalla saadaan kuvaa tekniikoiden tuomasta hyödystä. Molempien protojen taustalla olevat vaatimukset on määritelty kehittäjän mielenkiinnon pohjalta mahdollisimman kompleksiseksi, mutta silti skaalaltaan pieneksi. Määrittelyn tarkoituksena on pitää vaaditut resurssit kurissa, mutta silti antaa tilaa suunnittelumallien tuomille hyödyille. Tutkimuksessa käytetyt metriikat määritellään aiemmin esiteltyjen ylläpidettävyyden ja tietoturvan mittaamiseen määriteltyjen tietojen pohjalta tutkimuksen skaalan huomioon ottaen mahdollisimman laajaksi.

5.3 Käytetyt ylläpidettävyyden mittarit

Aiemmin ylläpidettävyyden ja tietoturvan määrittelyn yhteydessä määriteltiin tapoja ja erilaisia metriikoita ominaisuuksien mittaamiseksi. Esitellyistä metriikoista valitaan tutkimuksen rajoitteiden puitteissa monipuolinen kokonaisuus, jonka tarkoituksena on mitata mahdollisimman laajasti haluttuja ominaisuuksia. Metriikoiden valinnassa on pyritty kattamaan mahdollisimman laajasti www-sovellusten ylläpidettävyydelle tärkeäksi määritettyjä ominaisuuksia, sekä pyritty tuomaan esille mittaustavasta johtuvat heitot ominaisuuksia mittaavien metriikoiden antamissa tuloksissa. Ylläpidettävyyden metriikoiden mittauksessa on hyödynnetty PhpMetrics-analyysityökalua (Lépine 2015). Taulukossa 3 kuvataan kaikki ylläpidettävyyden metriikat lyhyen selityksen kera.

Metriikka	**Mitä mittaa**
koodirivien määrä (LOC)	Mittaa ohjelmiston laajuutta ja analysoitavuutta.
Loogisten koodirivien määrä (LLOC)	Mittaa ohjelmiston laajuutta ja analysoitavuutta ilman kommenttien ja ohjelmointityylin vaikutusta.
Keskimääräinen koko (Halstead volume)	Operaatioiden määrään pohjautuva laskennallinen mitta ohjelmiston laajuudelle.
Ylläpidettävyys indeksi (MI)	Ohjelmiston ylläpidettävyyden kokonaisuutta.
Keskimääräinen vaikeus (Halstead difficulty)	Kuinka vaikea ohjelmistoa on ymmärtää tai kirjoittaa.
Luokkien syklomaattinen monimutkaisuus	Mittaa yksiköiden monimutkaisuutta.

Taulukko 3. Ylläpidettävyyden analyysissa käytetyt metriikat

Ohjelmiston koko on yksi osa ohjelmiston ylläpidettävyyttä, ja aiemmin sen todettiin vaikuttavan ainakin ohjelmiston analysoitavuuteen (Heitlager;Kuipers ja Visser 2007). Riaz, Mendes ja Tempero (2009) suorittaman tutkimuksen mukaan kokoon liittyvät metriikat an-

tavat kuvaa jopa tasaisesti kaikista ylläpidettävyyden aliominaisuuksista. Laajojen ohjelmistokokonaisuuksien ylläpito on yleisesti haastavampaa, kuin pienempien kokonaisuuksien (Heitlager;Kuipers ja Visser 2007). Ohjelmiston kokoa voidaan mitata monilla erilaisilla tavoilla. Selkeä mittari ohjelmiston koolle on koodirivien määrä. Ylläpidettävyyden kannalta koodirivien määrän mittaamisen ongelmana on, että siinä ei oteta huomioon mistä ohjelmarivit koostuvat. Esimerkiksi ohjelmisto, joka on todella laajasti kommentoitu saattaa olla tällä mittarilla paljon laajempi, vaikka todellisuudessa hyvin kommentoitua ohjelmistoa on todennäköisesti helpompi ylläpitää. Ohjelmiston laajuuden tarkempaa tarkastelua varten on valittu myös loogisten koodirivien määrä. Loogisten koodirivien määrä kuvaa tarkemmin sitä, kuinka paljon erilaisia komentoja koodirivissä on toteutettu. Yksinkertainen koodiesimerkki on saatavilla liitteessä B. Koodiesimerkissä koodirivejä on 48, kun taas loogisia koodirivejä esimerkissä on 21.

Halsteadin koko on koodiriveihin perustuvasta laskusta hyvin paljon poikkeava ohjelmiston koon mittari. Halsteadin koko perustuu operaatioiden operaattoreiden määrään (Verifysoft Technology GmbH 2017). Halsteadin koon arvoksi suositellaan funktiolle vähintään 20 ja maksimissaan 1000, tiedostolle vähintään 100 ja maksimissaan 8000 (Verifysoft Technology GmbH 2017). Halsteadin koko itsessään on vaikea tulkita, mutta sitä hyödynnetään muissa lasketuissa metriikoissa, kuten ylläpidettävyys indeksi (Heitlager;Kuipers ja Visser 2007).

Ohjelmiston koon lisäksi ylläpidettävyyden mittaamista varten on rakennettu erilaisia laskennallisia mittoja (Riaz;Mendes ja Tempero 2009). Laskennalliset mitat perustuvat yleensä ohjelmakoodin ominaisuuksiin perustuviin laskentakaavoihin, joiden on todettu antavan kuvaa ohjelman ominaisuuksista. Tässä tutkimuksessa ylläpidettävyyden mittaamisessa hyödynnetään ylläpidettävyys indeksiä, sillä sen on todettu antavan kuvaa ohjelman ylläpidettävyydestä kokonaisuutena (Heitlager;Kuipers ja Visser 2007). Ylläpidettävyys indeksi on muodostettu keräämällä dataa suuresta määrästä eri ohjelmia ja tarkastelemalla niiden ylläpidettävyyttä (Heitlager;Kuipers ja Visser 2007). Ylläpidettävyys indeksistä on olemassa monia eri variaatioita. Eri variaatioissa skaala vaihtelee hieman ja lisäksi osa variaatioista katsoo kommentit positiivisena vaikuttajana ja osa ei huomioi ohjelmakoodista löytyviä kommentteja ollenkaan. Tässä tutkimuksessa käytetty ylläpidettävyyden indeksi huomioi

kommenttien osuuden tehdystä koodista ja se lasketaan kaavalla ”*MI = 171 - 5.2 * ln(Halstead Volume) - 0.23 * (Cyclomatic Complexity) - 16.2 * ln(Lines of Code) + 50 * sin(sqrt(2.4 * ((Comment lines of code) / (Lines of code))))”* (Lépine 2015). Ylläpidettävyys indeksi on välillä 0-118, korkeampi arvo tarkoittaa parempaa ylläpidettävyyttä (Lépine 2015). Ylläpidettävyys indeksin suurena ongelmana on, että se ei suoraan kerro, missä ongelma on, vaan antaa vain yleiskuvaa tilanteesta (Heitlager;Kuipers ja Visser 2007).

Laskennallisia ohjelmakoodin monimutkaisuutta ja vaikealukuisuutta kuvaavia metriikoita ovat Halsteadin keskimääräinen vaikeus, sekä syklomaattinen monimutkaisuus. Nämä metriikat on otettu mukaan, jotta saadaan ohjelmiston koon lisäksi tietoa myös siitä, kuinka muokattavaa ja testattavaa ohjelmakoodi on. Halsteadin keskimääräinen vaikeus perustuu uniikkien operandien määrää ohjelmakoodissa ja kertoo myös uniikkien operandien suhteesta operandien kokonaismäärään (Verifysoft Technology GmbH 2017). Halsteadin keskimääräisen vaikeuden on todettu kuvaavan ohjelmakoodin vaikeustasoa ja virheherkkyyttä (Verifysoft Technology GmbH 2017). Syklomaattinen monimutkaisuus puolestaan pohjautuu ohjelmiston kulkuun (Rychkova, ym. 2017). Syklomaattinen monimutkaisuus kertoo, kuinka monta lineaarisesti riippumatonta reittiä ohjelmiston läpi voidaan kulkea (Rychkova, ym. 2017). Syklomaattisen monimutkaisuuden ongelmana on, että se helposti tasoittaa pahimmat tapaukset, jolloin ongelma voi jäädä huomaamatta (Heitlager;Kuipers ja Visser 2007).

Tällä mittaristolla saadaan hyvä leikkauskuva aiemmin määriteltyihin ylläpidettävyyttä rakentaviin ominaisuuksiin. Ohjelmakoodin laajuus, yksikköjen koko ja yksikköjen monimutkaisuus ovat suoraan edustettuina. Koodin toistoa ei suoraan mitata, mutta suuri määrä koodin toistoa vaikuttavat Halsteadin keskimääräiseen vaikeuteen ja syklomaattiseen monimutkaisuuteen.

