OULUN SEUDUN
AMMATTIKORKEAKOULU

Jaana Helanen

BELLEBOOK-AJANHALLINTAJÄRJESTELMÄN TESTAUS

BELLEBOOK-AJANHALLINTAJÄRJESTELMÄN TESTAUS

Jaana Helanen
Opinnäytetyö
Kevätlukukausi 2012
Hyvinvointiteknologian koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Hyvinvointiteknologia

Tekijä: Jaana Helanen
Opinnäytetyön nimi: BelleBook-ajanhallintajärjestelmän testaus
Työn ohjaaja: Jukka Jauhiainen
Työn valmistuminen: Kevät 2012
Sivumäärä: 53

Opinnäytetyön aiheena oli tutustua erilaisiin testausmenetelmiin teoreettisesti ja niiden perusteella päättää, millä menetelmällä testataan BelleBook-sovellusta. BelleBook-sovellus on ajanvarausjärjestelmä, jota tällä hetkellä käytetään parturi-kampaamoissa. Sovellus toimii web-pohjaisena, jolloin asiakas voi varata itse ajan kirjautumalla palveluun.

Työn tavoitteena oli testata BelleBook-sovelluksen toimivuutta ja etsiä siitä virheitä, joita korjaamalla sovellusta voi parantaa. Testaus suoritettiin mustalaatikko-testausjärjestelmän avulla. Testaus suoritettiin syöttämällä etukäteen työntekijöiden työvuorot, joihin aika voitiin varata. Tämän jälkeen syötettiin asiakkaat ja kokeiltiin ajanvarausta. Työn tarkoituksena oli etsiä sovelluksesta virheitä, jotka auttavat tilaajaa sovelluksen kehittämisessä. Sovellusta testattiin erilaisilla toiminnoilla, joiden tarkoituksena oli saada joko virhetulos tai toimiva tulos. Testaukset suoritettiin yleisimmillä internetselaimilla.

BelleBook-sovelluksesta löytyi jonkin verran virheitä, jotka arvioitiin Nielsenin vakavuusasteen mukaisesti. Saatujen tulosten perusteella tilaaja voi kehittää sovellusta eteenpäin ja vähentää mahdollisten virheiden määrää tulevaisuudessa.

Asiasanat: Ajanhallintajärjestelmä, ajanvaraus, testaus, mustalaatikkotestaus

ALKULAUSE

Kiitos Pirjo Ritokangas-Huttuselle Belleviews Oyltä, joka toimi tilaajana ja antoi minulle mahdollisuuden saada tehdä tämän mielenkiintoisen opinnäytetyön.

Kiitos hyvinvointiteknologian yliopettaja, lääketieteellisen fysiikan dosentti Jukka Jauhiaiselle opinnäytetyön ohjaamisesta ja tarkastamisesta.

Kiitos Oulun seudun ammattikorkeakoulun lehtori Tuula Hopeavuori kielen ja rakenteen ohjaamisesta ja tarkastamisesta.

Kiitos hyvinvointiteknologian insinööri Anneli Hirvikoskelle tuesta ja avusta opinnäytetyön tekemisessä.

Lopuksi vielä erityiskiitos ystäville, jotka ovat jaksaneet vastailla kysymyksiin ja oikolukea työtäni.

Minun rakkaille lapsilleni, kun luette tämän. Olkaa äidistä ylpeitä, että hän aikuisiässä sai itsellensä insinööritutkinnon suoritettua. Rakastan teitä.

Oulussa 26.3.2012

Jaana Helanen

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheena oli testata BelleBook-ajanhallintajärjestelmän toimintaa sekä löytää mahdolliset virheet ohjelman käytössä. Tilaajana opinnäytetyöllä oli Belleviews Oy ja yhteyshenkilö sieltä Pirjo Ritokangas-Huttunen. Belleviews Oy on oululainen yritys, joka on perustettu 2004. Yritys on erikoistunut pelien ja hyvinvointisovellusten kehittämiseen ja tuottamiseen.

BelleBook-sovellus on tietojenhallintajärjestelmä, joka on pääsääntöisesti kehitetty ajanvarausjärjestelmäksi kauneudenhoitoalan pienyrittäjille. Ajanvarausjärjestelmän lisäksi se sisältää asiakas- sekä työntekijätiedot. Sovelluksen tarkoituksena on korvata paperiversiot ajanvarauksesta sekä helpottaa asiakkaan ajanvarausta. Sovellus toimii internetselaimella, jolloin asiakas voi varata ajan kotona omalla koneella. BelleBook-sovellus on kehitetty ja toteutettu Belleview Oy:lle opinnäytetyönä ja se on tällä hetkellä käytössä kahdella asiakkaalla.

Opinnäytetyö sisältää yleistä teoriaa sovellustestauksesta, testaustasoista sekä testausmenetelmistä. BelleBook-sovelluksen testaamiseen valittiin mustalaatikkotestaus. Mustalaatikkotestauksessa testaus tapahtuu käyttötasolla, milloin ei välitetä testattavan kohteen rakenteesta tai sisällöstä.

Työn tavoitteena oli testata BelleBook-sovellusta askel askeleelta. Jos ohjelmassa tuli virheitä vastaan, ne kirjattiin ja dokumentoitiin. Testauksen tulokset ja ongelmakohdat annettiin ohjelmiston kehittäjälle sekä ylläpitäjälle.

2 SOVELLUSTESTAUS

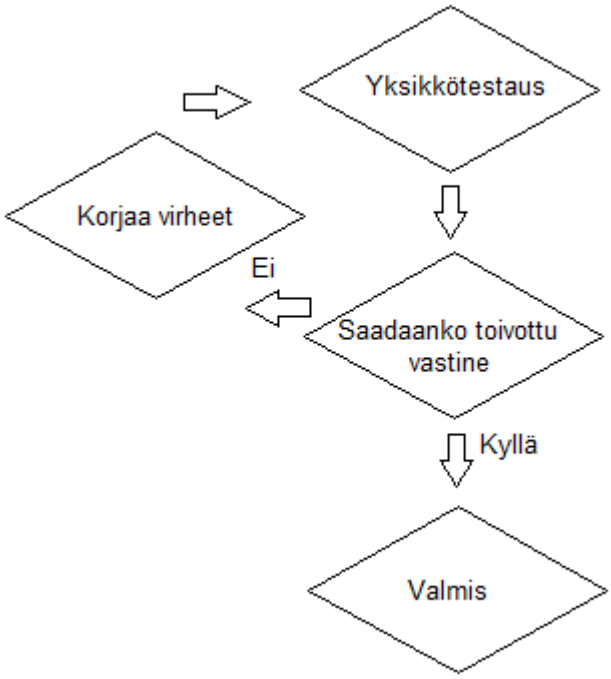
Sovelluksien testaaminen voidaan määritellä prosessiksi, jossa testaaja luo testitapauksia, joiden tarkoituksena on löytää sovelluksesta virheitä. Virheiden löytäminen on testauksen pääasia, koska kukaan loppukäyttäjä ei halua käyttää virheellistä tuotetta. Sovellustestauksen suurin tekijä on hyvä testaaja, jolta vaaditaan sekä kärsivällisyyttä että kokeilunhalua testejä tehdessä. Testaajalta odotetaan usein hyviä ohjelmointitaitoja. Tämän lisäksi katsotaan hyväksi, ettei testaaja tunne ennestään testattavaa sovellusta, jolloin testituloksista saadaan kattavammat. (Väisänen 2010.)

2.1 Testauksen tasot

Sovellustestauksessa käytetyt tavat on pääosin jaettu neljään eri tasoon sekä testausmenetelmään. Tasot ovat yksikkötestaus, integraatiotestaus, systeemitestaus ja hyväksymistestaus. Työnjohto päättää tason määräytymisen ja vaiheen, jossa ohjelmistotestaukset aloitetaan. (Pressman 2001, 477.)

2.1.1 Yksikkötestaus

Kehitettävä ohjelmisto koostuu useista yksiköistä, joita yksikkötestauksessa, testauksen ensimmäisessä vaiheessa, testataan yksitellen irrallaan muista komponenteista. Yksikkötestausta suorittaa sovelluksen kehittäjä. Yksikkötestauksen tarkoituksena on varmistaa metoditasolla, että yksittäinen koodi toimii ohjelmoijan haluamalla tavalla. Kuvassa 1 on nähtävillä yksikkötestauksen arkkitehtuurisuunnittelu vaiheittain. Jos yksi osa aiheuttaa ongelmaa, ei ongelma liity muihin osiin. Yksikkötestauksen avulla varmistetaan, että kaikki pienet osaset, jotka ovat osa ohjelmistosta, toimivat oikein. (Pressman 2001, 485–487.)



KUVA 1. Yksikkötestaus

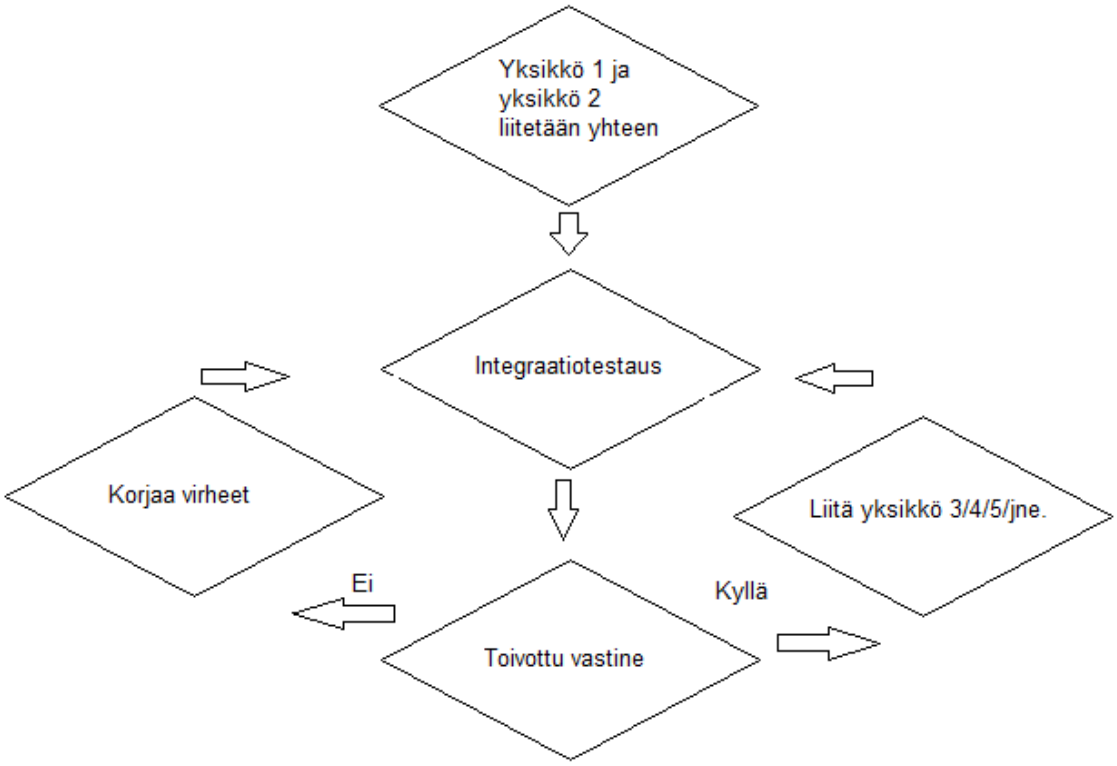
Yksikkötestausta pidetään usein koodausta tukevana työvaiheena ja usein näitä suoritetaan rinnakkain. Yksikkötestaus kuuluu olennaisesti ohjelmistokehitysprosessiin, joten yksiköiden on välttämätöntä vastata määrittelyyn, ennen kuin ne annetaan seuraavassa ohjelmistoprosessin vaiheessa yhdistettäviksi toisiin yksiköihin. (Pressman 2001, 485–487.)

Yksikkötestaus käsittää yksikön tietorakenteen, ohjelman logiikan ja ohjelman rakenteen tutkimisen yksikön kannalta. Yksikkö ei yleensä toimi itsenäisenä ohjelmana. Testauksessa vaaditaan testiympäristö, joka koostuu pääohjelmasta ja riittävästä määrästä avustavaa koodia. Joskus joudutaan toteuttamaan testipetejä, joilla yksikön toimivuutta kokeillaan. Yksikkötestaus yksinkertaistuu ja on parhaimmillaan silloin, kun yksiköt suunnitellaan niin, että niillä on mahdollisimman vähän tehtäviä. Hyvässä tapauksessa vain yksi tehtävä suoritetaan yhdessä yksikössä. Tällöin testitapauksia ei tarvita kovin paljon ja virheiden havaitseminen ja paikallistaminen käy mahdollisimman helposti. (Pressman 2001, 485–487.)

Yksikkötestauksessa on välttämätöntä testata funktiot kaikilla määritetyillä syötteillä, jolloin sen pitää tuottaa kaikki mahdolliset vasteet. Testauksessa tarkistetaan, että syötteenä annetut ja saadut parametrit ovat virheettömiä, niiden määrä on oikea ja tyypit sopivat yhteen sekä järjestys on sama kaikkialla. Paikallisten tietorakenteiden testaamisella varmennetaan, että kaikki väliaikaisesti tallennettu tieto säilyy oikeana koko sovelluksen ajon ajan. Tietorakenteiden testitapausten on käsiteltävä mahdolliset kirjoitusvirheet, muuttujien väärät alustukset ja oletusarvojen käyttö. Lisäksi on tarkastettava muuttujien nimien sopivuus ja oikea käyttö sekä muuttujien ali- ja ylikuormitustilanteista johtuvat poikkeukset. (Marttinen 2010, 7–9.)

2.1.2 Integraatiotestaus

Integraatiotestaus, joka näkyy kuvassa 2, on rajapintojen testaamista. Sillä tarkoitetaan vaihetta, jossa pienemmät yksiköt kasataan yhteen ja testataan kunnes koko systeemi on saatu kasaan. Rajapinta on se missä laitteisto ja ohjelmisto keskustelevat keskenään. Järkevin integraatiotestaus on sellainen, jossa komponentteja liitetään yksitellen ja testataan joka vaihe, jolloin virheen tullessa tiedetään, missä vaiheessa vika tuli. Toisen tyyppinen integraatiotestaus, jota kutsutaan nimellä ”big-bang”, on testaus jossa kasataan kaikki komponentit yhteen ja testataan toimivuutta. Tämä käytäntö on virheiden löytämisen kannalta työläintä. Integraatiotestauksen tarkoituksena on varmistaa, että kasatut osat toimivat määritellysti keskenään sekä löytää ja korjata mahdollisimman paljon yhtyeensoveltumattomia komponentteja. (Pressman 2001, 488–491.)



KUVA 2. Integraatiotestaus, yksiköt lisätään yksitellen

Integraatiotestaus on paljon kontrolloidumpi testausvaihe kuin yksikkötestaus. Useimmiten kehittäjät suorittavat integraatiotestauksen, mutta sen voi suorittaa myös testausryhmä. Testauksen toteutuksessa käytetään ennalta määrättyjä testitapauksia, jotka on kirjoitettu arkkitehtuurisuunnittelun ja järjestelmän määrittelyjen pohjalta. Integraatiotestaukseen vaikuttavat sovelluksen laadulliset tekijät, joita ovat koheesio ja kytkentä. Koheesio kertoo yksikön voimasta eli sen tehtävien itsenäisyydestä. Koheesio on korkea silloin, kun yksikkö toimii itsenäisesti suorittaen yksittäistä tehtävää ja kommunikoi vain rajallisesti muiden yksiköiden kanssa. Kytkentä taas mittaa eri moduulien välisiä kytkentöjä. Toisten yksiköiden toimintoja kutsuvat yksiköt ovat kytkettyjä. Integraatiotestaus on helpompaa silloin, kun koheesio on korkea ja vastaavasti kytkentä on matala. Integraatiotestaus voidaan aloittaa heti, kun osa järjestelmän yksiköistä on koottu yhteen testattavaksi kokonaisuudeksi. (Pressman 2001, 488–491.)

Paras tapa testata on aloittaa liittämällä toisiinsa ensin kaksi komponenttia ja testaamalla nämä. Kun testaus on hyväksyttävästi suoritettu, liitetään kolmas komponentti. Jos tämän jälkeen esiintyy virheitä, tiedetään virheiden melko todennäköisesti johtuvan kolmannen komponentin työskentelystä kahden ensimmäisen kanssa. Tällä tekniikalla voidaan suorittaa koko järjestelmän integraatiotestaus ja saavuttaa virheiden paikallistamisessa mahdollisimman hyvä löytämisprosentti, koska viimeksi lisätty komponentti on yleensä todennäköisin virheiden aiheuttajana. Toisaalta lisättäessä kolmas komponentti saattaa kahden ensimmäisen komponentin keskinäinen toiminta muuttua, mikä voi johtaa uusien virheiden syntyyn. Kolmantena käytännön vaikeutena on se, että virheiden korjaaminen monimutkaisessa integroidussa järjestelmässä voi aiheuttaa virheitä muissa, aiemmin toimineissa komponenteissa. (Pressman 2001, 488–491.)

Integraatiotestauksen kokoava testaus on kehitysprosessissa alemman tason komponentin kehittäminen ja testaaminen ennen siirtymistä korkeamman tason kehitykseen. Kokoava testaus aloitetaan alemman tason pienistä komponenteista ja suunnataan testausta vähitellen ylemmille tasoille. Testaus aloitetaan integroimalla alimman tason yksiköt yksitellen mukaan sovellukseen ja testaamalla nämä. Sen jälkeen integroidaan seuraavalla tasolla olevat yksiköt alimman tason kanssa. Tätä tekniikkaa jatketaan, kunnes kaikkein ylin taso mukaan lukien on integroitu ja testattu. Integraatiotestauksessa ajurit ovat suhteellisen yksinkertaisia koodata, koska niiden päätarkoitus on vain kutsua testattavana olevaa yksikköä, jotta sen suoritusta voidaan tarkkailla. Yleensä ajureiden tehtävänä on tarjota syötteet ja ottaa vastaan vasteet yksiköltä sekä tulostaa tulokset ruudulle testaajan tarkastettavaksi. (Pressman 2001, 488–491.)

Joissakin sovelluksissa kokoava testaus saattaa olla integraatiotestauksessa ainoa vaihtoehto. Tämä johtuu siitä, että kokonaisuus muodostuu osista, eikä kokonaisuuden testaaminen siten voi olla ensimmäinen vaihe. Tämä pätee myös järjestelmiin, joissa uudelleenkäytetään ja muokataan komponentteja muista järjestelmistä. Näissä komponentit on jo kertaalleen todettu toimiviksi,

jolloin ne teoriassa toimivat yksinään ja ainoa mahdollinen virheitä tuottava kohta voi olla niiden liittäminen yhteen sekä kommunikointi toisten komponenttien kanssa. (Pressman 2001, 488–491.)

2.1.3 Systeemitestaus

Systeemitestauksen tarkoituksena on testata koko järjestelmää sen käyttötarkoitusta vastaavassa ympäristössä. Siinä ei testata yksittäisiä funktioita, vaan osoitetaan poikkeavuudet tuotteen sekä sen vaatimusten ja dokumentaation välillä. Testaajiksi tulisi valita henkilöitä, jotka eivät ole osallistuneet kehitystyöhön. Systeemitestauksessa testataan myös tekniset ominaisuudet. (Sommerville 2004, 540–547.)

Systeemitestauksen tarkoituksena on pyrkiä selvittämään sovelluksen sekä siihen kuuluvien osa-alueiden toimintaa ja keskinäistä kommunikointia erilaisissa käytön aikana esiintyvissä tilanteissa. Systeemitestauksessa valmis ohjelmisto testataan sille määriteltyjen vaatimusten suhteen. Systeemitestaus pyritään suorittamaan mahdollisimman aidossa ympäristössä tai testaamista varten tehdyssä testiympäristössä. Etuna testiympäristön käytössä on mahdollisuus tehdä erilaisia suorituskyky- ja kuormitustestejä häiritsemättä kehittäjien tai jo systeemiä käyttävien henkilöiden toimintaa. Haittapuolena on mahdollinen ympäristön luonnista ja ylläpidosta koituva lisätyö. Testaajiksi ei saa valita ohjelmoijia, vaan testaajina on oltava henkilöitä, joilla ei ole ollut tekemistä kehitystyön kanssa. Näin testaajat eivät pyri tekemään testejä koodia myötäileviksi. Lisäksi heillä on suurempi kiinnostus havaita virheitä kuin itse koodin kirjoittajalla, joka mieluiten osoittaa koodinsa toimivaksi. (Sommerville 2004, 540–547; Pressman 2001, 496–497.)

Käytettävyystestauksen tarkoituksena on varmentua sovelluksen sopivuudesta, hyödyllisyydestä, helppokäyttöisyydestä ja toimivuudesta järjestelmän tuleville käyttäjäryhmille. Käytettävyystestaus on suurimmaksi osaksi käyttöliittymätestausta. Yleensä käytettävyystestausta varten ei kirjoiteta erillistä suunnitelmaa, vaan tämä vaihe sisältyy jo systeemitestaukseen. Vanhojen käyttäjien keskuudessa saattaa myös esiintyä muutosvastarintaa

uuteen siirtymistä kohtaan. Tällöin saattaa olla parempi käyttää ainakin osittain ihmisiä, jotka eivät ole käyttäneet aiempaa järjestelmää. Käytettävyystestauksen voi suorittaa myös joukko siihen erikoistuneita ihmisiä. Tällöin hyödynnetään käytettävyyslaboratorioita, joissa on laitteisto käyttäjän toiminnan rekisteröintiin. Yleensä tällöin käytetään apuna videokameraa, jonka nauhoitusta voi tarkastella myöhemmin. (Holappa 2011.)

Olennainen tarkkailtava asia testauksessa on se, onko sovelluksessa helppo navigoida eli pystyykö käyttäjä helposti syöttämään tietoa, liikkumaan ja löytämään paikasta toiseen sekä poistumaan ohjelmasta. Toisaalta on tarkasteltava, onko sovellus helppokäyttöinen, eli voiko käyttäjä tehdä haluamansa toiminnot itselleen sopivimmalla tavalla ja onko suoritustapa selkeä. Lisäksi sovelluksesta testataan tehokkuutta, eli käyttäjän on voitava suorittaa haluamansa toiminnot lyhyessä ajassa ja vähin askelin. Sovelluksen rakenteen, avustustiedostojen ja dokumentaation on tärkeää olla helposti käsitettävää. Tehokkuutta voidaan mitata tekemällä vaatimusmäärittely, kuinka kauan toiminnan tulisi kestää ja kuinka monella siirrolla saatu tulos pitäisi saada. Tätä vaatimusmäärittelyä verrataan testauksissa saatuihin tuloksiin. (Holappa 2011.)

Systeemitestaukseen kuuluu **toipumistestaus**, jota tarvitaan, kun tietokonesovellusten tulee selviytyä virhetilanteista ja jatkaa suoritusta ennalta määritellyn ajan kuluessa. Joissain tapauksissa sovelluksen tulee olla virhesietoinen, joka tarkoittaa, että mahdollisten virheiden syntyminen ei saa vaikuttaa koko sovelluksen toimintaan, vaan vian on pysyttävä paikallisena häiriönä. Tietyissä sovelluksissa toipuminen virhetilanteista ei ole automaattista, jolloin vaatimuksena tulee olla mahdollisuus ja kyky korjata virhetilanteet nopeasti. Tällöin taloudellista vahinkoa syntyisi mahdollisimman vähän. Toipumistestauksessa järjestetään tilanteita, joissa sovellus tulisi saada kaatumaan usein ja monissa erilaisissa tilanteissa. Kun sovellus on saatu kaadettua, tarkkaillaan sen toipumiskykyä. Tässä siis testataan järjestelmän toipumista normaalitilaan poikkeustilanteista. Tällaisia poikkeustilanteita voi olla esimerkiksi virtakatkos, muistihäiriö, levyvika, linjavika, aineistovirhe tai

prosessin keskeytyminen. Jos sovellus pystyy itse toipumaan automaattisesti, tarkkaillaan tilanteita alustuksesta, tietojen takaisin saannista ja uudelleenkäynnistyksestä. Jos käyttäjän on itse palautettava sovellus kunnolliseen toimivaan tilaan, on arvioitava tehtävään kuluvaa aikaa, jonka tulee olla hyväksyttävissä rajoissa. (Sommerville 2004, 540–547; Pressman 2001, 497.)

Systeemitestauksen yhtenä osana on **kokoonpanotestaus**, jolla testataan kehitetyn sovelluksen kykyä kommunikoida muiden tietokoneen sisäisten ja ulkoisten laitteiden kanssa. Tarvitut testit voivat keskittyä muutamaan asiakkaan määrittelemään kokoonpanoon. Pahimmassa tapauksessa lopullista toimintaympäristöä ei voida määritellä, vaan sovelluksen olisi periaatteessa toimittava kaikissa mahdollisissa kokoonpanoissa. Kokoonpanotestausta voidaan hyödyntää esimerkiksi WWW-sovelluksen testauksessa, jolloin testaus tulee suorittaa tarvittavilla eri selaimilla ja niiden versioilla eri käyttöjärjestelmissä. Mikäli sovellusta tullaan käyttämään erilaisissa laitteistokokoonpanoissa, tulisi sovellusta testata kaikissa kokoonpanoissa. Testausta suunniteltaessa on mietittävä, mitä ulkoisia laitteita sovelluksen kanssa tullaan käyttämään, mitä malleja ja versioita laitteista on saatavilla sekä mitä sovelluksessa hyödynnettäviä ominaisuuksia laitteistot tukevat. Tätä vaikeuttaa tilanne, jossa mahdollisia kokoonpanoja voi olla tuhansittain, joten kaikkia yhdistelmiä ei voida testata. Ensinnäkin sovellus on testattava useammassa tietokoneessa, käyttäen erilaisia ulkoisia laitteita ja ajureita, muuttaen muistin määrää ja kokeillen eri rajapintoja. (Sommerville 2004, 540–547; Pressman 2001, 496–497.)

Regressiotestauksella tarkoitetaan sitä testausta, joka suoritetaan, kun ohjelmistosta löydetyt virheet on korjattu tai toiminnallisuutta on lisätty tai vaihtoehtoisesti muutettu. Tällä testauksella pyritään varmistamaan sovelluksen toiminta korjausten jälkeenkin alkuperäisten vaatimusten mukaisesti. Testausta tarvitaan, jotta voidaan taata, ettei uusia virheitä ole ilmennyt aiemmin moitteettomasti toimiviin toimintoihin. Regressiotestausta suoritetaan sekä ohjelmiston kehitys- että ylläpitovaiheessa. Testauksessa käytetään

testitapauksista osajoukkoa, joka kattaa sovelluksen päätoiminnallisuudet. Testauksen lopuksi kuuluu tehdä vielä täydellinen, kaikki testitapaukset kattava uudelleentestaus. (Marttinen 2010, 11.)

Suorituskykytestaus suoritetaan usein systeemitestauksen osana, jolloin koko integroitu järjestelmä on käytössä. Joiltakin osin testausta voidaan suorittaa jo testauksen aiemmissa vaiheissa, lasilaatikkotestauksen aikana. Luotettavuustestaus tarkistaa järjestelmälle asetettujen luotettavuusvaatimuksien pitävyyttä. Suorituskykytestauksella pyritään selvittämään sovelluksen toiminta ympäristössään, kun taas luotettavuustestauksella pyritään tarkistamaan sovelluksen toiminta ympäristössään pitkällä aikavälillä. Luotettavuustestaus suoritetaan yleensä kustannusten kannalta tehokkaasti lyhyessä ajassa oloissa, jotka ovat tavanomaista käyttöä ankarammat. Sovellukseen syötetään runsas määrä tavanomaisia syötteitä ja luotettavuus lasketaan sen perusteella, kuinka monta väärää vastetta saadaan. Näiden syötteiden on vastattava todennäköisiä syötteitä, jotta saadaan luotettavampi arvio. (Sommerville 2004, 540–547; Pressman 2001, 496–497.)

Eräs välttämätön ominaisuus sovelluksilla on joustavuus. Tämä on oleellista, jotta sovellusta tai sen osia voidaan helposti käyttää uudelleen muualla. Toinen joustavuudella tavoiteltu piirre on se, ettei sovellus vanhene jos sen käyttöympäristö muuttuu, vaan sitä voidaan pienten muutosten jälkeen käyttää uudelleen. Joskus testaus suoritetaan vähäisillä tiedoilla sekä muutamilla samanaikaisilla käyttäjillä, vaikka ohjelman on toimittava oikeasti tuhannen käyttäjän sovelluksena. Joustavuustestauksen tarkoituksena onkin pyrkiä luomaan oikeat olot ja testaamaan sovellusta niissä. (Sommerville 2004, 540–547; Pressman 2001, 496–497.)

Hyväksymistestauksessa sovellusta testataan yhdessä asiakkaan kanssa. Testausmenetelmä on jaettu kahteen eri vaiheeseen ennen lopullista sovelluksen hyväksyntää. Hyväksymistestauksen kaksi eri vaihetta on alfa- ja betatestaus. Alfa-testaus suoritetaan siten, että todelliset käyttäjät testaavat

sovellusta sovelluksen kehittäneessä ympäristössä. Beta-testauksen suorittavat loppukäyttäjät todellisessa ympäristössä. Näiden testausten jälkeen, sovellus on valmis julkaistavaksi. (Sommerville 2004, 540–547; Pressman 2001, 496–497.)

2.2 Testausmenetelmät

Sovelluksen testausmenetelmät ovat turvallisuustestaus, puutetestaus, dynaaminen ja staattinen lasilaatikkotestaus ja mustalaatikkotestaus. Testausmenetelmän valitsee yleensä työryhmä. Valintaan vaikuttaa, millaisesta lopputuloksesta on tarve. (Tersa 2002, 55.)

2.2.1 Turvallisuustestaus

Tietoturva on merkittävä ominaisuus monissa järjestelmissä, koska niissä liikkuu luottamuksellista tietoa. Turvallisuustestauksella pyritään kattamaan koko sovellus suunnittelusta ja matalan tason tehtävistä aina tärkeimpiin toimintoihin asti. Tähän testaukseen ei varsinaisesti tarvita uusia testitapauksia, vaan muun testauksen aikana tutkitaan, ettei tietoturvasta tingitä. Tietoturvatestien aikana testaaja jäljittelee ihmistä, joka haluaa tunkeutua sovellukseen. Testaaja pyrkii hankkimaan tietoonsa salasanoja, rikkomaan sovelluksen tietosuojan tarkoitukseen sopivilla työkaluilla tai varaamaan systeemin kaiken kapasiteetin estäen käytön muilta. Hän voi myös tahallisesti aiheuttaa systeemiin virheitä, koska toipumisaikana sovellus saattaa olla helpoiten haavoitettavissa. Sovelluksen kehittäjien tehtävänä on tuottaa niin hyvä sovellus, että siihen tunkeutumiseen tarvitut resurssit ovat suuremmat kuin tunkeutumalla sovelluksesta saatava hyödynnettävä tieto. (Pressman 2001, 497–498.)