5.4 Käytetyt tietoturvan mittarit

Aiemmin tässä tutkimuksessa esiteltiin tapoja mitata sovelluksen tietoturvaa. Tietoturvan mittaus jaettiin kahteen pääkategoriaan: staattiseen ja dynaamiseen analyysiin

(Fonseca;Vieira ja Madeira 2014). Tietoturvan mittaus katsottiin kattavimmaksi, kun hyödynnetään molemman kategorian tarkastelua (Fonseca;Vieira ja Madeira 2014). Tästä syystä tutkimusta varten valittiin kaksi erillistä metriikkaa. Metriikat määriteltiin sen pohjalta, että saadaan tietoa kehitetyistä protoista sekä staattisen analyysin, että dynaamisen analyysin keinoin.

Tekniikka	**Mitä mittaa**
phpcs-security-audit	Staattinen analyysi protojen toteutukselle. Etsii potentiaalisia tietoturvaongelmia.
Manuaalinen määritelty testaus.	Dynaaminen analyysi protojen toteutukselle. Testataan, pystytäänkö protoja saamaan virhetilaan yleisillä haitallisilla kutsuilla.

Taulukko 4. Tietoturvan analyysissa käytetyt tekniikat

Staattinen analyysi suoritetaan hyödyntämällä phpcs-security-audit -työkalua (Marcil 2015). Phpcs-security-audit hyödyntää PHP-ohjelmakoodia haistelevaa työkalua. Työkalu valittiin käyttöön, koska se ymmärtää PHP-ohjelmakoodia, sekä osan protoissa hyödynnetyn Laravel-ohjelmistokehyksen ominaisuuksista. PHP-ohjelmointikieltä tarkastelevia työkaluja oli hyvin rajoitetusti saatavilla ilmaiseksi.

Dynaaminen analyysi suoritetaan määrittelemällä testikerta, joka suoritetaan molemmille protoille. Testikerrassa käydään läpi jokainen toteutettu näkymä, jossa otetaan vastaan käyttäjän syötteitä. Lisäksi tarkastellaan, voidaanko protoja huijata syöttämällä virheellisiä syötteitä osoiteriville. Testikerran syötteiden määrittelyssä on käytetty hyödyksi (W3Schools 2017) SQL injektion esimerkkejä sekä (OWASP 2016) SQL injektion testausohjetta.

5.5 Metodin heikkoudet

Metodin selkeäksi heikkoudeksi muodostuu se, että protot ovat täysin toisistaan erillisiä. Protot ohjelmoidaan täysin eri aikaan ja erilaisessa mielentilassa, joten kaikkia muuttujia on

vaikea huomioida. Lisäksi molemmat protot ovat saman henkilön kehittämiä, joten suunnittelumallien antaman ohjeistuksen lisäksi kehittäjän taidot Laravel-sovelluskehyksen käytössä kehittyvät ja myös tieto siitä, miten proto kokonaisuutena rakentuu. Ongelmaksi muodostuu se, että on vaikeaa vetää rajaa sille, mikä on kehittäjän oppimaa ja mikä on suunnittelumallien tuomaa etua. Varsinaisen opin lisäksi myös kehittäjän vireys voi vaikuttaa kehitetyn ohjelmiston laatuun. Jos kehittäjä toteuttaa toisen proton virkeänä ja siten, että aikaa on reilusti, sillä voi olla myös vaikutusta lopputulokseen.

Toinen suuri ongelma metodin kannalta syntyy mittareiden määrityksestä. Tarkasteltava ilmiö ei ole yksinkertainen ja selkeästi mitattava luonnonvakio, vaan kyseessä on ilmiö, jota voidaan tarkastella eritavoin, mikäli asia vain määritellään eri tavalla. Ominaisuuksille määrityt mittarit eivät ole siis yksiselitteisiä ja aina totuuden paljastavia, vaan mittarit voivat antaa vääristynyttäkin tietoa joissain tapauksissa.

Näitä heikkouksia on pyritty lieventämään tekemällä protoja ainoastaan ohjelmointiin sopivassa mielentilassa siten, että virtaa on tarpeeksi kelvollisten ratkaisujen tekemiseen. Laravel-sovelluskehyksen oppimisen vaikutuksia on pyritty välttämään sillä, että protot kehittänyt henkilö on tutustunut käyttämiinsä ominaisuuksiin jo etukäteen täysin erillisessä projektissa, ennen töiden aloittamista. Mitattavan ilmiön luonnetta on pyritty huomioimaan käyttämällä useita yleisesti käytettyjä mittareita ja tulkitsemalla niitä yhdessä.

5.6 Kuvaus kehitettävistä protoista

Esimerkkisovellus on www-sovellus, joka toteuttaa DKP nimisen pistejärjestelmän, jota hyödynnetään joissain pelaajaryhmissä. DKP on resurssien jakamisen järjestelmä, jota pelaajat käyttävät virtuaalisissa maailmoissa jakaakseen hyödykkeitä perustuen pelin lisäksi myös pelaajan suorittamaan vaivannäköön (Castronova ja Fairfield 2007). Pistejärjestelmän ideana on tallentaa tietoa kunkin pelaajan tekemistä saavutuksista, sekä pelaajan saamista hyödyistä. Näiden tietojen perusteella voidaan muodostaa lista, joka kertoo kunkin henkilön pistemäärän, joka ikään kuin priorisoi kuka on tehnyt eniten ja saanut suhteessa vähiten vastineeksi.

DKP järjestelmän taustalla on idea, että pelaajat kuluttavat ansaitsemiaan pisteitä (Gran ja Reinemo 2008). Jos käyttäjät eivät kuluta omia pisteitään vaan keräävät pisteitä johonkin tärkeämpään tavaraan, niin tavarat menevät hukkaan ja kaikkien käyttäjien pistemäärät kasautuvat, eikä uudet pelaajat voi helposti liittyä mukaan toimintaan (Gran ja Reinemo 2008). Lisäksi pistejärjestelmällä annettavat bonukset voivat motivoida pelaajia valmistautumaan paremmin (Gran ja Reinemo 2008). Näistä seikoista johtuen DKP järjestelmästä on vedetty monia variaatioita, joiden ideana on ratkaista järjestelmässä olevia ongelmia (Gran ja Reinemo 2008). Perus variaatioiden tueksi normaalien pistejärjestelmän toimintojen lisäksi käyttäjien pisteille pitää pystyä suorittamaan joustavia, myös prosentteihin perustuvia muutoksia.

Jotta järjestelmää olisi helppo ja nopea käyttää, niin tietoihin tallennetaan myös metatietoa, luokkia ja rooleja, joiden mukaan haluttuja tietoja voidaan valita ja suodattaa. Jotta toteutus ei palvelisi vain yhden pelin pelaajakuntaa, niin luokkien ja roolien tietoja ja määrää pitää voida muuttaa. Peleissä käytetyissä termeissä on muutenkin eroja, joten proto tulee lokalisoida.

Käyttäjien kirjautumistiedot ja käyttäjien pistetiedot tallennetaan tietokantaan, josta ne haetaan tarvittaessa erilaisiin näkymiin. Sovellukseen kirjautuneet käyttäjät voivat suorittaa pistedatalle erilaisia muokkauksia ja ennalta määriteltyjä laskutoimituksia. Kehitettävät protot tehdään PHP-ohjelmointikielellä hyödyntäen Laravel-ohjelmistokehystä. Protojen dataa tallentavana tietokantana käytetään MySql-tietokantaa.

5.6.1 Kehitysympäristön valinta

Kehitettävien protojen tarkoituksena on kuvata tyypillisiä www-sovelluksia, joten protot toteutetaan mahdollisimman tyypillisessä www-sovellusten kehitysympäristössä. Protoissa käytetään ohjelmointikielenä PHP-ohjelmointikieltä, sillä www-sovellusten kehityksessä yleisin käytetty ohjelmointikieli on PHP (W3Techs 2017). Www-sovelluksissa käytettyjen ohjelmointikielten osuuksia on hyvin vaikea määritellä tarkasti, mutta löydetyn datan perus-

teella todella selkeä enemmistö www-sovelluksista käyttää palvelinpuolen ohjelmointikielenä PHP:tä. W3Techs:n (2017) mittausten mukaan jopa yli 82% tutkituista sivustoista käyttää ohjelmointikielenään PHP:tä.

Laravel on valittu hyödynnettäväksi ohjelmistokehykseksi, koska se on yksi käytetyimmistä www-sovellusten ohjelmistokehyksistä (Github 2017). Lisäksi protot toteuttavalla kehittäjällä on jo hieman kokemusta Laravel-ohjelmistokehyksestä, joten ensimmäistä kehitettävää protoa ei tarvitse kehittää täysin uudessa ympäristössä. Lisäksi Laravel on hyvin joustava ja tarjoaa pohjan MVC-suunnittelumallin hyödyntämiselle.

Protojen sisältämää dataa tallennetaan hyödyntäen MySql-tietokantamoottoria. MySql-tietokantamoottori on valittu käyttöön, koska se on yleinen avoimenlähdekoodin www-sovelluksissa käytetty tietokantamoottori. Muita yleisimpiä tietokantamoottoreita ovat Oracle, PostgreSQL ja Microsoft SQL Server. Mainituista tietokantamoottoreista ainoastaan MySQL- ja PostgreSQL- tietokantamoottoreista on saatavilla ilmaiseksi täysi ja rajoittamaton versio.