Kaikissa tietojärjestelmissä käyttäjät tunnistetaan käyttäjätunnuksen ja salasanan perusteella. Huonon salasanan voi kuka tahansa vähällä vaivalla arvata ja halutessaan päästä sen avulla käsiksi tiedostoihin ja sähköpostiin. Vaikkei sivustolla olisi mitään salattavaa tai henkilökohtaista, heikentää helposti arvattava salasana koko järjestelmän turvallisuutta. Salasanan ei ole tarvitse

olla joukko satunnaisia kirjaimia, sillä tällöin on aivan liian suuri kiusaus kirjoittaa salasana muistiin paperille. Salasanan tulee olla juuri niin vaikea, että sen muistaa itse, mutta se ei ole toisten arvattavissa. (Pressman 2001, 497–498.)

Hyvän salasanan vaatimuksia ovat seuraavat:

- vähintään 8 merkkiä pitkä, ja sen tulee sisältää sekä kirjaimia että numeroita

- ei missään kielessä esiintyvä sana, eli ei löydy sanakirjoista

- ei nimi (puolison, kumppanin, lemmikin, koneen tai ihmisen)

- ei liity omaan työhön tai harrastuksiin

- ei ole helposti arvattavissa omasta toiminnasta tai olemuksesta.

Ei suositella käytettäväksi skandinaavisia ja muita erikoiskirjaimia, kuten å, ä tai ö. Suositellaan välttämään myös välimerkkien ja muiden erikoismerkkien käyttöä. Osa laitteistoista hyväksyy nämä merkit, mutta monet järjestelmät eivät kelpuuta niitä. (Helpdesk – tietotekniikan neuvonta ja käyttötuki 2007.)

2.2.2 Puutetestaus

Puutetestauksen tarkoituksena on paljastaa virheitä ennen sovelluksen julkaisemista. Puutetestaus on onnistunut, kun se saa systeemin toimimaan väärin ja tuottamaan virheellisiä vasteita. Puutetestauksen vastakohta on validointitestaaminen, jonka tarkoituksena on osoittaa järjestelmän toimivan oikein ja määrittelyjensä mukaisesti. Validointitestaukset suoritetaan yleensä hyväksymistestitapausten avulla. Ensin on hyväksymistestejä, joiden avulla pyritään osoittamaan sovelluksen toimivuus. Toiseksi on hylkäämistestejä, joiden tarkoituksena on pyrkiä osoittamaan sovelluksen virheet. Testaukset kannattaa aloittaa helpommilla testeillä, jolloin pyritään osoittamaan sovelluksen toimivuus tavanomaisimmissa tapauksissa. Tällaisilla hyväksymistesteillä sovelluksesta saatetaan havaita runsaasti virheitä. Kun sovelluksen tavanomainen toiminta on saatu varmistettua, voidaan siirtyä testaukseen, jonka tarkoituksena on pyrkiä kaatamaan sovellus. Tätä voidaan kutsua myös

virheiden pakotukseksi. Tässä testaus kohdistetaan sovelluksen heikkouksiin ja testataan sovellusta sen äärirajoilla, suurella määrällä tietoa, yhtäaikaisilla tapahtumilla ja etsien mahdollisuuksia käyttää sovellusta jopa määrittelyjen vastaisesti. (Pressman 2001, 478–483.)

2.2.3 Staattinen lasilaatikkotestaus

Lasilaatikkotestauksessa testaajalla on mahdollisuus tarkastella koodia ja käyttää sitä hyödykseen saadakseen vihjeitä testaukseensa sovelluksen heikoista paikoista. Lasilaatikkotestauksessa testaaja siis tavallaan näkee koodia sisältävän laatikon sisään, joten hän voi tarkastella koodin rakennetta ja toimintaa. Tämän avulla hän voi päätellä, millä arvoilla sekä missä kohdissa testaus kannattaa suorittaa. Testausta ei saa kuitenkaan suorittaa koodin ehdoilla eli testaaja ei saa sovittaa testitapauksia koodin mukaisiksi. Erityisen hyödyllistä lasilaatikkotestauksessa on sen mahdollisuus löytää virheitä aikaisin. (Pressman 2001, 444–445.)

2.2.4 Dynaaminen lasilaatikkotestaus

Lasilaatikkotestausta kutsutaan rakenteelliseksi testaukseksi, koska testaus perustuu järjestelmän sisäisen rakenteen tutkimiseen. Koska testattava ohjelma tunnetaan kooditasolla, voidaan arvioida mahdollisia virhealttiita paikkoja ja keskittää testaus näihin. Dynaamisella lasilaatikkotestauksella voidaan ohjelman sisäisen rakenteen avulla rakentaa testitapauksia, joita suoritetaan ja joiden tuloksia analysoidaan suhteessa odotettuihin tuloksiin. Lasilaatikkotestaus on välttämätöntä, koska koodin ominaisuuksien takia kaikkia virheitä ei mustalaatikkotestauksella havaita. Dynaamisella lasilaatikkotestauksella voidaan taata kaikkien toimintapolkujen kokeilu vähintään kerran. Lisäksi testataan loogisten päätöslauseiden suorittamista arvoilla tosi sekä epätosi. Ihanteellisessa tapauksessa lasilaatikkotestauksella voitaisiin taata myös silmukoiden toiminta raja-arvoilla ja arvoalueen sisälle mahtuvilla syötteillä. Tällä testauksella kokeillaan sisäiset tietorakenteet niiden oikeellisuuden varmistamiseksi. Testausta vaikeuttaa virheen monimutkaisuus, sillä virheet muodostuvat usein monen tekijän vaikutuksesta. Testaukseen

annettavan syötteen X lisäksi testaustulokseen vaikuttaa sovelluksen sisäinen tila Z. Näiden perusteella saadaan vaste Y, jonka oikeellisuutta mustalaatikkotestauksessa arvioidaan. Lasilaatikkotestauksessa päästään tarkastelemaan myös sisäisten tilojen muuttumista. Kaikkia mahdollisia tiloja ei voida testata, joten on mietittävä, milloin testaus on riittävää. Dynaamista lasilaatikkotestausta voidaan suorittaa lähes samoin kuin tavanomaista virheiden etsintää ja korjausta. Dynaamisella lasilaatikkotestauksella pyritään havaitsemaan virheitä, kun taas virhetestauksella avulla pyritään paikallistamaan sekä korjaamaan jo havaittuja virheitä. (Pressman 2001, 445–459.)

Testauksen laatua arvioidaan tavallisesti sillä, miten täydellisesti ohjelma on testattu. Kattavuusmitoilla voidaan mitata, kuinka perusteellisesti lasilaatikkotestit kattavat ohjelman toiminnot. Testaamisen kattavuuden arviointiin sopivia mittausmenetelmiä ovat esimerkiksi lause-, polku- ja erilaiset ehtokattavuudet. Käytännössä sadan prosentin kattavuutta ei saavuteta edes pieniä ohjelmia testattaessa. Vaikka lasilaatikkotestausta käytetään vain testauksen alemmilla tasoilla, ei kaikkea voida testata suoraviivaisesti. Testikattavuus on helpointa saada korkeaksi yksikkötasolla. Tällöinkään ei voida taata sadan prosentin kattavuutta, sillä esimerkiksi lausekattavuudessa todellinen kattavuus nousee harvoin yli 80 prosenttiin. (Pressman 2001, 445–459.)

Testikattavuus ei takaa sovelluksen toimivuutta, mutta sitä pidetään kuitenkin tärkeänä. Se kuitenkin tarjoaa suuntaa antavan ja johdonmukaisen arvon siitä, miten järjestelmällistä ja johdonmukaista sovelluksen testaus on ollut. Jos koodista puuttuu esimerkiksi ehtolauseissa jokin oleellinen ehtoarvo, mikään kattavuusmitoista ei yleensä huomaa arvon puuttumista, vaan tällaiset asiat ovat yksin testaajan vastuulla. Tällöin osan koodista puuttuessa ei tietenkään kattavuusmitalla enää saavuteta oikeaa tulosta. Mitään kattavuusmitoista ei tulisi käyttää ainoana takuuna testauksen ja ohjelmiston laadusta. Kattavuustyökalujen voidaan ajatella olevan hyödyllisiä, kun niitä käytetään lisäämään ajattelua, ei korvaamaan sitä. (Pressman 2001, 445–459.)

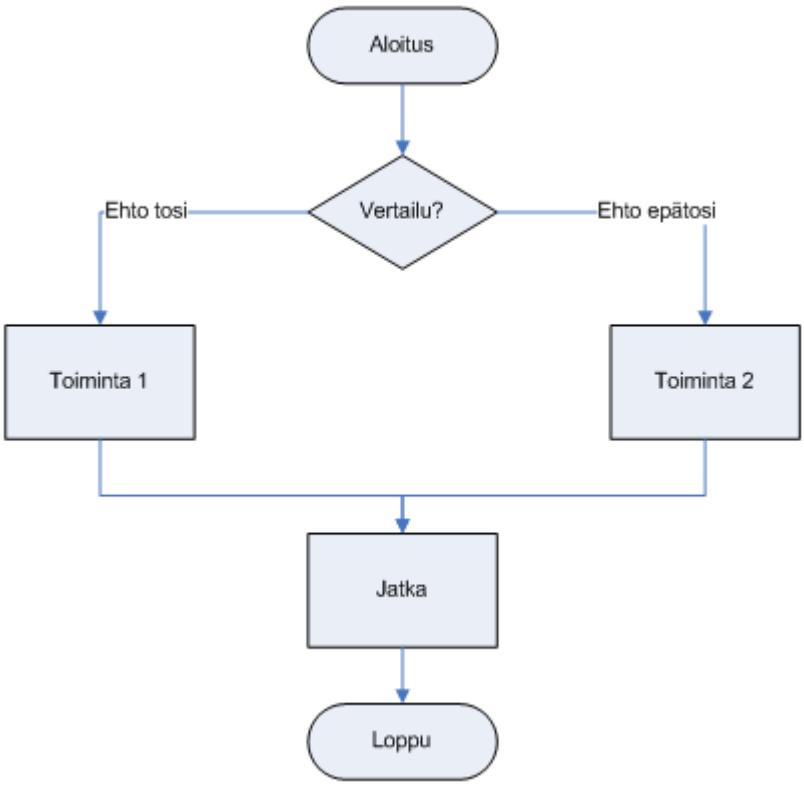
Dynaamisen testauksen menetelmät voidaan jakaa kolmeen ryhmään: kontrollin kulku, silmukkatestaus ja tietovirtatestaus. Ensimmäinen dynaamisen lasilaatikkotestauksen tekniikoista on kontrollin kulku. Tässä sovellus pakotetaan kulkemaan erilaisia polkuja pitkin sitä suoritettaessa. Eri kontrollin kulun menetelmissä kuljettavat polut valitaan kullekin menetelmälle ominaisten piirteiden mukaisesti. Kontrollipolkujen läpikäyntiin keskittyvät tekniikat ovat lause-, polku- ja erilaiset ehtokattavuudet. Näillä on olemassa keskinäinen järjestys siitä, mikä on vahvempi menetelmä kuin toinen. Yleisin kattavuusmitoista on lausekattavuus. Sataprosenttinen kattavuus toteutuu, kun jokainen ohjelman lause on suoritettu ainakin kerran. Sadan prosentin lausekattavuus voi kuulostaa yksinkertaiselta vaatimukselta. Todellisessa projektissa sitä on vaikea saavuttaa, sillä lauseiden määrä on valtava. Tarkkailemalla lauseiden käyttöä voidaan löytää lauseet, joita ei koskaan suoriteta. Tämä testaus kuitenkin takaa vain sen, että jokainen lause on suoritettu kerran, mutta se ei takaa, onko kaikki ohjelman polut käyty läpi. (Pressman 2001, 445–459.)

Ehtokattavuudessa päätöksen kaikkien ehtojen on saatava kaikki arvonsa. Tämän lisäksi ohjelman jokaisen päätöksen kaikkien osaehtojen on saatava kaikki arvonsa testiajojen aikana. (Laine 2001.) Ehtokattavuudessa vaaditaan kaikkien ehdossa olevien muuttujien arvojen läpikäynti. Ehtokattavuuksien luokitteluun kuuluu myös moniehtokattavuus. Moniehtokattavuudessa testaus suoritetaan kaikkien ehtojen kaikilla yhdistelmillä. Paremmuusjärjestys kattavuusmitoille määritellään sen mukaan, mikä mitoista on vahvin, eli minkä voimassa ollessa myös muut ovat voimassa. Jos tietyllä mitalla tehty testi takaa myös toisella tehdyn testin, on ensimmäinen vahvempi. (Pressman 2001, 445–459.)

Yksi kontrollin kulun testauksen tekniikoista on polkutestaus. Tämä rakenteellinen testausmenetelmä on yksi vanhimmista menetelmistä ja sitä käytetään lasilaatikkotestauksen lisäksi myös mustalaatikkotestauksessa. Ensin valitaan testattava polku, määritellään syötteet ja oikeat vasteet. Ohjelma suoritetaan manuaalisesti ja tuloksia verrataan alussa määritettyihin tuloksiin,

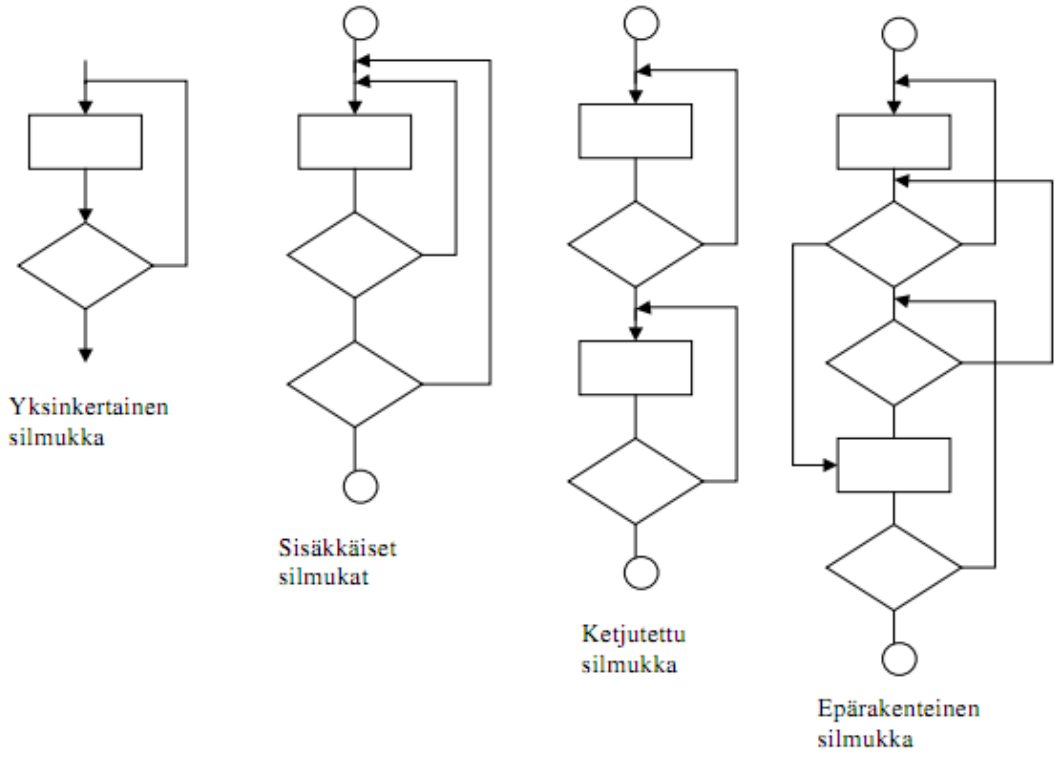
minkä jälkeen mahdolliset virheet raportoidaan. Kun komponentit integroidaan kokonaisuudeksi, rakenteellisten testausmenetelmien käyttö ei enää ole mahdollista. Kokonaisuuden myötä polkujen määrä kasvaa valtavaksi ja integroinnin jälkeen testauksen luonne ja tavoitteet muuttuvat. Polkutestausta käytetäänkin yleensä vain yksikkötestauksessa. Polkutestauksessa tarkoituksena on käydä läpi jokainen komponentin tai ohjelman polku. Tämä takaa, että jokainen ohjelman lause tulee suoritettua vähintään kerran. Ohjelman ehtolauseet testataan arvoilla tosi sekä epätosi. (Pressman 2001, 445–459.)

Polkujen valinta voidaan tehdä eri tavoin. Optimaalista olisi käydä läpi kaikki eri vaihtoehdot, mutta se on harvoin mahdollista, sillä esimerkiksi silmukkarakenteet aiheuttavat polkujen määrän suuren kasvun. Polkutestauksen ensimmäinen vaihe on vuokaavion määrittäminen. Vuokaaviossa ohjelma esitetään yksinkertaistettuna rakenteena niin, että päätöskohtia esittävät solmut ja ohjelman kulkua solmujen väliset linkit eli reunat. Ehtolauseet tosi ja epätosi sekä silmukat, aiheuttavat kaavioon uuden polun syntymisen. Kuvassa 3 on nähtävillä vuokaavio, jonka avulla voidaan tarkistaa, että kaikki polut huomioidaan ja ne tulevat testatuiksi vähintään yhden kerran. Täten jokainen lause ja ehtolauseet molemmilla arvoillaan tulevat testatuiksi. (Pressman 2001, 445–459.)



KUVA 3. Vuokaavio (Holvikivi 2010)

Toinen dynaaminen lasilaatikkomenetelmä on silmukkatestaus, joka on näkyvissä kuvassa 4. Tässä silmukoiden testaaminen on haastavaa, koska kaikkien mahdollisuuksien läpikäynti on usein mahdotonta. Luokkia ovat yksinkertaiset silmukat, sisäkkäiset silmukat, ketjutetut silmukat ja epärakenteiset silmukat. (Pressman 2001, 445–459.)



KUVA 4. Silmukoiden eri luokat (Pressman 2001,459)

Ohjelman tila sekä oikeellisuus testataan kolmessa eri vaiheessa: silmukkaan tultaessa, silmukan suorituksen aikana ja siitä poistuttaessa. Vaikeimpia ovat silmukat, jotka ovat osittain sisäkkäisiä. Epärakenteisten silmukoiden testaaminen on todella vaikeaa. Epärakenteisiin silmukoihin tulee yleensä helpommin virheitä koodausvaiheessa ja niitä varten on vaikeaa suunnitella testejä. Mahdollisuuksien salliessa, tällaiset silmukat tulisi ennen testausta kirjoittaa uudelleen helpommiksi silmukkarakenteiksi testauksen mahdollistamiseksi. (Pressman 2001, 445–459.)

Eräs rakenteisen testauksen alue on tietovirtatestaus, jossa tarkastellaan tiedon kulkua sovelluksessa. Yksikkötasolla tämä tarkoittaa tiedon kuljettamista yksittäisen moduulin läpi. Systeemitestaustasolla se tarkoittaisi tiedon kuljettamista koko ohjelman läpi, mikä olisi varsin aikaa vievää. Lasilaatikkotestauksessa tiedon kulkua voidaan tarkkailla syötteistä ja vasteista sekä sovelluksen sisäisten muuttujien arvojen muutoksista virhetestauksen sovellusta ajettaessa. Tietovirtatestauksessa tarkastellaan tietorakenteiden

tilamuutoksia ohjelman suorituksen aikana. Tilamuutoksina huomioidaan arvon asettaminen ja arvon hyväksikäyttö. Tiedon testauksessa voidaan suorittaa erityinen virhetestaus. Sovelluksen arvot ovat näkyvissä koko testauksen ajan, joten sovellukselle voidaan syöttää arvoja, joilla saadaan tuotettua vuorotellen kukin virhetilanne. Tällaista erityistä virheiden tuottamista suoritettaessa on huomioitava, ettei yritä pakottaa sovellusta toimimaan arvoilla, joita se ei oikeasti ikinä voi saada. Tämä on kuitenkin hyvä tapa testata kaikkien virheviestien kunnollisuus. (Pressman 2001, 445–459.)

2.2.5 Mustalaatikkotestaus

Mustalaatikkotestauksessa ei testata sovelluksen sisäistä toimintaa, vaan ainoastaan sen syötteet ja vasteet. Ohjelmalle annetaan syöte käsiteltäväksi ja verrataan saatua vastetta määrittelyjen mukaiseen oikeaan tulokseen. Tässä testauksessa päästään eroon lasilaatikkotestauksen haittapuolesta eli lähdekoodin tarpeesta testaamista varten. Testaustapaukset jaetaan dynaamiseen mustalaatikkomenetelmään ja staattiseen mustalaatikkomenetelmään. (Pressman 2001, 459–468.)

Mustalaatikkotestauksen perusajatus määritellään yksinkertaisesti. Ensin sovellukselle annetaan tietty syöte *x*, ja saadaan sovelluksen vaste *f(x).* Vastetta *f(x)* verrataan tunnettuun oikeaan vasteeseen *y*. Jos *f(x) = y,* testi on läpäisty, muuten on havaittu virhe. Puutetestauksen onnistuessa testi tuottaa virheellisiä vasteita. Suunnitteluvaiheessa olisi pyrittävä kokoamaan testitapaukset, jotka tuottavat määrittelyjen vastaisia vasteita. Vaikeutena on keksiä testiaineisto, joka suurella todennäköisyydellä tuottaisi virheitä. Mustalaatikkotestauksen pyrkimyksiä on löytää käyttöliittymävirheet, virheet tietorakenteissa tai pääsyssä ulkoisiin tietorakenteisiin, toimintavirheet sekä käynnistys- ja lopetusvirheet. Näissä menetelmissä siis yleisesti keskitytään sovellukselle asetettuihin toiminnallisuusvaatimuksiin ja tämän testauksen tulee täydentää lasilaatikkotestausta. (Pressman 2001, 459–468.)

Staattinen mustalaatikkotestaus kattaa vain kirjoitetun tuotteen määrittelyn testaamisen, kun taas dynaaminen kattaa kaikkien muiden kehitettyjen osa-

alueiden testauksen. Testaamalla määrittelydokumentaatio vältetään monien virheiden välittyminen virheellisestä dokumentaatiosta koodausvaiheessa tuotteeseen. Koska ainoastaan määrittelydokumentin on oltava valmiina, voidaan staattinen mustalaatikkotestaus suorittaa jo projektin varhaisessa vaiheessa. Aluksi tuotteen määrittely testataan korkealta tasolta pyrkien hahmottamaan kokonaiskuva sekä havaitsemaan yleisiä ongelmia ja puutteita. (Pressman 2001, 459–468.)

Mustalaatikkotestauksessa ensimmäinen vaihe on sovelluksen toiminnan testaus. Tämän tarkoituksena on löytää virheitä tai puutteita ohjelman määrittelyn ja sen todellisen toiminnan välillä. Määrittelyjä analysoimalla on saatava tietoon sallitut arvot kaikkiin sovelluksen syötekenttiin. Tavallisesti testaus suoritetaan myös määrittelyalueen ulkopuolisilla syötteillä. Jos sovelluksen toiminnasta ei ole tietoa, tulisi testata kaikilla mahdollisilla syötteillä, jotta voitaisiin olla varmoja toiminnan oikeellisuudesta. Kaikkien mahdollisten syötekombinaatioiden testaaminen on kuitenkin käytännössä mahdotonta. (Pressman 2001, 459–468.)

Kattavan testiaineiston valinta on vaikea ongelma. Tämä toteutetaan usein testaajan omien mielipiteiden ja kokemusten pohjalta eikä systemaattista menetelmää käyttäen. Tämä korostuu erityisesti pienissä projekteissa, joissa ohjelmoijat itse testaavat ja valitsevat testiaineiston. Tällöin he huomaamattaan voivat valita koodiin sopivia aineistoja sekä lähes välttelevät valitsemasta vaikeita tai harvoin esiintyviä ongelmia tuottavia syötteitä testitapauksiinsa jolloin testaus kattaa vain osan sovelluksesta. Sovelluksen perinpohjaisessa testaamisessa kaikkia mahdollisia syötemuunnelmia ei voida kokeilla, mikä asettaa testiaineiston valinnalle suuria vaatimuksia. Tiedon testauksessa on seurattava, että käyttäjän antama tieto, sovelluksen tuottamat tulokset ja väliaikaiset sovelluksen sisäiset tiedot käsitellään oikein. Kaikkiaan tietoa on paljon, eikä täydellistä yksityiskohtaista testausta voida suorittaa. (Pressman 2001, 459–468.)

Käytännössä testattavia syötteitä ovat nollat ja tyhjät arvot. Esimerkiksi tekstikenttien toimintaa on hyvä testata jättämällä kenttä tyhjäksi. Joissain tapauksissa kenttä saattaa hyväksyä tyhjän arvon, mutta sovellus ei osaakaan käsitellä tietoa, jolloin päädytään virhetilanteeseen. Kun kentät on määrätty pakollisiksi, tulisi käyttäjän saada kunnollinen huomautus arvon puuttumisesta. (Marttinen 2010, 13.)

2.2.6 Valkolaatikkotestaus

Valkolaatikkotestauksessa testaajalla on tiedossaan järjestelmän sisäinen rakenne kokonaisuudessaan. Testattavasta kohteesta vastaava taho on tietoinen testauksesta ja sen yksityiskohdista. Testi testaa kohteesta vastaavan tahon kykyä hallita ja puolustaa kohdetta, mutta ei kykyä toimia ennalta arvaamattomissa poikkeustilanteissa. Testin etu on, että testaajalla on kokonainen näkemys testattavasta kohteesta yhdistettynä kohteesta vastaavan tahon vasteeseen. Testin laajuus ja syvyys riippuvat testaajalle annetun tiedon tarkkuudesta ja testaajan tietotaidoista. Testi voidaan suorittaa myös jättämällä kertomatta kohteesta vastaavalle taholle testauksesta. Näin voidaan testata sisältä päin tulevaa tunkeutumista. (Koskinen – Kelo – Mähönen – Oikarinen – Salo – Vartiainen 2008, 4.)

2.2.7 Harmaalaatikkotestaus

Harmaalaatikkotestauksessa testaajalla on rajoitetusti tietoa järjestelmän sisäisestä rakenteesta. Kohteesta vastaava taho on tietoinen suoritettavasta testauksesta, mutta vain osittain sen yksityiskohdista. Testi testaa tekijän tietotaidon lisäksi myös kohteesta vastaavan tahon valmiutta reagoida poikkeustilanteisiin. Testiä pidetään ajankäytöltään tehokkaana niin testauksen suorittamisen kuin vastesuunnitelman harjoittelun kannalta. Testin laajuus ja syvyys riippuvat testaajalle etukäteen annetun tiedon määrästä ja tietotaidosta. Testityyppi simuloi parhaiten todellista hyökkääjää, koska hänellä useimmiten on enemmän tietoa kuin mustalaatikkotestauksessa oletetaan. (Koskinen ym. 2008, 4.)

2.3 Vakavuusluokitus

Jokainen arvioinnissa löydetty ongelma tulisi luokitella asteikolla, joka kertoo asiantuntijan mielipiteen käytettävyysongelman vakavuudesta. Ongelman vakavuuden luokituksen tulisi nojata ainakin seuraavaan neljään seikkaan:

- Esiintymistiheys eli kuinka usein potentiaaliseen ongelmatilanteeseen törmää?

- Vaikutukset eli onko ongelmatilanteesta helppo vai vaikea selvitä?

- Toistuvuus eli onko ongelma helposti ohitettavissa, kun sen on kerran tunnistanut, vai vaivaako se jatkuvasti?

- Markkinavaikutukset eli tekeekö virhe tuotteesta markkinoilla merkittävästi huonomman tai jopa käyttökelvottoman?

Perinteisesti käytetään Nielsenin käyttöliittymän suunnitteluvirheluokkia, jotka jakautuvat seuraavasti:

1. Ei käytettävyysongelmaa
2. Kosmeettinen ongelma, ei tarvitse korjata jos ei ole ylimääräistä aikaa
3. Pieni ongelma, korjauksella alhainen prioriteetti
4. Suuri ongelma, korjaamisella suuri prioriteetti
5. Katastrofaalinen ongelma, pitää korjata ennen kuin järjestelmä otetaan käyttöön.

Suunnitteluvirheluokkien perusteella sovelluksen tekijä voi aloittaa korjaamistoimenpiteet vaatimustason mukaisesti. Suunnitteluluokat voi jaotella myös kustannus- sekä ajankulun perustein. (Holappa 2011.)

3 BELLEBOOK-SOVELLUKSEN ARKKITEHTUURI

BelleBook-sovelluksen omistaja on vuokrannut webhotellista kiintolevytilaa, johon sovellus on tallennettu. Webhotellista saa verkkotunnuksen, esimerkiksi www.yritys.fi/bellebook, jonka avulla sovelluksen sivulle on helppo löytää. Webhotelli koostuu palvelintietokoneesta ja Internet-verkkoyhteydestä. Webhotelli voi tarjota erilaisia palveluita: sähköposti-, keskustelu-, hallinta- tai verkkokauppapalveluita. Kun tehdään sopimus webhotellin palvelun tarjoan kanssa, saadaan käyttöön FTP-tunnus, joka toimii tiedonsiirtovälineenä kahden tietokoneen välillä. Tämän lisäksi saadaan SSH-tunnukset, jotka on tarkoitettu salattuun yhteydenottoon palvelimen kanssa. Sovelluksen omistaja siirtää FTP:llä sovelluksen palvelimelle. Sovelluksen käyttäjä, esim. Loppukäyttäjä 2, kirjautuu sovellukseen www-osoitteella ja pääsee siten käyttämään sovelluksen tarjoamia palveluita. (Wikipedia 2012, hakusana webhotelli.)

Varausjärjestelmän käyttöliittymä on toteutettu HTML/CSS-tekniikalla. Siinä on käytetty JavaScriptiä niin sanotun Ajax-tekniikan tavoin. Tiedon varastoimiseen on käytetty MySQL-tietokantaa. Sieltä voi lukea ja sinne voi kirjoittaa PHP-ohjelmointikielen avulla. Mitään suuria vaatimuksia serveriltä ei ole, ainoastaan, että PHP-kääntäjä on asennettu ja MySQL-tietokanta käytössä. Selaimen pitää tukea JavaScriptiä, ja näin ollen vaikkei ajanvarausjärjestelmään ei ole kehitetty erillistä mobiiliversiota, nykyajan älypuhelimilla sovelluksen pitäisi toimia. (Jänkälä 2012.)