5.6.2 Kehitettävien protojen vaatimukset

Jotta kehitettävistä protoista saadaan mahdollisimman samankaltaisia, sekä niiden välisiä eroja voidaan tarkastella, niin molemmille protoille määritellään yhteiset vaatimukset. Lista kehitettävien protojen vaatimuksista on lueteltuna liitteessä A.

5.7 Suunnittelumalleja hyödyntämätön proto (SoulDKP)

SoulDKP on proto, joka on toteutettu vastaamaan yleisiä www-sovelluksia, ilman että sen toteutuksessa on tarkoituksella hyödynnetty mitään suunnittelumalleja. Laravel-ohjelmistokehys ohjaa toteuttamaan MVC mallin tyyliin. SoulDKP proto ei kuitenkaan toteuta MVC mallia kokonaan. SoulDKP hyödyntää vain näkymiä (View) ja ohjaimia (Controller), mutta malleja (Model) ei ole toteutettu. Tästä johtuen SoulDKP proton ohjelmakoodissa nimen perusteella ohjainluokilta vaikuttavat luokat sisältävät muutakin ohjelmakoodia, kuin pel-

kästään ohjaimeen kuuluvaa ohjelmakoodia. SoulDKP hyödyntää vain vähän Laravelin tietokantaominaisuuksia ja toteuttaa toiminnan MySQL-tietokannan proseduureilla, joita vain kutsutaan ohjelmakoodista yksinkertaisilla kutsuilla. Tietokantaproseduurit suorittavat ohjelman tarvitsemat toiminnot tietokantaan ja palauttavat tiedot.

SoulDKP proton kehitys aloitettiin ennen kuin oli edes varmaa, että kyseinen proto tulee lopulta toimimaan tutkimuksessa käytettävänä protona. Kun SoulDKP päätettiin ottaa tutkimukseen protoksi, niin sen kehitystä jatkettiin hieman ja muutettiin vastaamaan tutkimuksessa protolle asetettuja vaatimuksia. SoulDKP toteutettiin iltaisin ja viikonloppuisin varsinaisen työpäivän jo päätyttyä.

5.8 Suunnittelumalleja hyödyntävä proto (JelaDKP)

JelaDKP on proto, joka on toteutettu hyödyntäen MVC- ja template metodi -suunnittelumalleja. JelaDKP on toisen proton tapaan toteutettu hyödyntäen Laravel-ohjelmistokehystä. JelaDKP hyödyntää Laravel-ohjelmistokehyksen näkymä ja ohjainluokille määritettyjä ominaisuuksia, mutta JelaDKP:n malliluokat on toteutettu template metodi suunnittelumallia ja Laravel-ohjelmistokehyksen tietokanta operaatioita hyödyntämällä. Laravelin toteuttamia malliluokkia ei siis hyödynnetä suoraan. JelaDKP toteuttaa kaiken logiikan PHP-koodissa, eikä koodi ole siis jakautunut varsinaisen ohjelman ja tietokannan välille.

JelaDKP:n kehitys aloitettiin vasta kun kehittäjä oli tarkoin tutustunut MVC- ja template metodi -suunnittelumalleihin. Proton toteutuksen alkuun mennessä ensimmäiseksi tehty SoulDKP proto oli ollut jo pitkään valmiina. JelaDKP toteutettiin yhden viikon aikana. JelaDKP:n toteutuksessa on hyödynnetty SoulDKP:n näkymiä ja ohjainluokille on otettu pohjaa SoulDKP:n ohjain rakenteesta. Ensimmäisen kokonaisuuden toteutukseen kului suuri osa ajasta, mutta myöhemmin uudelleenkäytettävät osaset ja määritellyt rutiinit helpottivat ja nopeuttivat kehitystä. Toteutus tehtiin varsinaisella työajalla, mutta työpäivät olivat normaalia pidempiä (noin 10-12 tuntia). JelaDKP on kehitetty pitkien työpäivien aikana, suurelta osin hyvässä vireessä.

6 Tulokset

Suunnittelumallien hyötyjä mitattiin erikseen ylläpidettävyyteen ja tietoturvaan perehtyvillä metriikoilla. Ylläpidettävyyden osalta protojen välille saatiin eroja eri ylläpidettävyyden osa-alueiden välille. Tietoturvan osalta molemmat protot pärjäsivät hyvin. Toinen tietoturvan mittaamisessa käytetyistä metriikoista antoi poikkeavia tuloksia protojen välille.

6.1 Suunnittelumallien vaikutus ylläpidettävyyteen

Ylläpidettävyyttä kuvaavat metriikat antoivat toisistaan poikkeavia tuloksia protojen välille. Metriikoiden välillä oli selkeitä eroja kaikkien muiden paitsi Halsteadin keskimääräisen vaikeuden kohdalla. Tulokset on merkitty kiteytettynä taulukkoon 5. Sarakkeessa ”% muutos” (▼) tarkoittaa negatiivista muutosta ja (▲) tarkoittaa positiivista muutosta.

Metriikka	**SoulDKP tulos**	**JelaDKP tulos**	**% muutos**
Koodirivien määrä (PHP + SQL)	(721 + 732) = 1453	2279	+ 56,8% (▼)
Loogisten koodirivien määrä	(400 + 372) = 772	1088	+ 40,9% (▼)
Halstead koko keskiarvo	1103	357	- 67,6% (▲)
Ylläpidettävyys indeksi (MIN)	59,03	70,18	+ 18,9% (▲)
Ylläpidettävyys indeksi (AVG)	85,34	93,35	+ 9,4% (▲)
Halstead keskimääräinen vaikeus	4,07	3,92	- 3,7% (▲)
Luokkien syklomaattisen monimutkaisuuden keskiarvo	5,17	1,85	- 64,2% (▲)

Taulukko 5. Ylläpidettävyyden metriikoille saadut arvot

SoulDKP proton koodit ovat jaettuna osa PHP koodiin ja osa SQL-tietokantaan funktioina, näkyminä ja proseduureina. Koodirivien laskussa SoulDKP:n osalta yhdistettiin molemmat koodit, jotta luvut olisivat paremmin verrattavissa. JelaDKP:n osalta kaikki koodi on PHP koodissa, joten vastaavaa yhdistämistä ei tarvittu. SoulDKP protossa koodiriveihin perustuvat arvot olivat pienempiä, kuin JelaDKP protossa. JelaDKP:n koodimäärä on noin puolitoista kertainen verrattuna SoulDKP protoon. Ero on kuitenkin pienempi loogisten rivien mittauksessa, kuin kaikkien koodirivien mittauksessa.

Halsteadin koko keskiarvo mittarilla puolestaan JelaDKP on selkeästi pienempi kuin SoulDKP. JelaDKP proton tiedostot ovat siis selkeästi SoulDKP:n tiedostoja pienempiä. JelaDKP:n kokonaisuudet ovat noin kaksi kolmasosaa pienempiä kuin SoulDKP:n kokonaisuudet.

Ylläpidettävyys indeksin laskussa SoulDKP:n tietokannan monimutkaisuus ei ole mukana, sillä kaikki tietokantaan liittyvät operaatiot ovat erillisenä kokonaisuutena MySQL-tietokannassa. JelaDKP:n osalta tietokannan rakenne on osana ohjelmakoodia, joten myös tietokannan rakenteiden käsittelyt monimutkaistavat ohjelmakoodia. Tästäkin huolimatta JelaDKP:n ylläpidettävyysindeksien keskiarvo oli korkeampi, kuin SoulDKP:n ylläpidettävyysindeksien keskiarvo. Lisäksi SoulDKP:n heikoin ylläpidettävyysindeksi oli paljon alhaisempi, kuin JelaDKP:n heikoin ylläpidettävyysindeksi. SoulDKP:n heikoin ylläpidettävyysindeksi (59,03) lasketaan ylläpidettävyysindeksin metriikan mukaan matalan ylläpidettävyyden luokitukseen (MI < 65) (Lépine 2015). JelaDKP:n alhaisin ylläpidettävyysindeksi (70,18) puolestaan pysyy ylläpidettävyysindeksin metriikan mukaan keskivälin ylläpidettäväksi (65 < MI < 85) (Lépine 2015).

Halstead keskimääräinen vaikeus oli protojen välillä hyvin lähellä toisiaan. SoulDKP:n osalta metriikassa ei näy tietokantaoperaatioiden luomat haasteet, sillä kuten ylläpidettävyysindeksin kohdalla, ne ovat MySQL-tietokannassa. Tästä eroavaisuudesta huolimatta JelaDKP:n keskimääräinen vaikeus oli hieman pienempi, kuin SoulDKP:n keskimääräinen vaikeus.

Luokkien syklomaattisen monimutkaisuuden keskiarvoissa oli protojen välillä suuri ero. JelaDKP:n syklomaattisen monimutkaisuuden keskiarvo oli lähes kaksi kolmasosaa pienempi,

kuin SoulDKP:n syklomaattisen monimutkaisuuden keskiarvo. JelaDKP:n ohjelman kulku sisältää siis vähemmän lineaarista riippuvuutta.