4 BELLEBOOK-SOVELLUKSEN TESTAUS

BelleBook-sovellusta testattiin mustalaatikkotestausmenetelmällä, joka on testaustasoltaan systeemitestausta. Käytännössä tämä tarkoittaa, ettei testaajalla ole käytössä koodia, vaan testataan sovelluksen toimivuutta syöttämällä kenttään arvoja ja tarkastelemalla tulosta.

4.1 Testausmenetelmät

Sovellusta testattiin perustoiminnoilla, erikoismerkeillä, virhelyönneillä sekä poikkeustiloilla. Perustoiminnoilla testaaminen tarkoittaa aakkoset a–ö. Erikoismerkeillä testaaminen tarkoittaa koodauksessa käytettäviä ASCII–erikoismerkkejä (esimerkiksi *, {, @, ;, :, /, \ ).Virhelyönneillä testaaminen tarkoittaa esimerkiksi napin pohjaan jäämistä. Poikkeustiloilla testaaminen tarkoittaa esim. internetyhteyden katkeamista tai koneen kaatumista.

Sovellusta testattiin yleisimmillä internetselaimilla:

- Google Chrome 17.0.963.79

- Windows Internet Explorer 9.0.8112.16421

- Mozilla Firefox 11.0

- Safari 5.1.4.

Tämän lisäksi sovellusta testattiin iPhone 4:llä sekä Samsung GT-S5600 -älypuhelimella, joka tukee JavaScriptiä.

Saatu tulos on merkitty nimikkeillä OK, ERROR 1 ja ERROR 2. OK-tulos tarkoittaa, että sovellus toimii odotetulla tavalla. ERROR 1 tarkoittaa, että sovelluksen ei pitäisi hyväksyä syötettä ja sovellus toimii odotetulla tavalla. ERROR 2 tarkoittaa, että sovelluksen ei pitäisi hyväksyä syötettä mutta sovellus hyväksyy sen. ERROR 2:n tapahtuessa virheelle arvioitiin vakavuusaste, joka on merkitty 1–5.

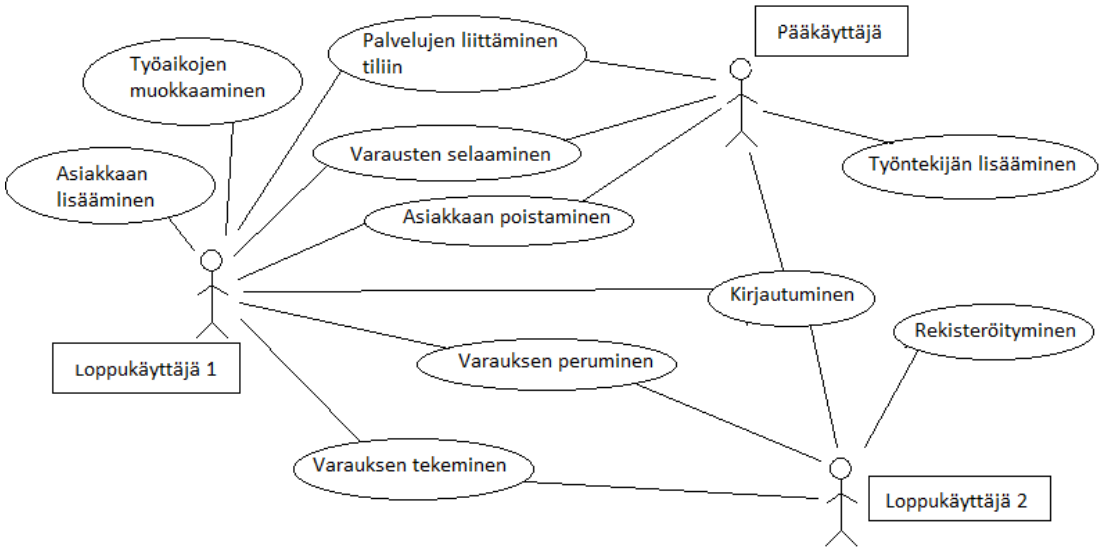
Sovellusta testattiin kaikkien käyttäjien käyttökulmasta:

- Pääkäyttäjä, joka omistaa admin-oikeudet ja voi lisätä työntekijöitä, toimipisteitä sekä toimenpiteitä. Näiden lisäksi pääkäyttäjä määrittää toimipisteen aukioloajan.

- Loppukäyttäjä 1, joka on työntekijä. Loppukäyttäjä 1 voi määrittää toimenpiteiden ajan sekä omat työtunnit.

- Loppukäyttäjä 2, joka on asiakas. Loppukäyttäjä 2 voi muokata omia tietoja, etsiä vapaan ajan, varata ajan ja kirjoittaa vaihtoehtoisesti lisätietoja-kenttään varatessaan aikaa.

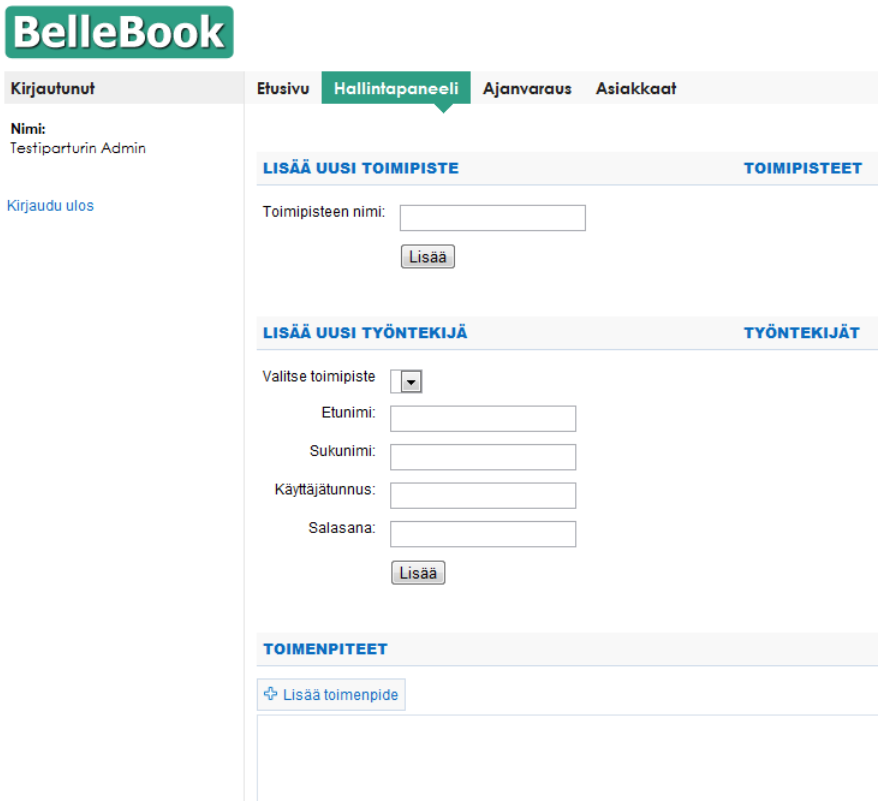
Kuvassa 5 on ajanvarausjärjestelmän käyttötapauskaavio, jonka mukaan sovelluksen tulisi toimia kullekin käyttäjälle.



KUVA 5. Käyttötapauskaavio

4.2 Testaaminen pääkäyttäjän käyttökulmasta

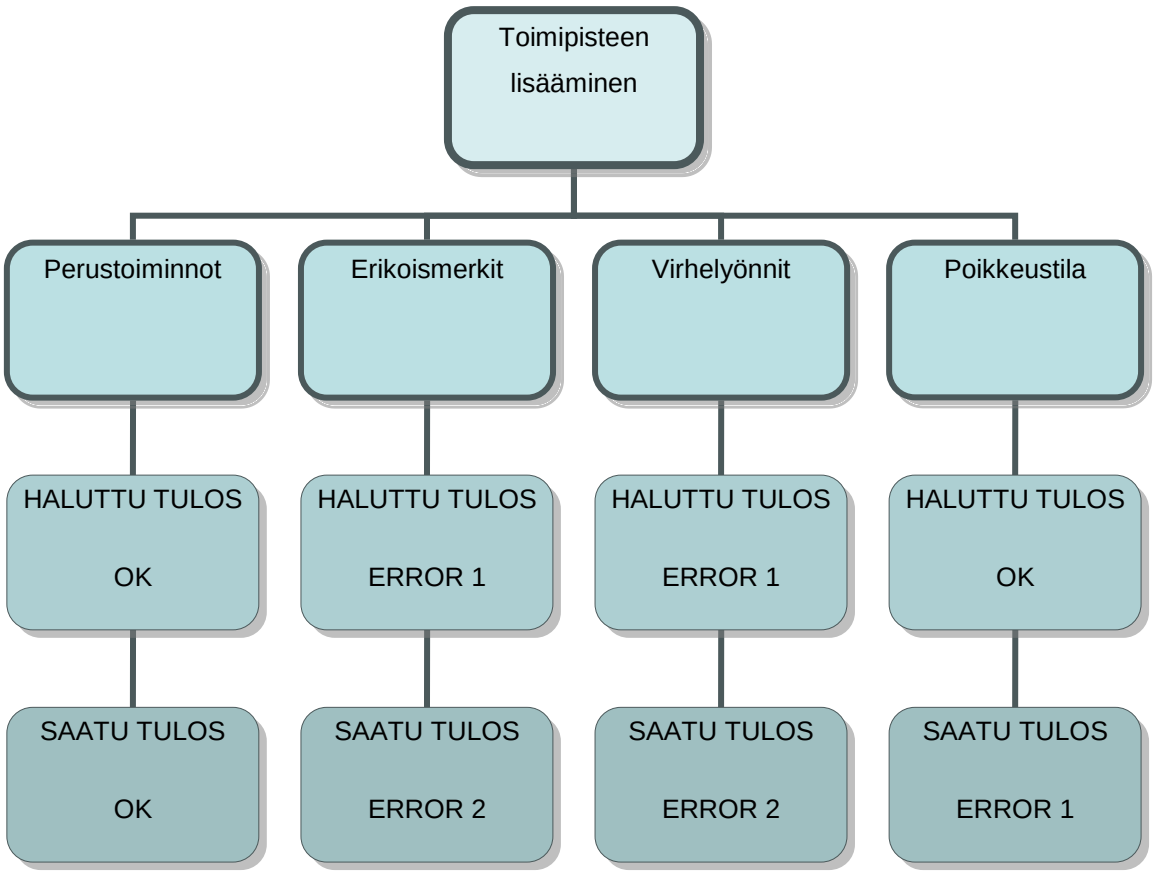
Sovellukseen on annettu pääkäyttäjä, testiparturin Admin, joka on sovelluksen tilaaja ja hänellä on admin-oikeudet. Testiparturin Admin voi lisätä toimipisteitä, työntekijöitä sekä toimenpiteitä sovellukseen. Kuvassa 6 on sovelluksen etusivu pääkäyttäjän ollessa sisäänkirjautuneena.



KUVA 6. Pääkäyttäjä sisäänkirjautuneena sovellukseen

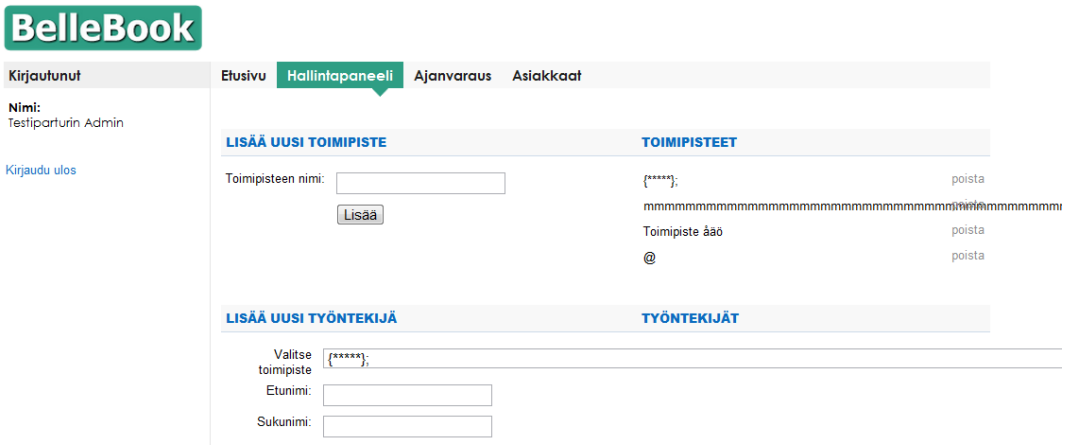
4.2.1 Toimipisteen lisääminen

Toimipisteen lisääminen on tehty sovelluksessa helpoksi. Pääkäyttäjän tulee kirjoittaa toimipisteen nimi ja sen jälkeen painaa nappia Lisää. Kuvassa 7 on nähtävillä toimipisteen lisäämisen toimintakaavio sekä odotetut ja saadut tulokset.



KUVA 7. Toimipisteen lisäämisen toimintakaavio

Sovellus tuntui hyväksyvän kaikki syötteet. Toivottavaa voisi olla, ettei sovellus hyväksyisi erikoismerkeillä alkavaa sanaa eikä virhelyöntejä. Toisaalta nämä eivät estäneet sovelluksen käytettävyyttä tai aiheuttaneet minkäänlaista virhetilaa, joten vakavuusaste molemmille on 3. Sovellus hyväksyy jopa 255 kirjaimen sanan, mikä tuntuu turhalta, ja hyväksyttyä kirjainmäärää voisi myös pienentää. Tämä on kuitenkin virheenä vakavuusasteeltaan 2. Kuvassa 8 on hallintapaneelin ulkoasu, kun siihen on syötetty erityylisiä toimipisteitä.