6.2 Suunnittelumallien vaikutus tietoturvaan

Tietoturvan metriikat antoivat hyvin odotettuja tuloksia. Yksikään testitapaus ei saanut kumpaankaan järjestelmään suurta tuhoa aikaan. Tulokset on merkitty kiteytettynä taulukkoon 6.

Testi	**SoulDKP tulos**	**JelaDKP tulos**
phpcs-security-audit	Varoituksia: 2 Virheitä: 0	Varoituksia: 38 Virheitä: 0
Manuaalinen määritelty testaus.	Testitapauksia: 18 Onnistui: 18	Testitapauksia: 18 Onnistui: 18

Taulukko 6. Tietoturvan metriikoille saadut arvot

Phpcs-security-audit testaus ei löytänyt selkeää virhettä kummastakaan kehitetystä protosta. SoulDKP protossa kaikki oli suoritettu yhdessä funktiossa, joten parsija näki, ettei käyttäjän syötteitä viety suoraan SQL kyselyihin. JelaDKP:n kaikki malliluokat antoivat jonkin verran varoituksia, sillä varsinaiset kutsun suorittavat kyselyt olivat omissa funktioissaan, jotka antavat parametrin suoraan Laravel-ohjelmistokehyksen tietokanta luokkien hoidettavaksi. Phpcs-security-audit antoi varoituksia, sillä työkalu ei ole tietoinen siitä, että Laravel-tietokantaoperaatiot tarkastavat annetun syötteen käytetyillä funktioilla. Liitteessä D on esillä phpcs-security-audit -työkalun antamat tulosteet molemmille protoille.

Manuaalinen määritelty testaus suoritettiin molemmille protoille. Kaikki testitapaukset toimivat odotetulla tavalla molemmissa protoissa. Kummastakaan protosta ei siis löydetty heikkouksia määritellyillä testitapauksilla. Tarkempi kuvaus kunkin testitapauksen suorituksesta löytyy liitteestä E.

7 Johtopäätökset

Aluksi esiteltiin taustaa www-sovellusten kohtaamista haasteista. Haasteista siirryttiin www-sovelluksille tärkeisiin ominaisuuksiin: ylläpidettävyyteen ja tietoturvaan. Ylläpidettävyys ja tietoturva liitettiin www-sovelluskehitykseen ja etsittiin linkkejä näiden ominaisuuksien väliltä. Seuraavaksi näiden ominaisuuksien kehittämistä varten tutustuttiin suunnittelumallien hyödyntämiseen, sekä esiteltiin kaksi suunnittelumallia, joiden pitäisi auttaa kyseisten ominaisuuksien kehittämisessä. Nämä suunnittelumallit ovat www-sovelluksille jo tuttu MVC-suunnittelumalli, sekä vähemmän vastaan tullut template metodi -suunnittelumalli. Esiteltyjen suunnittelumallien tuomia potentiaalisia hyötyjä tarkasteltiin tekemällä kaksi protoa, yksi ilman suunnittelumalleja ja toinen suunnittelumalleja hyödyntäen. Protojen eroja tarkasteltiin erilaisilla ylläpidettävyyden ja tietoturvan metriikoilla.

Suunnittelumallien hyödyntäminen antoi protojen välille ristiriitaista tietoa ohjelmiston kokoon (volume) liittyen. Suunnittelumallin hyödyntäminen lisäsi proton ohjelmakoodin määrää riveinä mitattuna, mutta yksittäiset ohjelman osaset pienenivät. Tästä johtuen suoraan ohjelmakoodin riveihin perustuvat koon metriikat osoittivat ylläpidettävyyden heikkenemistä, kun taas yksikköjen koot huomioon ottava Halsteadin koko metriikka osoitti potentiaalista ylläpidettävyyden paranemista.

Suunnittelumallien aiheuttamaa koodimäärän laajenemista selittää se, että suunnittelumallit auttoivat jäsentämään koodin järkeviin palasiin, sillä koodin jäsentäminen palasiin aiheuttaa joidenkin kielessä pakollisten osioiden moninkertaistumista (esimerkiksi luokan määrittely, konstruktori yms.). Lisäksi pienessä protossa uudelleenkäytettävien osien hyödyt jäävät verrattain pieniksi. Protojen kehittäjän havaintojen mukaan pienenkin proton tapauksessa koodin järkevä jakaminen kuitenkin helpotti ominaisuuksien kehittämistä. Suunnittelumallia hyödynnettäessä ensimmäisen ominaisuuden tekoon kului kauemmin, mutta uusia ominaisuuksia lisätessä vaadittu työmäärä pieneni jatkuvasti. Tästä johtuen suunnittelumalleja hyödyntävä proto vaikutti protojen kehittäjälle jo pienessä protossa skaalautuvammalta.

Suunnittelumallien hyödyntäminen jakoi koodin siten, että tärkeät ratkaisut tehtiin selkeissä paikoissa ja tätä kautta www-sovelluksesta tuli helpommin uudelleenkäytettävä, analysoi-

tava ja muokattava, sekä koodi oli vähemmän virheherkkää. Suunnittelumallien avulla tietoturvalle olennaiset operaatiot oli mahdollista keskittää helposti, eikä kriittisiä valintoja tarvitse muistaa huomioida kaikessa ohjelmakehityksessä. Tästä johtuen suunnittelumallien avustaman koodin järkevän jäsentämisen voisi nähdä myös tietoturvaan positiivisesti vaikuttavana tekijänä. Tätä havaintoa tukee Zhangin ja Budgenin (2012) tutkimuksessa esitetty päättely, jonka mukaan suunnittelumallit parantavat ohjelmoijan tuotteliaisuutta, ohjelmiston laatua ja parantaa aloittelijan suunnittelu taitoja.

Syklomaattisen monimutkaisuuden keskiarvo oli selkeästi parempi suunnittelumallien avulla kehitetyssä protossa. Tämä viittaisi siihen, että suunnittelumallien avulla kehitetyn proton kulkua on helpompi seurata (Heitlager;Kuipers ja Visser 2007). Halsteadin keskimääräinen vaikeus oli protojen välillä kuitenkin hyvin lähellä toisiaan. Suunnittelumallien avulla tehty koodi ei siis käytetyn metriikan mukaan ole ainakaan paljon helpompaa, kuin ilman suunnittelumalleja tehty koodi (Heitlager;Kuipers ja Visser 2007). Tähän tulokseen voi vaikuttaa se, että ilman suunnittelumalleja kehitetyssä protossa monimutkaiset tietokantaoperaatiot ovat testidatan ulkopuolella, kun taas suunnittelumallien avulla kehitetyssä protossa monimutkaiset tietokantaoperaatiot ovat integroituna koodissa.

Tietokantakyselyissä ei ollut kummassakaan protossa havaittavissa haavoittuvuuksia. Tähän tulokseen vaikuttaa ratkaisevasti Laravel-ohjelmistokehyksen SQL-yhteyksiä varten toteuttamat apuriluokat, jotka ohjaavat kehittäjää toteuttamaan tietoturvallisia kyselyitä (Laravel 2017). Jos Laravelin toteuttamille apuriluokille erottelee kyselyn ja varsinaiset muuttujat, niin kaikki syötteet tarkastetaan automaattisesti (Laravel 2017). Jos kaikki tietokantayhteydet olisi toteutettu ilman Laravel-ohjelmistokehyksen apua, niin haavoittuvuuksia olisi voinut esiintyä helpommin. Vaikka protojen välillä ei ollut metriikoilla mitattavaa eroa SQL-injektiolle haavoittuvuudessa, niin koodin rakenteen vuoksi suunnittelumalleja hyödyntämätön proto vaatii kehittäjältä enemmän haavoittuvuuksien pitämiseksi poissa. Suunnittelumalleja hyödyntämättömässä protossa jokainen kutsu toteutetaan logiikan seassa ja kehittäjä voisi helposti epähuomiossa jättää nimetyn sitomisen (named binding) pois kutsusta. Tätä ongelmaa ei ole suunnittelumalleja hyödyntävässä protossa, sillä siinä kutsut suoritetaan

yleisten funktioiden läpi antamalla parametrit muodossa, joka ei ota kantaa varsinaiseen kutsuun. Tällöin suunnittelumallien avulla rakennetussa protossa rakennetta noudattamalla ei voida aiheuttaa SQL-injektio haavoittuvuuksia.

Ylläpidettävyysindeksin keskiarvo oli molemmilla protoilla hyvällä tasolla. Molemmat prototot olivat metriikan mukaan kokonaisuutena hyvin ylläpidettäviä. Suunnittelumalleja hyödyntämättömässä protossa oli kuitenkin yksi tiedosto, joka hallitsi muuta koodia ja menikin skaalan mukaan heikosti ylläpidettäväksi tiedostoksi. Suunnittelumalleja hyödyntävässä protossa yksikään tiedosto ei mennyt heikosti ylläpidettävien tiedostojen skaalalle. Suunnittelumalleja hyödyntävä proto ei siis sisällä samankaltaista ylläpidettävyyden ongelmakohtaa, kuin suunnittelumalleja hyödyntämätön proto.