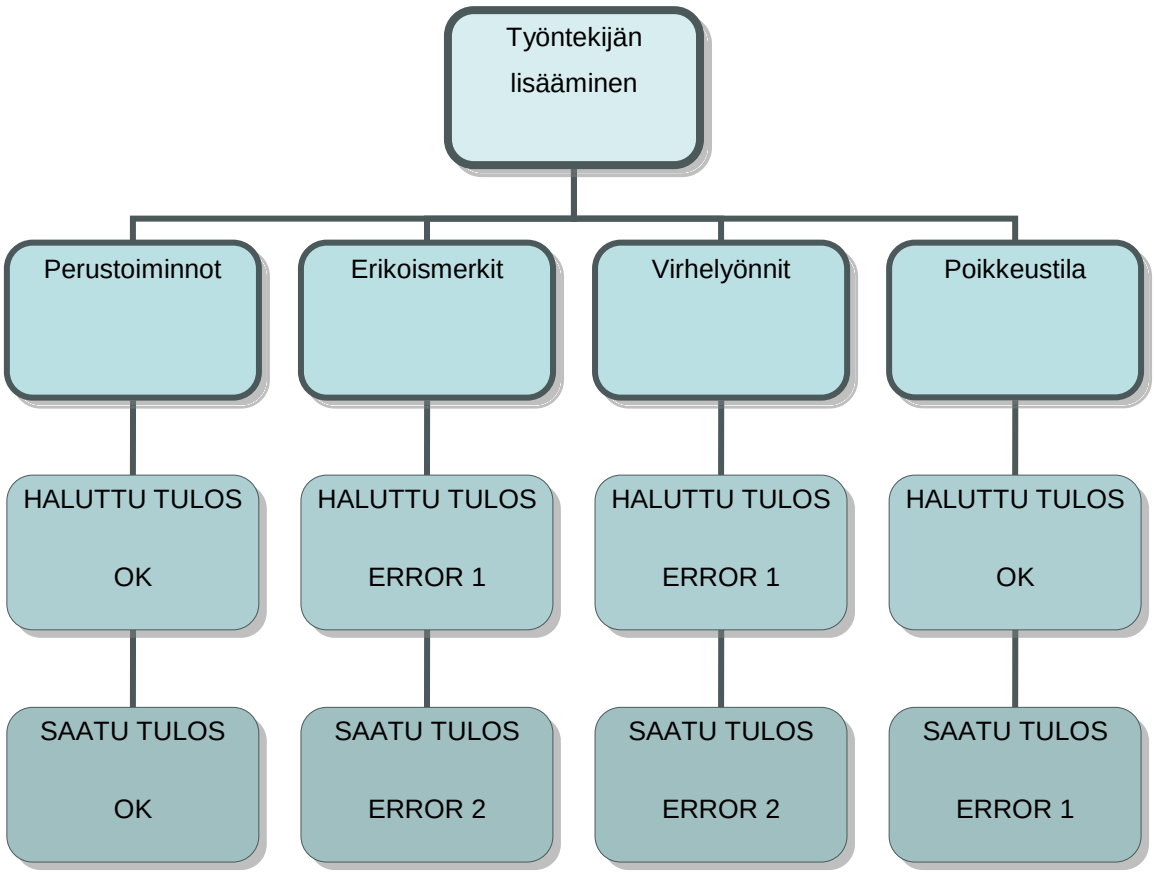


KUVA 8. Hallintapaneeli toimipisteiden lisäämisen jälkeen

Internetyhteyden katketessa sovellus kertoi, ettei toimipisteen lisääminen onnistunut, mikä on toivottu tulos.

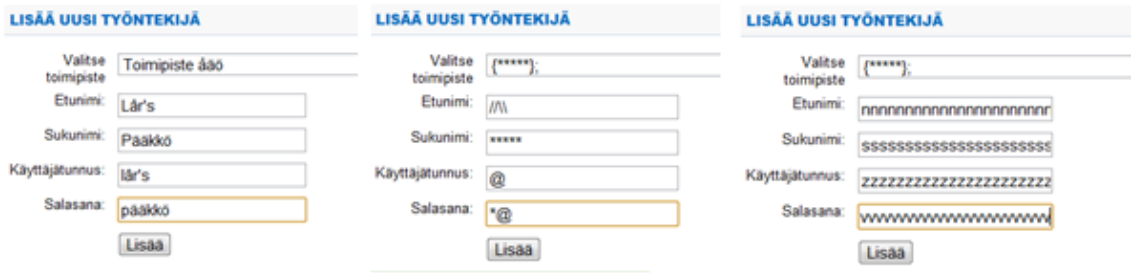
4.2.2 Työntekijän lisääminen

Kuvassa 6 on nähtävillä myös työntekijän lisäämistoiminto. Vaadittavat tiedot työntekijältä ovat nimi, käyttäjätunnus sekä salasana. Tämän lisäksi pyydetään toimipisteen valintaa. Näiden jälkeen painetaan Lisää-nappia. Kuvassa 9 on nähtävillä työntekijän lisäämisen toimintakaavio sekä odotetut ja saadut tulokset.



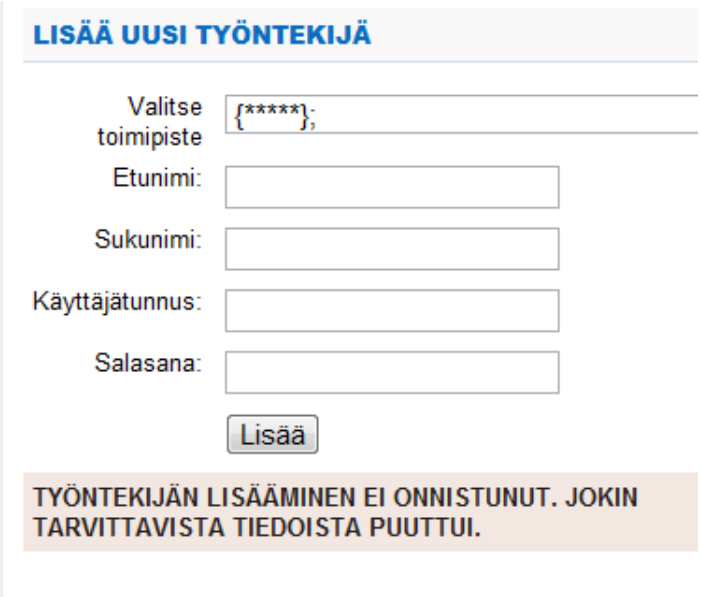
KUVA 9. Työntekijän lisääminen

Tässäkin toiminnossa sovellus hyväksyi kaiken, mitä siihen näppäili. Olisi toivottavaa, ettei sovellus hyväksyisi nimeksi tai osoitteeksi erikoismerkeillä alkavaa sanaa tai virhelyöntejä. Tämäkään ei tuottanut virhetilaa, joten vakavuusaste molemmille on 3. Toiminnossa on myös turhan suuri enimmäiskirjainmäärä, joka ei tuota muuta kuin kosmeettisen ongelman sovelluksen ulkonäköön. Sovellus hyväksyi salasanaksi erikoismerkit. Tämä on hyväksyttävää, mutta toivottavaa olisi minimimäärän säätäminen näihin osioihin. Kuvassa 10 on näkyvät erilaiset vaihtoehdot, jotka sovellus hyväksyi.



KUVA 10. Hyväksytyt työntekijän lisäämisvaihtoehdot

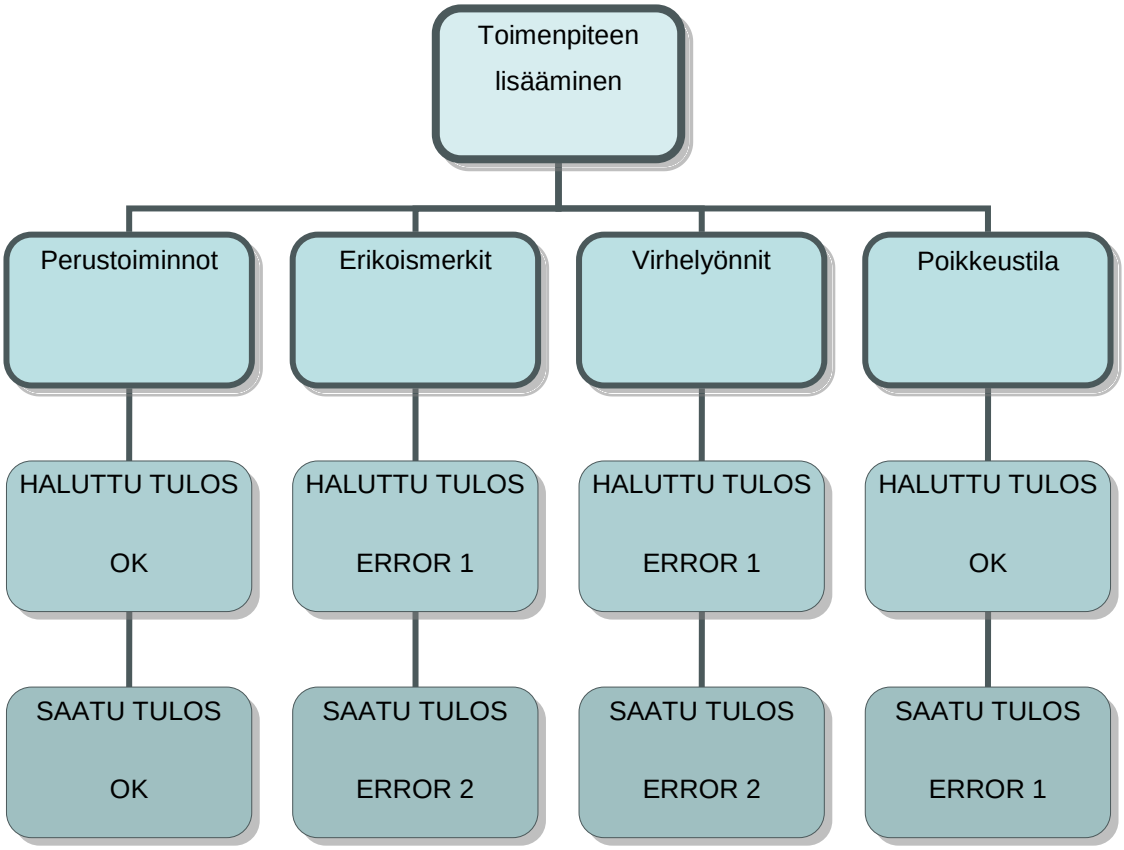
Internetin katketessa sovellus kertoi, ettei työntekijän lisääminen onnistunut, mikä on toivottu tulos. Saatu tulos on näkyvillä kuvassa 11. Sovellusta kokeiltiin myös jättämällä kenttiä tyhjiksi, jolloin sovellus ilmoitti virhetuloksen, joka oli toivottu tulos. Tässä on kuitenkin puutteena, ettei sovellus kerro, mikä tiedoista puuttuu. Vakavuusasteeltaan tämä on 5. Sovelluksen tulee kertoa, mikä tieto puuttuu.



KUVA 11. Virheilmoitus sovelluksessa

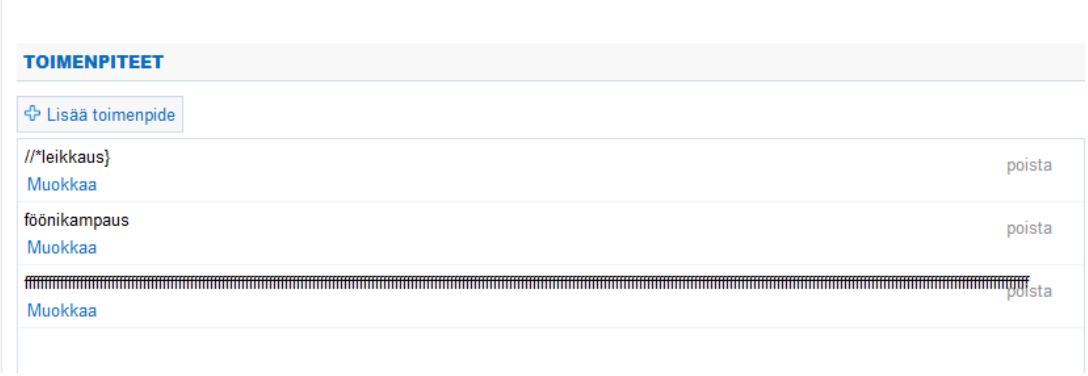
4.2.3 Toimenpiteen lisääminen

Pääkäyttäjä voi muokata toimenpiteiden lisäämistä sovelluksessa. Tässä on tarkoituksena nimetä toimenpide ja painaa sen jälkeen Lisää-nappia. Kuvassa 12 on nähtävillä toimenpiteen lisäämisen toimintakaavio sekä odotetut ja saadut tulokset.



KUVA 12. Toimenpiteen lisääminen

Sovellus hyväksyi kaikki merkit sekä toiminnot. Vakavuusaste erikoismerkkien sekä virhelyöntien hyväksymiseen on 3. Näillä ei ollut vaikutusta sovelluksen käyttämiseen, eikä sovellus joutunut virhetilaan. Kuvassa 13 on nähtävillä syötettyjen toimenpiteiden nimet.



KUVA 13. Syötetyt toimenpiteet

Internetyhteyden ollessa poikki sovellus ei ilmoita mitään. Tätä voisi pitää kosmeettisena virheenä, koska virhetilaa ei kerrota niin kuin aiemmin. Sovelluksen aukioloaikaa testattiin epäloogisella ajalla (esimerkiksi 8.30-6.30), jolloin sovellus ilmoitti virhetuloksen, mikä oli toivottu tulos.

4.3 Testaaminen loppukäyttäjä 1:n käyttökulmasta

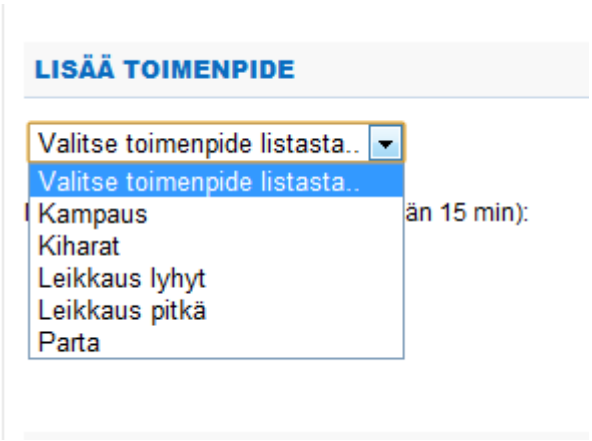
Loppukäyttäjä 1 on työntekijä, joka voi omalla tunnuksellaan määrittää omat työaikansa sekä muokata omia tietojaan. Loppukäyttäjä 1:lle on myös näkyvillä Asiakkaat-välilehti. Kun Loppukäyttäjä 1 kirjautuu sovellukseen ensimmäisen kerran tunnuksilla, jotka pääkäyttäjä on hänelle luonut, olisi toivottavaa, että sovellus käskisi loppukäyttäjä 1:n muuttamaan oman salasanansa. Tätä sovellus ei kuitenkaan tee, vakavuusasteeltaan tämä on 4.

Kohdassa 4.2.2 kokeiltiin perustoiminnoilla ja erikoismerkeillä työntekijöiden lisäämistoimintoa. Kirjaimet ä ja ö tulevat näkyviin kuvan 14 mukaisesti. Tämä ei estä sovelluksen toimivuutta, mutta ongelma on vakavuusasteeltaan 3.

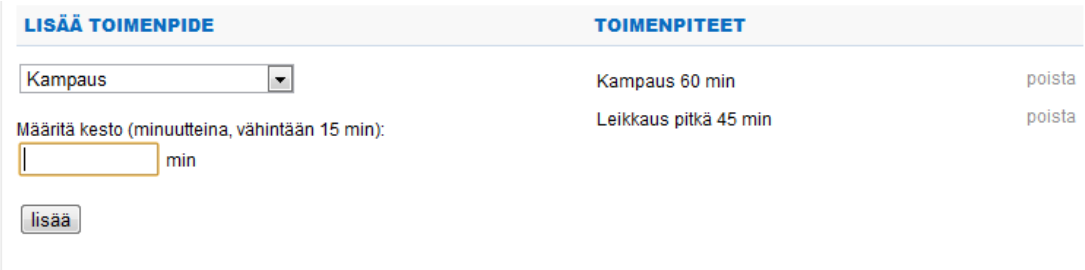
KUVA 14. Työntekijän nimen ä- ja ö-kirjainten näkyminen sovelluksessa

4.3.1 Toimenpiteen ajan määrittäminen

Jokainen loppukäyttäjä 1 määrittelee itse toimenpiteen keston. Tämä tapahtuu kuvan 15 ja 16 mukaisesti.