Suunnittelumallien hyödyntäminen www-sovelluksen kehityksessä vaikuttaa aineiston perusteella olevan ylläpidettävyyttä kehittävä asia ja ylläpidettävyyden kautta hieman myös tietoturvaa kehittävä. Tässä tutkimuksessa kerätty aineisto on kuitenkin niin pieni ja altis ulkoisille tekijöille, että suunnittelumallien hyödyntämisestä ei voida vetää suoraa syy-seuraus-suhdetta asioiden välille. Tämä tutkimus antaa kuitenkin suuntaa sille, että suunnittelumalleista voi olla hyötyä www-sovellusten kehityksessä ylläpidettävyyden ja tietoturvan kannalta.

7.1 Neuvoja suunnittelumallien hyödyntämiseen

Suunnittelumallit eivät ole korvike kehittäjän ammattitaidolle. Suunnittelumallit mahdollistavat hyvien ideoiden dokumentoinnin ja välittämisen eri kehittäjien ja tahojen välillä. Suunnittelumallien avulla voidaan auttaa hyvien tapojen noudattamista ja auttaa kehittäjää ohjaamaan ajatusta oikeaan suuntaan. Oikein dokumentoituna suunnittelumallit ovat yleisiä, eikä niiden sisältämä oppi vanhene nopeasti.

Suunnittelumallit ovat vahvimmillaan silloin, kun kehittäjällä on edessään jokin kompleksinen ongelma, jota ei ole aiemmin ratkaissut mielekkäällä tavalla. Ongelman ratkaiseva suunnittelumalli voi tarjota suoraan hyvän idean ongelman ratkaisuun, tai auttaa kehittäjää pilk-

komaan ongelmaa helpommin hallittaviin palasiin. Lisäksi suunnittelumallit tarjoavat kehittäjille ja eri tahoille termejä ja kokonaisuuksia, jotka helpottavat kompleksisista suunnitteluongelmista keskustelua abstraktimmalla tasolla.

Toisaalta suunnittelumallit voivat olla myös vaarallisia. Jos kehittäjä päättää hyödyntää jotain hyvässä maineessa olevaa suunnittelumallia ilman suoraa tarvetta suunnittelumallille, niin suunnittelumalli voi tuoda ohjelmistoon vain ylimääräistä monimutkaisuutta. Mielestäni on tärkeää, että suunnittelumalleja hyödynnetään hyvin ongelmalähtöisesti. Suunnittelumallit ovat ratkaisumalleja esille tulleisiin ongelmiin.

7.2 Tutkimuksen luotettavuus

Tutkimuksessa hyödynnetyt protot on toteuttanut sama henkilö. Ensimmäiseksi toteutetusta protosta on voinut karttua oppia ja ymmärrystä toteutettavan proton luonteesta. Tämä oppi voi vaikuttaa positiivisesti toisena kehitettyyn protoon. Lisäksi ensimmäisenä kehitetty proto on kehitetty osissa vaatimusten kehittyessä, kun taas toisena kehitetty proto on kehitetty suoraan tiedossa olevien vaatimusten pohjalta. Toisena kehitetyssä protossa on myös hyödynnetty ensimmäistä protoa varten kehitettyjä näkymiä, sekä otettu pohjaa ohjaimille. Lisäksi ensimmäinen proto on toteutettu ilta- ja viikonlopputöinä, kun taas toinen proto on toteutettu varsinaisina työpäivinä, vaikka työpäivät ovat olleetkin normaalia toimistotyöaikaa pidempiä.

Edellä mainitut kokonaisuudet voivat selittää kehitettyjen protojen välillä havaittua eroa ylläpidettävyydessä. Lisäksi käytetyt ylläpidettävyyden ja tietoturvan metriikat sisältävät paljon rajoitteita. Tarkemmalla protojen tarkastelulla protojen väliltä on mahdollista löytää paljon muitakin eroja, kuin tässä tutkimuksessa on havaittu.

7.3 Jatkotutkimushaasteet

Tässä tutkimuksessa löydettiin viitteitä suunnittelumallien hyödyllisyydestä ylläpidettävyyden kehittämiseksi. Jatkotutkimuksessa voitaisiin kehittää useampia protoja ja hajauttaa kehitys useammalle eri taholle. Useampia protoja kehittämällä voitaisiin pienentää ulkoisten

tekijöiden vaikutusta tuloksiin. Lisäksi protoista voisi tehdä suurempia, jotta suunnittelumallien hyötyjä saataisiin paremmin esille.

Jatkotutkimuksessa ylläpidettävyyden mittaamista voisi myös monipuolistaa ulkoisiin mittoihin ja arvosteluihin. Ylläpidettävyyttä voisi testata esimerkiksi tarkastelemalla kuinka paljon resursseja kuluu kehitettyjen protojen laajentamiseen tietyllä ominaisuudella. Tietoturvan mittaamista voisi myös monipuolistaa ja tarkastella laajemmilla testeillä.

Lähteet

Castronova, Edward, ja Joshua Fairfield. *Dragon kill points: A summary whitepaper.* SSRN: https://ssrn.com/abstract=958945, 2007.

Chong, Stephen, ym. ”Secure web applications via automatic partitioning.” *ACM SIGOPS Operating Systems Review* 41, nro 6 (2007): 31 - 44.

Fetterly, Dennis, Mark Manasse, Marc Najork, ja Janet Wiener. ”A large-scale study of the evolution of web pages.” *Proceedings of the 12th International Conference on World Wide Web.* Budapest: ACM, 2003. 669--678.

Fonseca, José, Marco Vieira, ja Henrique Madeira. ”Evaluation of Web Security Mechanisms Using Vulnerability amp; Attack Injection.” *IEEE Transactions on Dependable and Secure Computing* 11, nro 5 (2014): 440-453.

Gamma, Erich, Richard Helm, Ralph Johnson, ja John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education India, 1995.

Github. *Web application frameworks.* 2017. https://github.com/showcases/web-application-frameworks (haettu 22. syyskuu 2017).

Gran, Ernst Gunnar, ja Sven-Arne Reinemo. ”Dragon Kill Points: Loot Distribution in MMORPGs.” *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games* (ACM) NetGames '08 (2008): 100 - 101.

Gupta, Shivangi, ja Saru Dhir. ”Issues, Challenges and Estimation Process for Secure Web Application Development.” *2016 Second International Conference on Computational Intelligence Communication Technology (CICT).* 2016. 219-222.

Hegedűs, Péter, Dénes Bán, Rudolf Ferenc, ja Tibor Gyimóthy. ”Myth or reality? analyzing the effect of design patterns on software maintainability.” *Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity* (Springer), 2012: 138 - 145.

Heitlager, Ilja , Tobias Kuipers, ja Joost Visser. ”A practical model for measuring maintainability.” *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference.* IEEE, 2007. 30 - 39.

Hoffman, Kevin, ja Patrick Eugster. ”Towards Reusable Components with Aspects: An Empirical Study on Modularity and Obliviousness.” *Proceedings of the 30th International Conference on Software Engineering.* ACM, 2008. 91 - 100.

ISO/IEC 25010. *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.* 2011.

ISO/IEC 9126. *Software engineering -- Product quality.* 2001.

Koskimies, Kai, ja Tommi Mikkonen. *Ohjelmistoarkkitehtuurit.* Talentum, 2005.

Laravel. *Database: Query Builder*. 2017. https://laravel.com/docs/5.5/queries (haettu 14. lokakuu 2017).

Lépine, Jean-François. *PhpMetrics.* 2015. http://www.phpmetrics.org/ (haettu 8. lokakuu 2017).

Marcil, Jonathan. *phpcs-security-audit.* 22. joulukuu 2015. https://github.com/FloeDesignTechnologies/phpcs-security-audit (haettu 10. lokakuu 2017).

Mawal, Ali, ja Elish Mahmoud O. ”A comparative literature survey of design patterns impact on software quality.” *Information Science and Applications (ICISA), 2013 International Conference.* IEEE, 2013. 1 - 7.

Mickens, James. ”Lecture 8: Web Security Model.” *MIT OpenCourseWare 6.858 Computer Systems Security.* 2014.

Mickens, James. ”Lecture 9: Securing Web Applications.” *MIT OpenCourseWare 6.858 Computer Systems Security.* 2014.

OWASP. *Source Code Analysis Tools.* The Open Web Application Security Project. 29. syyskuu 2017. https://www.owasp.org/index.php/Source_Code_Analysis_Tools (haettu 10. lokakuu 2017).

—. *Testing for SQL Injection (OTG-INPVAL-005).* The Open Web Application Security Project. 26. huhtikuu 2016. https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005) (haettu 10. lokakuu 2017).

Pop, Dragos-Paul, ja Adam Altar. ”Designing an MVC Model for Rapid Web Application Development.” *Procedia Engineering* 69, nro Supplement C (2014): 1172 - 1179.

Riaz, Mehwish, Emilia Mendes, ja Ewan Tempero. ”A Systematic Review of Software Maintainability Prediction and Metrics.” *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement.* IEEE Computer Society, 2009. 367 - 377.

Rychkova, Irina , Fabrice Boissier, Hassane Chraibi, ja Valentin Rychkov. *A Pragmatic Approach for Measuring Maintainability of DPRA Models.* arXiv:1706.02259, 2017.

Shvets, Alexander. *Template Method Design Pattern.* 2015. https://sourcemaking.com/design_patterns/template_method (haettu 15. syyskuu 2017).

W3Schools. *SQL Injection.* 2017. https://www.w3schools.com/sql/sql_injection.asp (haettu 10. lokakuu 2017).

W3Techs. *Usage of server-side programming languages for websites.* Q-Success. 2017. https://w3techs.com/technologies/overview/programming_language/all (haettu 15. Syyskuu 2017).

Verifysoft Technology GmbH. *Measurement of Halstead Metrics with Testwell CMT++ and CMTJava (Complexity Measures Tool).* 10. elokuu 2017. http://www.verifysoft.com/en_halstead_metrics.html (haettu 8. lokakuu 2017).

Zeldovich, Nickolai. ”Lecture 1: Introduction, Threat Models.” *MIT OpenCourseWare 6.858 Computer Systems Security*. 2014.

Zhang, Cheng, ja David Budgen. ”What Do We Know about the Effectiveness of Software Design Patterns?” *IEEE Transactions on Software Engineering* 38, nro 5 (2012): 1213 - 1231.

Liitteet

A Kehitettävien protojen vaatimukset

1. Kirjautuminen
    - Järjestelmään voi kirjautua käyttäjänimellä ja salasanalla
    - Järjestelmästä voi kirjautua ulos painikkeella
2. Kaikille avoimet näkymät
    - Kirjautumisnäkymä, johon voidaan syöttää käyttäjänimi ja salasana ja painaa lähetä.
        - Jos kirjautuminen onnistuu, näkymä ohjautuu kojelautanäkymään.
        - Jos kirjautuminen epäonnistuu, käyttäjälle esitetään ilmoitus epäonnistumisesta.
    - Kojelauta näkymä, josta näkee kaikki järjestelmässä olevat hahmot. Hahmojen listauksesta tulee saada selville seuraavat tiedot:
        - nimi, luokka, rooli, ansaitut pisteet, kulutetut pisteet, normalisoidut pisteet, nykyinen pistetilanne
    - Kojelauta näkymän rivejä tulee voida suodattaa nimen perusteella.
    - Kojelauta näkymässä olevan hahmon nimeä painamalla voidaan avata hahmon tarkemmat tiedot.
    - Hahmon tiedot näkymässä tulee selvitä seuraavat tiedot:
        - Hahmon nimi, luokka ja rooli.
        - Hahmon vastaanottamat tavarat. Tavaran yhteydessä tulee selvitä seuraavat asiat:
            - Tavaran nimi, tavaran hinta, linkki tapahtumaan, josta se on saatu.
        - Tapahtumat, joihin hahmo on osallistunut. Tapahtuman yhteydessä tulee selvitä seuraavat tiedot:
            - Linkki tapahtumaan, tapahtuman kommentti, tapahtuman kerryttämät pisteet, tapahtuman päivämäärä

- Tapahtumat näkymä, jossa esitetään kaikki järjestelmään kirjatut tapahtumat. Kustakin tapahtumasta tulee esittää seuraavat tiedot:
  - Linkki tapahtumaan, tapahtuman kerryttämät pisteet, tapahtuman kommentti, tapahtumaan osallistuneiden hahmojen määrä, tapahtuman päivämäärä
- Yksittäisen tapahtuman näkymä, jossa esitetään tapahtuman tiedot. Näkymän tulee esittää tapahtumasta seuraavat tiedot:
  - Nimi, päivämäärä, tapahtuman kerryttämät pisteet, tapahtumassa saadut tavarat, tapahtumaan liittyvät hahmoille asetetut pisteiden muutokset ja tapahtumaan osallistuneet hahmot.
  - Tapahtumassa saaduista tavaroista tulee esittää:
    - nimi, hinta, tavaran saanut hahmo. Hahmon nimi toimii linkkinä hahmon tarkempiin tietoihin.
  - Tapahtumassa hahmoille asetetuista pisteiden muutoksista tulee esittää seuraavat tiedot:
    - Hahmo jota muutos koskee, muutoksen pistemäärä ja syy muutokselle.
  - Tapahtumaan osallistuneista hahmoista tulee esittää seuraavat tiedot:
    - Hahmon nimi, hahmon luokka, hahmon rooli.
- Tilastotieto näkymä, jossa esitetään järjestelmän hahmoihin liittyvää tilastotietoa. Tilastotieto näkymässä tulee esittää vähintään seuraavat tiedot:
  - Hahmon nimi joka toimii linkkinä hahmon tarkempiin tietoihin, tieto siitä kuinka moneen viimeisimmästä kymmenestä tapahtumasta hahmo on osallistunut, sekä tieto siitä kuinka moneen tapahtumaan hahmo on koko historiansa aikana osallistunut

3. Kirjautuneille käyttäjille avoimet näkymät
   - Kirjautuneen käyttäjän näkymiin tulee päästä vain, mikäli on kirjautunut järjestelmään.
   - Tapahtumien hallinta näkymä, jossa voidaan lisätä uusi tapahtuma, poistaa olemassa oleva tapahtuma tai siirtyä muuttamaan olemassa olevia tapahtumia.

        - Tapahtuman lisäämiseksi tulee syöttää piste arvo, kommentti ja päivämäärä. Mikäli syöte on virheellinen tai puutteellinen, niin esitetään virhe.
        - Tapahtumien listauksessa esitetään linkki normaaliin näkymään, tapahtuman pistemäärä, tapahtuman kommentti, tapahtumaan osallistuneiden hahmojen määrä, tapahtuman päivämäärä, sekä linkit tapahtuman muokkaamiseen ja poistamiseen.
        - Mikäli tapahtuman poistamisen painiketta painetaan, niin käyttäjältä kysytään varmistus poistamisesta. Mikäli käyttäjä hyväksyy poiston, niin tapahtuma ja siihen liittyvät tiedot poistetaan.
    - Tapahtumien muokkaus näkymä. Muokkaus näkymässä voidaan muokata tapahtuman tietoja, lisätä ja poistaa uusia tavaroita, sekä pisteiden muutoksia. Lisäksi näkymässä tulee pystyä muokkaamaan tapahtumaan osallistuneita hahmoja.
        - Tapahtuman muokattavat tiedot ovat pistemäärä, kommentti ja tapahtuman päivämäärä
        - Tavaran lisäämiseksi tulee valita hahmo, jolle tavara lisätään, tavaran hinta, sekä tavaran nimi.
        - Pisteiden muutoksen lisäämiseksi tulee valita hahmo, jolle muutos tehdään, muutoksen arvo, sekä muutokseen liittyvä kommentti.
        - Tapahtumaan osallistuvien hahmojen lisääminen ja poistaminen pitää pystyä tekemään hahmojen listauksesta.
    - Hahmojen hallinta näkymä, jossa voidaan lisätä uusi hahmo, poistaa olemassa oleva hahmo tai siirtyä muuttamaan olemassa olevia hahmoja.
        - Hahmon lisäämiseksi tulee syöttää nimi, hahmon luokka ja rooli. Mikäli syöte on virheellinen tai puutteellinen, niin esitetään virhe.
        - Hahmojen listauksessa esitetään linkki normaaliin hahmo näkymään, hahmon luokka, hahmon rooli sekä linkit hahmon muokkaamiseen ja poistamiseen.

        - Mikäli hahmon poistamisen painiketta painetaan, niin käyttäjältä kysytään varmistus poistamisesta. Mikäli käyttäjä hyväksyy poiston, niin hahmo ja siihen liittyvät tiedot poistetaan.
    - Hahmon muokkaus näkymä. Muokkausnäkymässä voidaan muuttaa hahmon nimeä, luokkaa ja roolia.
        - Mikäli käyttäjän valinnat ovat virheellisiä tai puutteellisia, niin käyttäjälle esitetään virhe.
    - Normalisointi näkymä. Näkymässä voidaan lisätä uusi normalisointi kaikille järjestelmässä oleville käyttäjille. Näkymässä listataan kaikki tehdyt normalisoinnit. Lisäksi näkymästä voidaan siirtyä muokkaamaan viimeisintä käyttäjille tehtyä normalisointia.
        - Normalisoinnin lisäämistä varten käyttäjän tulee syöttää normalisoinnin prosenttiyksikkö, sekä normalisoinnin syytä kuvaava kommentti.
            - Mikäli käyttäjän valinnat ovat virheellisiä tai puutteellisia, niin käyttäjälle esitetään virhe.
        - Normalisointia esittävällä rivillä esitetään normalisoinnin lisääjän nimi, normalisoinnin suuruus, normalisointiin liittyvä kommentti, normalisoinnin päivämäärä, sekä painike normalisoinnin poistamiseen.
            - Normalisoinnin poisto-painiketta painettaessa käyttäjältä kysytään varmistus poistamisesta. Mikäli käyttäjä hyväksyy poiston, niin normalisointi poistetaan järjestelmästä.
    - Viimeisimmän normalisoinnin muokkaamisen näkymä.
        - Näkymässä listataan kaikki hahmot, joille normalisointi on laskettu. Rivillä esitetään hahmon nimi, normalisoitujen pisteiden suuruus, sekä painike, josta käyttäjän normalisoinnin määrän voi laskea uudelleen.
4. Tietorakenteen toiminta
    - Hahmojen luokat pitää olla muokattavissa tietokannassa.
    - Hahmojen roolit pitää olla muokattavissa tietokannassa.