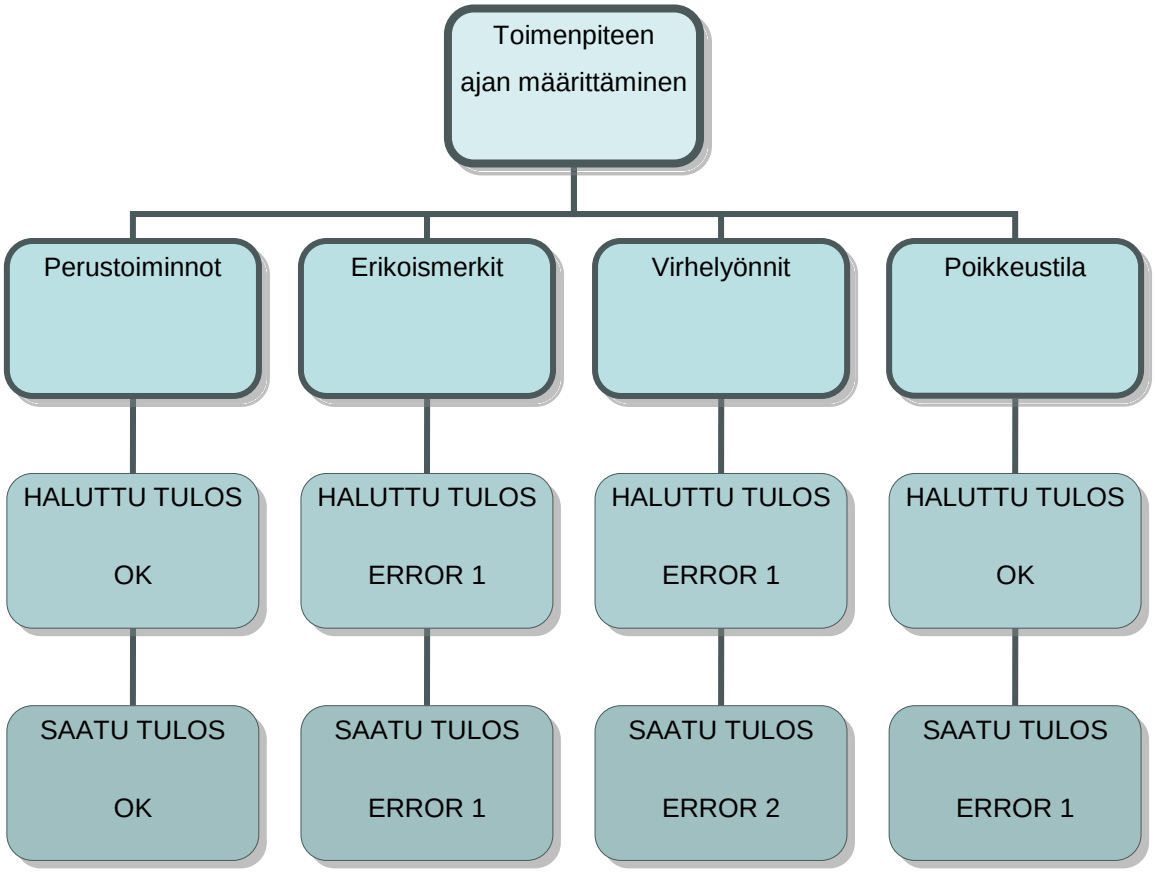


KUVA 15. Toimenpiteen valitseminen



KUVA 16. Toimenpiteen keston (min) määrittäminen

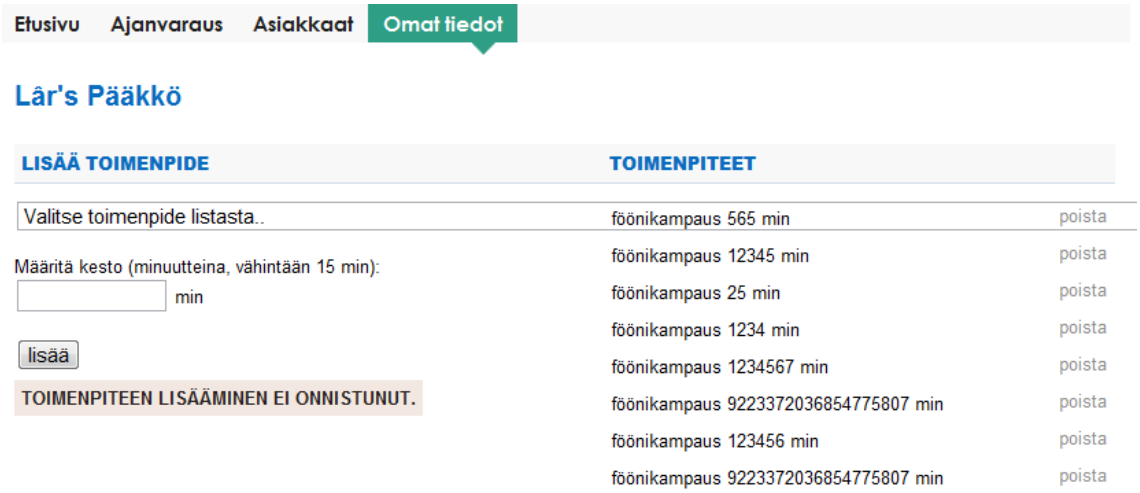
Kuvassa 17 on toimenpiteen ajan määrittämisen toimintakaavio sekä odotetut ja saadut tulokset.



KUVA 17. Toimenpiteen ajan määrittäminen

Sovellukseen kokeiltiin määrittää aika alle 15 minuutin, jolloin sovellus antoi virheilmoituksen (18). Kuvassa on näkyvillä myös muut kokeilut. Sovellus ei anna laittaa ajan määrittämisen kohtaan muuta kuin numeroita, mikä on todella

hyvä. Kosmeettisena virheenä ohjelmassa on se, ettei ole luokiteltu maksimiaikaa, esim. 8 tuntia = 480 minuuttia, joka vastaa normaalia työpäivää.

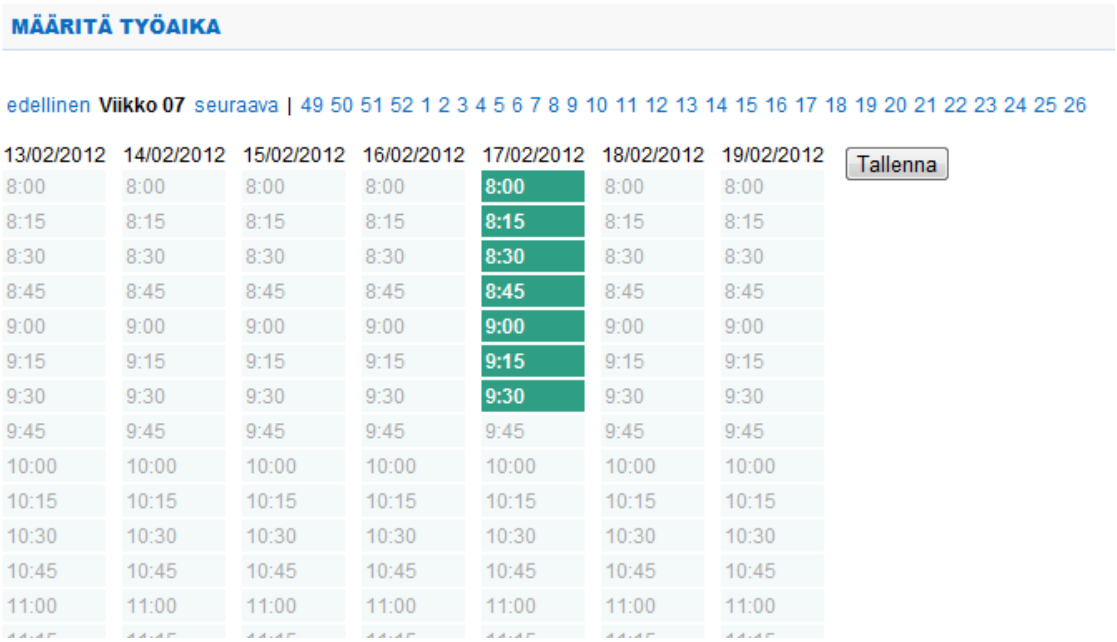


KUVA 18. Toimenpiteen ajan määrittämisen kokeilut

Internetyhteyden katketessa sovellus ilmoittaa virheilmoituksen.

4.3.2 Työajan määrittäminen

Loppukäyttäjä 1 määrittää itse omat työaikansa. Tämä tapahtuu kuvan 19 mukaisesti. Kun loppukäyttäjä 1 on maalannut työajat, painetaan painiketta Tallenna, jonka jälkeen ajat ovat näkyvillä loppukäyttäjä 2:lle.



KUVA 19. Loppukäyttäjän 1 työaikojen luominen

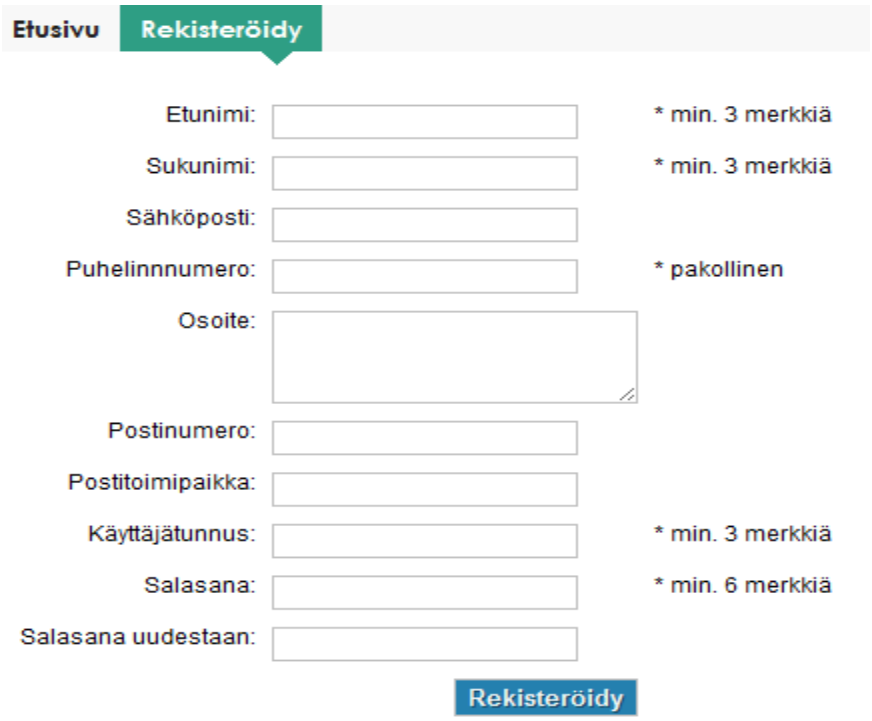
Tämä on helppo ja kätevä tapa. Tässä kuitenkin loppukäyttäjä 1:n tulee muistaa, että hänen halutessaan lähteä töistä klo 16 viimeinen maalattu ruutu on 15.45.

4.4 Testaaminen loppukäyttäjä 2:n käyttökulmasta

Loppukäyttäjä 2 on asiakas, joka voi rekisteröity sovellukseen, muokata omia tietoja, varata sekä peruttaa ajan.

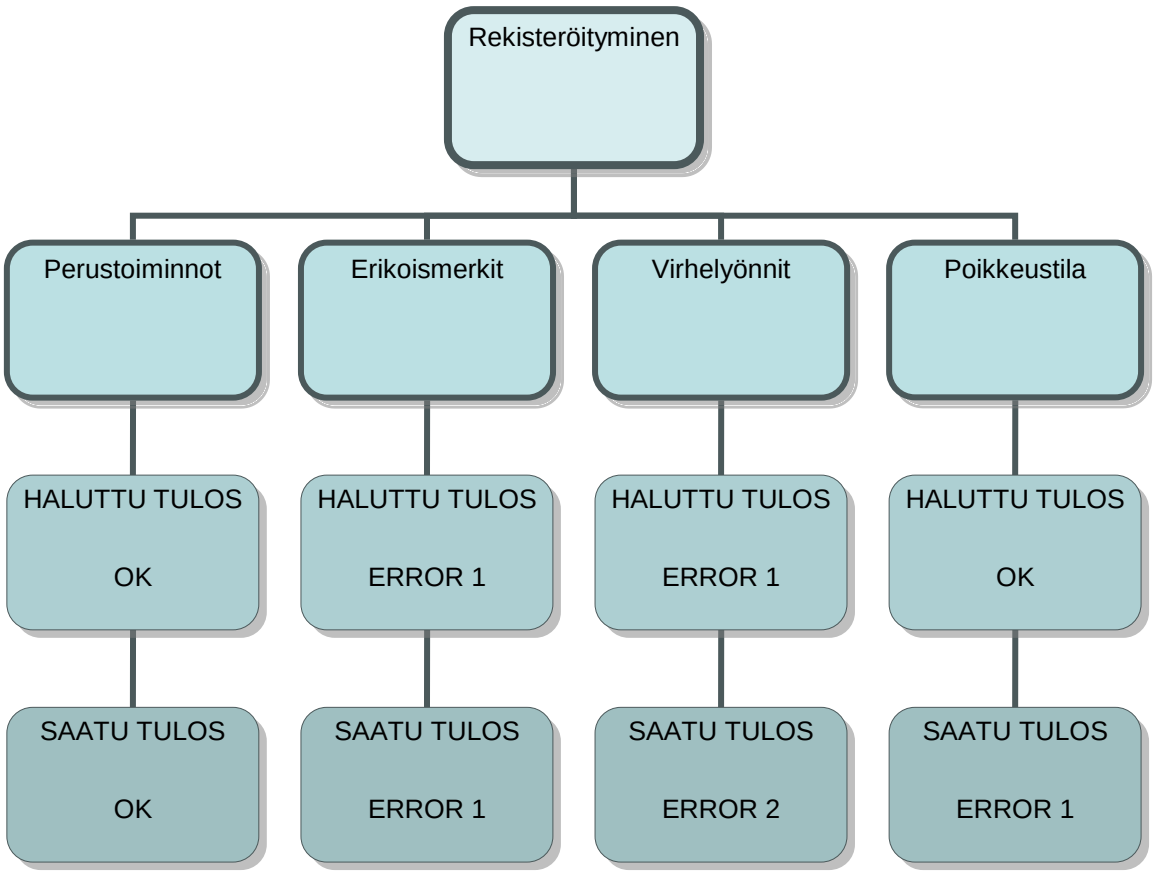
4.4.1 Rekisteröityminen

Loppukäyttäjä 2:n rekisteröityessä sovellukseen vaadittavia tietoja ovat nimi, puhelinnumero, käyttäjätunnus ja salasana. Rekisteröintisivu on nähtävillä kuvassa 20.



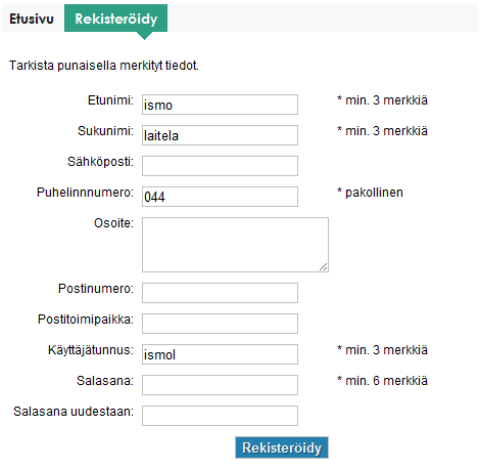
KUVA 20. Loppukäyttäjä 2:n rekisteröintisivu

Kuvassa 21 on nähtävillä loppukäyttäjä 2:n rekisteröityminen toimintakaavio sekä odotetut ja saadut tulokset.