- Normalisoinnin aiheuttamat muutokset pisteisiin pitää olla erotettavissa muista pistemuutoksista.
- Tavaroihin kulutetut pisteet pitää olla erotettavissa muista pistemuutoksista.
- Hahmoille tehdyt pisteiden muutokset tapahtumissa vaikuttavat hahmojen ansaitsemien pisteiden laskuun.

B Koodin koko –esimerkki

```
<?php
/**
 * SimpleClass just for displaying differences in volume
measures.
 *
 * @author Jere Junttila jere.a.s.junttila@student.jyu.fi
 * @license GPL
 */
class SimpleClass {
    private $intValue;

    /**
     * Create a new instance of SimpleClass. SimpleClass
stores value of one integer.
     *
     * @return    SimpleClass       Newly created instance
of SimpleClass.
     */
    public function __construct() {
        //Set arbitary default to intValue.
        $this->intValue = 11;
    }
```

```
    /**
     * Set new value for the class stored integer.
     *
     * @param     integer  $number  Number that is set.
     *
     * @return    boolean  True, if value was set, false
otherwise.
     */
    public function setInt($number) {
        //If $number is indeed integer, update the value,
otherwise keep old value.
        if(is_int($number)){
            $this->intValue = $number;
            //Value was updated, return true.
            return true;
        }
        //Value wasn't updated, return false.
        return false;
    }

    /**
     * get the stored integer value.
     *
     * @return    integer  Value of class' stored integer.
     */
    public function getInt() {
        return $this->intValue;
    }
}
```

C Tietoturvan testaus-syötteet

Näkymien syötteiden testaus

1.

Näkymä: Kirjautuminen '/login'

Syötteet:

Username: 1' or '1' = '1

Password: 1' or '1' = '1

Tavoite:

SQL: SELECT * FROM users WHERE name='1' OR '1' = '1' AND pas word='1' OR '1' = '1'

#Muutetaan tarkastukset muotoon, jotka ovat aina tosia.

2.

Näkymä: Kirjautuminen '/login'

Syötteet:

Username: 1' or '1' = '1' LIMIT 1/*

Password: foo

Tavoite:

SQL: SELECT * FROM users WHERE name='1' or '1' = '1' LIMIT 1/*' AND password='foo'

#Haetaan vain ensimmäinen, palautuksen määrän tarkistuksista päästään ohi.

3.

Näkymä: Hahmolistaus / lisäys '/character_management'

Syötteet:

Name: test' ,1 ,1); DROP TABLE classes

Class: 1

Role: 1

Tavoite:

#Lopetetaan insert ja pudotetaan taulu. Jos onnistuu, löydetty heikkous.

4.

Näkymä: Hahmon muokkaus 'modify_character/6'

Syötteet:

Name: test', char_role = 1 WHERE char_id=1; DROP TABLE classes

Class: 2

Role: 3

Tavoite:

#Lopetetaan update ja pudotetaan taulu. Jos onnistuu, löydetty heikkous.

5.

Näkymä: Raidlistaus / lisäys '/raid_management'

Syötteet:

DKP Value: 20' ,'inject' , '2017-02-02'); DROP TABLE classes

Comment: inject' , '2017-02-02'); DROP TABLE classes

Date: 2017-02-02

Tavoite:

#Lopetetaan insert ja pudotetaan taulu. Jos onnistuu, löydetty heikkous.

6.

Näkymä: Raid muokkaus 'modify_raid/9'

Syötteet:

DKP Value: 21' , raid_date = '2017-02-02' WHERE raid_id = 9; DROP TABLE classes

Comment: inject' , raid_date = '2017-02-02' WHERE raid_id = 9; DROP TABLE classes

Date: 2017-02-02

Tavoite:

#Lopetetaan update ja pudotetaan taulu. Jos onnistuu, löydetty heikkous.

7.

Näkymä: Raid muokkaus / tavaran lisäys '/modify_raid/9'

Syötteet:

Character: 1' ,'20' , 'inject'); DROP TABLE classes

Item Price: 20' , 'inject'); DROP TABLE classes

Item Name: inject'); DROP TABLE classes

Tavoite:

#Lopetetaan insert ja pudotetaan taulu. Jos onnistuu, löydetty heikkous.

8.

Näkymä: Raid muokkaus / muutoksen lisäys '/modify_raid/9'

Syötteet:

Character: 1' ,'20' , 'inject'); DROP TABLE classes

DKP Price: 20' , 'inject'); DROP TABLE classes

Comment: inject'); DROP TABLE classes

Tavoite:

#Lopetetaan insert ja pudotetaan taulu. Jos onnistuu, löydetty heikkous.

9.

Näkymä: Normalisaatio listaus / lisäys '/normalization_management'

Syötteet:

Percent: 20' ,'inject'); DROP TABLE classes

Comment: inject'); DROP TABLE classes

Tavoite:

#Lopetetaan insert ja pudotetaan taulu. Jos onnistuu, löydetty heikkous.

URL syötteiden testaus

10.

URL: /raid/8 AND 1=2

Tavoite:

#Jos palautus on tyhjä, testaa '/raid/8 AND 1=1'. Jos toinen testattu palauttaa taas datan, tiedetään että heikkous on löydetty yksittäisen raidin id käsittelystä.

11.

URL: /char/2 AND 1=2

Tavoite:

#Jos palautus on tyhjä, testaa '/char/2 AND 1=1'. Jos toinen testattu palauttaa taas datan, tiedetään että heikkous on löydetty yksittäisen hahmon id käsittelystä.

12.

URL: /modify_character/14 AND 1=2

Tavoite:

#Jos palautus on tyhjä, testaa '/modify_character/14 AND 1=1'. Jos toinen testattu palauttaa taas datan, tiedetään että heikkous on löydetty yksittäisen hahmon muokkauksen id käsittelystä.

13.

URL: /modify_raid/5 AND 1=2

Tavoite:

#Jos palautus on tyhjä, testaa 'modify_raid/5 AND 1=1'. Jos toinen testattu palauttaa taas datan, tiedetään että heikkous on löydetty yksittäisen hahmon muokkauksen id käsittelystä.

14.

URL: /delete_raid?raid_id=5

Tavoite:

#Kokeile antaako delete ottaa vastaan GET parametrin normaalisti, jos onnistuu testaa kuten yksittäinen raid.

15.

URL: /delete_raid/item?item_id=5

Tavoite:

#Kokeile antaako delete ottaa vastaan GET parametrin normaalisti, jos onnistuu testaa kuten yksittäinen raid.

16.

URL: /delete_raid/adjustment?raid_id=5&char_id=2

Tavoite:

#Kokeile antaako delete ottaa vastaan GET parametrin normaalisti, jos onnistuu testaa kuten yksittäinen raid.

17.

URL: /delete_character?char_id=2

Tavoite:

#Kokeile antaako delete ottaa vastaan GET parametrin normaalisti, jos onnistuu testaa kuten yksittäinen raid.

18.

URL: /delete_normalization?normalization_id=1

Tavoite:

#Kokeile antaako delete ottaa vastaan GET parametrin normaalisti, jos onni tuu testaa kuten yksittäinen raid.

Jatkoideoita, mikäli haavoittuvuus löytyy.

URL: /raid/10; INSERT INTO users (id, name, email, password, created_at, updated_at, remember_token) VALUES (NULL, 'test', 'test', 'test', NULL, NULL, NULL);

Tavoite:

#Saadaan lisättyä käyttäjä tauluun

URL: /raid/8 ORDER BY 10--

Tavoite:

#Valitaan order by numero niin suureksi, että ensiksi saadaan virhe. Pienennetään määrää kunnes on selvitetty haettujen kolumnien määrä. Mahdollistaa UNION käytön.

URL: /raid/8 AND IF(version() like '5%', sleep(10), 'false'))--

Tavoite:

#Jos vastauksessa kestää 10 sekuntia, niin tiedetään että käytössä on MySql versio 5.X

D phpcs-security-audit -tulokset

SoulDKP

FILE: /var/www/souldkp/public/index.php

FOUND 0 ERROR(S) AND 2 WARNING(S) AFFECTING 2 LINE(S)

---

22 | WARNING | Possible RFI detected with __DIR__ on require

36 | WARNING | Possible RFI detected with __DIR__ on require_once

---

JelaDKP

FILE: /var/www/jeladkp/database/seeds/DefaultUsersSeeder.php

FOUND 0 ERROR(S) AND 1 WARNING(S) AFFECTING 1 LINE(S)

---

18 | WARNING | Crypto function bcrypt used.

FILE: /var/www/jeladkp/public/index.php

FOUND 0 ERROR(S) AND 2 WARNING(S) AFFECTING 2 LINE(S)

---

22 | WARNING | Possible RFI detected with __DIR__ on require

36 | WARNING | Possible RFI detected with __DIR__ on require_once

---

FILE: /var/www/jeladkp/app/Models/PointsUsedModel.php

FOUND 0 ERROR(S) AND 4 WARNING(S) AFFECTING 2 LINE(S)

---

38 | WARNING | Potential SQL injection in where with param #1

38 | WARNING | Potential SQL injection with direct variable usage in where | | with param #1

```
 67 | WARNING | Potential SQL injection in where with param #1
 67 | WARNING | Potential SQL injection with direct variable usage in where
    |         | with param #1
--------------------------------------------------------------------------------

FILE: /var/www/jeladkp/app/Models/RaidAdjustmentModel.php
--------------------------------------------------------------------------------
FOUND 0 ERROR(S) AND 2 WARNING(S) AFFECTING 2 LINE(S)
--------------------------------------------------------------------------------
 83 | WARNING | Filesystem function delete() detected with dynamic parameter
 87 | WARNING | Potential SQL injection in where with param #1
--------------------------------------------------------------------------------

FILE: /var/www/jeladkp/app/Models/Model.php
--------------------------------------------------------------------------------
FOUND 0 ERROR(S) AND 11 WARNING(S) AFFECTING 7 LINE(S)
--------------------------------------------------------------------------------
  78 | WARNING | Potential SQL injection with direct variable usage in fields
     |         | with param #1
 149 | WARNING | Filesystem function delete() detected with dynamic parameter
 153 | WARNING | Potential SQL injection in where with param #1
 153 | WARNING | Potential SQL injection with direct variable usage in where
     |         | with param #1
 234 | WARNING | Potential SQL injection in where with param #1
 234 | WARNING | Potential SQL injection with direct variable usage in where
     |         | with param #1
 234 | WARNING | Potential SQL injection with direct variable usage in where
     |         | with param #1
 276 | WARNING | Potential SQL injection with direct variable usage in fields
     |         | with param #1
 300 | WARNING | Potential SQL injection with direct variable usage in fields
```

| | with param #1
303 | WARNING | Potential SQL injection in where with param #1
303 | WARNING | Potential SQL injection with direct variable usage in where
| | with param #1

FILE: /var/www/jeladkp/app/Models/NormalizationPointModel.php
---
FOUND 0 ERROR(S) AND 3 WARNING(S) AFFECTING 3 LINE(S)
---
99 | WARNING | Potential SQL injection with direct variable usage in fields
| | with param #1
103 | WARNING | Potential SQL injection in where with param #1
117 | WARNING | Potential SQL injection in where with param #1

FILE: /var/www/jeladkp/app/Models/NormalizationModel.php

FOUND 0 ERROR(S) AND 2 WARNING(S) AFFECTING 1 LINE(S)
---
90 | WARNING | Potential SQL injection in where with param #1
90 | WARNING | Potential SQL injection with direct variable usage in where
| | with param #1

FILE: /var/www/jeladkp/app/Models/Fields.php

FOUND 0 ERROR(S) AND 1 WARNING(S) AFFECTING 1 LINE(S)
---
34 | WARNING | Potential SQL injection with direct variable usage in fields

```
   |         | with param #1
----------------------------------------------------------------------------

FILE: /var/www/jeladkp/app/Models/RaidAttendanceModel.php
----------------------------------------------------------------------------
FOUND 0 ERROR(S) AND 5 WARNING(S) AFFECTING 4 LINE(S)
----------------------------------------------------------------------------
 44 | WARNING | Potential SQL injection with direct variable usage in fields
    |         | with param #1
104 | WARNING | Potential SQL injection with direct variable usage in fields
    |         | with param #1
110 | WARNING | Filesystem function delete() detected with dynamic parameter
141 | WARNING | Potential SQL injection in where with param #1
141 | WARNING | Potential SQL injection with direct variable usage in where
    |         | with param #1
----------------------------------------------------------------------------

FILE: /var/www/jeladkp/app/Models/StatsModel.php
----------------------------------------------------------------------------
FOUND 0 ERROR(S) AND 2 WARNING(S) AFFECTING 1 LINE(S)
----------------------------------------------------------------------------
 61 | WARNING | Potential SQL injection in groupBy with param #1
 61 | WARNING | Potential SQL injection with direct variable usage in groupBy
    |         | with param #1
----------------------------------------------------------------------------

FILE: /var/www/jeladkp/app/Http/Controllers/CharacterController.php
----------------------------------------------------------------------------
FOUND 0 ERROR(S) AND 1 WARNING(S) AFFECTING 1 LINE(S)
----------------------------------------------------------------------------
 199 | WARNING | Filesystem function delete() detected with dynamic parameter
```

```
----------------------------------------------------------------------

FILE: /var/www/jeladkp/app/Http/Controllers/NormalizationController.php
----------------------------------------------------------------------
FOUND 0 ERROR(S) AND 1 WARNING(S) AFFECTING 1 LINE(S)
----------------------------------------------------------------------
 130 | WARNING | Filesystem function delete() detected with dynamic parameter
----------------------------------------------------------------------

FILE: /var/www/jeladkp/app/Http/Controllers/RaidController.php
----------------------------------------------------------------------
FOUND 0 ERROR(S) AND 3 WARNING(S) AFFECTING 3 LINE(S)
----------------------------------------------------------------------
 196 | WARNING | Filesystem function delete() detected with dynamic parameter
 232 | WARNING | Filesystem function delete() detected with dynamic parameter
 269 | WARNING | Filesystem function delete() detected with dynamic parameter
----------------------------------------------------------------------
```

E Tietoturvan testaus -tulokset

Testitapausten tulokset SoulDKP

1. OK, incorrect details.
2. OK, incorrect details.
3. OK, hahmo lisätty annetulla nimellä.
4. OK, hahmo muokattu annetulle nimelle.
5. OK, DKP Value tarkistetaan, eikä kyselyä viedä kantaan. Testattu myös DKP Value = 20.
6. OK, DKP Value tarkistetaan, eikä kyselyä viedä kantaan. Testattu myös DKP Value = 20.
7. OK, Character ja Item Price tarkistetaan, eikä kyselyä viedä kantaan. Testattu myös Character = 1 ja Item Price = 20.

8. OK, Character ja DKP Price tarkistetaan, eikä kyselyä viedä kantaan. Testattu myös Character = 1 ja DKP Price = 20.
9. OK, Percent tarkistetaan, eikä kyselyä viedä kantaan. Testattu myös Percent = 20.
10. OK, Tiedot näytetään normaalisti.
11. OK, Tiedot näytetään normaalisti.
12. OK, Tiedot näytetään normaalisti.
13. OK, Tiedot näytetään normaalisti.
14. OK, Näytetään virhe "MethodNotAllowedHttpException".
15. OK, Näytetään virhe "MethodNotAllowedHttpException".
16. OK, Näytetään virhe "MethodNotAllowedHttpException".
17. OK, Näytetään virhe "MethodNotAllowedHttpException".
18. OK, Näytetään virhe "MethodNotAllowedHttpException".

Testitapausten tulokset JelaDKP

1. OK, incorrect details.
2. OK, incorrect details.
3. OK, hahmo lisätty annetulla nimellä.
4. OK, hahmo muokattu annetulle nimelle.
5. OK, DKP Value tarkistetaan, eikä kyselyä viedä kantaan. Testattu myös DKP Value = 20.
6. OK, DKP Value tarkistetaan, eikä kyselyä viedä kantaan. Testattu myös DKP Value = 20.
7. OK, Character ja Item Price tarkistetaan, eikä kyselyä viedä kantaan. Testattu myös Character = 1 ja Item Price = 20.
8. OK, Character ja DKP Price tarkistetaan, eikä kyselyä viedä kantaan. Testattu myös Character = 1 ja DKP Price = 20.
9. OK, Percent tarkistetaan, eikä kyselyä viedä kantaan. Testattu myös Percent = 20.
10. OK, Tiedot näytetään normaalisti.
11. OK, Tiedot näytetään normaalisti.

12. OK, Tiedot näytetään normaalisti.
13. OK, Tiedot näytetään normaalisti.
14. OK, Näytetään virhe "MethodNotAllowedHttpException".
15. OK, Näytetään virhe "MethodNotAllowedHttpException".
16. OK, Näytetään virhe "MethodNotAllowedHttpException".
17. OK, Näytetään virhe "MethodNotAllowedHttpException".
18. OK, Näytetään virhe "MethodNotAllowedHttpException".