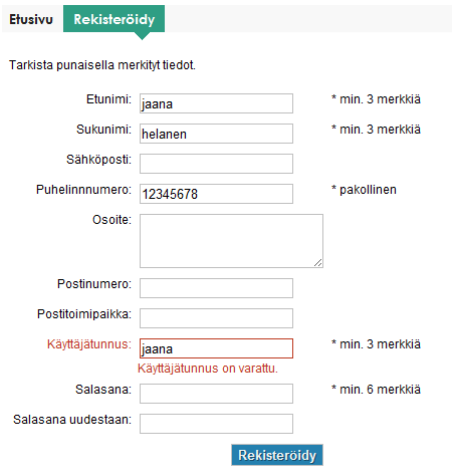
KUVA 21. Rekisteröityminen

Sovellus hyväksyy erikoismerkit muualle paitsi puhelinnumerokenttään. Puhelinnumero-kenttä vaatii kahdeksanmerkkisen numeron. Virheilmoituksessa ei kuitenkaan erikseen merkata, mistä kohdasta rekisteröitymisen epäonnistuminen johtuu. Sovellus latautumisen jälkeen on nähtävillä kuvassa 22. Vakavuusasteeltaan tämä on 5.



KUVA 22. Rekisteröinti ei onnistu, koska puhelinnumerokentässä liian vähän merkkejä

Kun yritetään rekisteröityä jo käytössä olevalla käyttäjätunnuksella, sovellus kertoo, mistä rekisteröitymisen epäonnistuminen johtuu. Tämä on nähtävillä kuvassa 23.

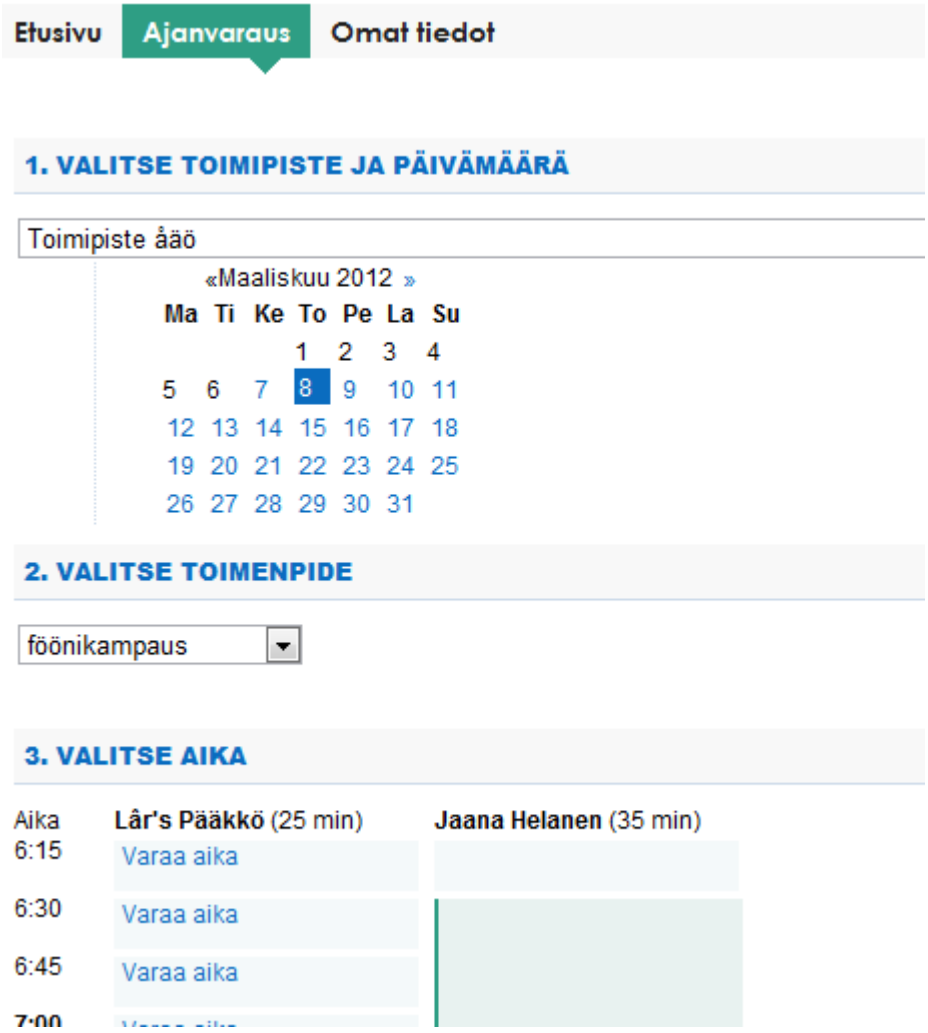


KUVA 23. Varattu käyttäjätunnus

Samantyylinen ratkaisu olisi hyvä myös puhelinnumeron ollessa viallinen. Internetyhteyden katketessa kirjoitetut arvot tyhjenevät ja pakollisten kenttien fontti muuttuu punaiseksi. Tämä on hyvä asia sovelluksessa. Kun loppukäyttäjä on kirjautunut palveluun, tulee sama näkymä kuin kuvassa 14, jos nimessä on ä- tai ö-kirjaimia.

4.4.2 Ajanvaraus

Aikaa varatessa loppukäyttäjä 2:n tulee ensin valita toimipiste, johon ajan haluaa. Tämän jälkeen valitaan päivämäärä ja toimenpide. Toimenpide-valikkoon tulevat esille vain siinä toimipisteessä tehtävät toimenpiteet. Käyttäjän valittua toimenpide avautuu alas kuvan 24 mukainen työntekijöiden aikataulu.



KUVA 24. Ajanvaraus ja vapaat sekä varatut ajat.

Kuvasta 24 huomaa, että ä- ja ö-kirjaimet loppukäyttäjä 1:n nimessä näkyvät normaalisti. Ajanvaraus on tehty mahdollisimman yksinkertaiseksi loppukäyttäjä 2:lle. Varausta tehtäessä sovellukseen tulee ilmoitus ”onnistui” tai ”epäonnistui”. Aikaa ei pysty varaamaan menneeseen aikaan, mutta sovelluksessa on

kompastuskivi, jonka mukaan ajan voi varata, jos toimenpiteen lopetusaika menee muutaman minuutin yli ajanvarausaikaa. Esimerkiksi klo 16.47 voi varata ajan alkavaksi klo 16, jos toimenpide kestää yli 50 minuuttia. Tämä on vakavuusasteeltaan 5. Aikaa ei kuitenkaan voi varata muualle kuin loppukäyttäjä 1:n määrittämille ajankohdille. Loppukäyttäjä 2 voi myös perua varaamansa ajan vaikka aika olisi menossa, esimerkiksi varaus klo 10 voidaan perua klo 10.30, jos toimenpide kestää 35 minuuttia. Vakavuusasteeltaan tämä on 5. Sovellukseen pitäisi määrittää aikaraja, jolloin peruminen ei enää onnistu. Aikarajan määrityksen tulisi olla riippuvainen pääkäyttäjän mielipiteestä. Loppukäyttäjä 2 voi varata niin monta aikaa kuin haluaa, mikä ei välttämättä ole hyvä asia. Tässä voisi sovellukselta tulla ilmoitus ”Teillä on jo varattuna aika”, jolloin ei tulisi mahdollisia väärinkäsityksiä.

Kohdassa 4.2.2 pystyi lisäämään työntekijän, jonka nimessä oli perustoimintoja, Lâr's Pääkkö. Syystä tai toisesta hänelle ei kuitenkaan voi varata aikaa. Sovellus ei ilmoita virheilmoitusta eikä tee tallennusta. Syynä ei ole ä- tai ö-kirjain, jonka vaikutusta kokeiltiin, joten syynä voi olla â-kirjain, jonka vuoksi sovellus ei tee mitään.

5 TULOKSET

Taulukossa 1 on listattuna sovelluksesta saadut virhetulokset, niiden vakavuusaste sekä korjausehdotukset. Sovellus ei kaatunut missään vaiheessa, mikä on hyvä. Yhden nimen kohdalla sovellus jumiutui, ja kyseessä on aika varmasti erikoismerkin käyttö nimen yhteydessä. Sovelluksessa oli jonkin verran pieniä kosmeettisia ongelmia ja muita vakavampia ongelmia. Vakavat ongelmat liittyvät loppukäyttäjä 2:n mahdollisuuksiin varata ja siirtää aikaa. On mahdollista, ettei tällaisia vaatimuksia ole tehty sovelluksen tekijälle alussa, joten ne ovat jääneet. Sovellus toimi kaikilla internetsovelluksilla samalla tavalla.

TAULUKKO 1. Saadut virhetulokset

Virhetapaus	**Kommentti**	**Vakavuusaste**	**Korjausehdotus**
Erikoismerkkien hyväksyminen	Salasanoissa ok, muissa ei	3	Ei mahdollista sanan alussa
Virhelyöntien hyväksyminen	Merkkien maksimimäärä	2	Maksimimäärä 50 merkkiä
Salasana	Ei minimimäärää	2	Olisi hyvä olla
Toimenpiteen ajan määrittäminen	Ei maksimiaikaa	2	Maksimimäärä esim. 480 min.
Tyhjät kentät	Ei ilmoita, mikä tieto puuttuu	5	Pitää näkyä, mikä puuttuu
Rekisteröinti ei onnistu	Ei näytä aina, mikä tieto puuttuu	5	Pitää näkyä, miksei onnistu
Ajan varaaminen menneisyyteen	Ajan voi varata, jos toimenpide kestää yli nykyisen ajan	5	Ei missään nimessä voi onnistua
Ajan peruminen	Ajan voi perua, kun aika on sillä hetkellä	5	Ei missään nimessä voi onnistua
Aikoja voi varata peräkkäin	Voi tulla väärinkäsityksiä	4	Pitäisi tulla ilmoitus "Teillä on jo aika"

Suurempana ongelmana sovelluksessa on tyhjien kenttien tai tietojen puutteellisuuden näkymätön ilmoitus. Käyttäjälle ei kerrota, mikä on se kohta, jota sovellus ei hyväksy, tai miksi sovellus ei mene eteenpäin. Loppukäyttäjä

2:n rekisteröityessä sovellukseen sähköpostin merkitseminen voisi olla pakollinen kenttä. Näin loppukäyttäjä 2:n varattua ajan sen voi varmistaa sähköpostiosoitteeseen.

Salasanan turvallisuus on tullut vasta pari vuotta sitten kunnolla käyttäjien tietoisuuteen, ja edelleenkään ei monissa sovelluksissa vaadita salasanalta erikoismerkintöjä. Salasanan luotettavuus sekä turvallisuus ovat käyttäjän vastuulla. Sen monipuolisuuden vaatiminen voisi olla kuitenkin luotettava merkki sovelluksesta. Virhelyöntien hyväksymistä, tässä viitataan lähinnä napin pohjaan jäämiseen, voisi rajoittaa. Kuitenkaan merkkejä ei tarvita mielettömiä määriä sovelluksessa, vaan 50 merkkiä toimipisteen, toimenpiteen, työntekijän ja asiakkaan kentissä pitäisi riittää. Erikoismerkkien hyväksyminen ei toisaalta tuota ohjelmalle mitään erikoista, joten niiden poistaminen on ajallisesti turhaa. Erikoismerkkien testaaminen työssä johtui halusta saada kokeilla, voiko sovelluksen saada kaadettua kokonaan. Tällaista lopputulosta ei kuitenkaan saatu aikaiseksi.

Käyttöliittymä sovelluksessa on yksinkertainen ja tehty jokaiselle käyttäjälle mahdollisimman helpoksi käyttää. Loppukäyttäjän 2:n ajan peruuttamista varten joutui hetken aikaa pohtimaan, mistä kyseinen toiminto löytyi.

6 POHDINTA

Insinöörityön tarkoituksena oli tutustua erilaisiin testauksen tasoihin ja menetelmiin ja näitä tietoja apuna käyttäen testata BelleBook-sovelluksen ajanvarausjärjestelmää mustalaatikkotestauksella. Työssä päästiin annettuihin tavoitteisiin, vaikka aikataulu hieman petti. Toisaalta aikataulun muuttuminen oli tilaajan kannalta hyvä, jolloin voitiin testata sovelluksen uusinta versiota. Työn aikana pystyi hyvin hyödyntämää hyvinvointiteknologian koulutuksen oppeja varsinkin käytettävyyden arvioinnin sekä ohjelmistotuotannon puolelta.

Työssä keskityttiin paljon teoriaan, jonka tarkoituksena oli tukea testejä. Testauksia suunnitellessa ja tehdessä huomasi, kuinka laaja sovelluksen testauksen pitää olla. Tässä myös huomasi, kuinka hyvä oli, että testaajana toimii joku muu kuin sovelluksen kehittäjä. Näin tuotteelle saatiin paras toimivuus sekä pääkäyttäjän, loppukäyttäjän 1 että loppukäyttäjän 2 näkökulmista. Työtä olisi tukenut hyvin vaatimusmäärittely, joka on annettu sovelluksen kehittäjälle. Silloin olisi voinut suoraan testata vaatimusmäärittelyjen toimintaa sekä sitä, kuinka tilaaja on halunnut sovelluksen toimivan. Jos esimerkiksi ajan peruuttamiseen on vaadittu tietty aikaraja, jolloin sen vielä pystyy tekemään, olisi saatuja tuloksia voinut suoraan peilata niihin. Työssä auttoi myös palvelualalla työskentelevä henkilö, joka kiinnitti heti näihin seikkoihin huomiota. Palvelualalla yleensä vaaditaan peruuttamis- tai myöhästymismaksu, koska vain tehdystä työstä saadaan yritykseen rahaa.

Mustalaatikkotestaus menetelmä soveltui hyvin Bellebook-sovelluksen testaamiseen. Mustalaatikkotestauksen periaate on, ettei koodia ole saatavilla ja testataan rajapintoja. Tiedetään etukäteen, miten sovelluksen tulisi toimia, ja haetaan virheitä toimimalla ulkoapäin sovellukseen. Testaustapauksia oli yllättävän vaikea miettiä. Työn alussa ei tärkeimpänä ajatuksena ollut saada mahdollisesti sovellus kaatumaan, vaan lähinnä vakuuttua sovelluksen toimivuudesta. Lopputuloksena kuitenkin sai hyviä tuloksia sekä tilaajalle että

sovelluksen kehittäjälle. Työssä oli haastavuutta sekä mielenkiintoa. Aihe sopi hyvin insinöörityöksi ja siinä pystyi hyödyntämään saatuja oppeja.

LÄHTEET

Helpdesk - tietotekniikan neuvonta ja käyttötuki 2007. Saatavilla: http://www.helsinki.fi/atk/neuvonta/ohjehakemisto/salasana.html. Hakupäivä 19.2.2012.

Holappa, Terhi 2011. T213003 Käytettävyyden arviointi, 3 op. Opintojakson opetusmateriaali keväällä 2011. Oulu: Oulun seudun ammattikorkeakoulu, tekniikan yksikkö.

Holvikivi, Jaana 2010. Ohjelmointi-luentomateriaali. Helsinki: Helsingin ammattikorkeakoulun tekniikan osasto.

Jänkälä, Ville 2012. Re:BelleBookin käyttöjärjestelmä. Sähköpostiviesti. Vastaanottaja: Jaana Helanen. 7.3.2012

Koskinen, Jukka – Kelo, Tomi – Mähönen, Timo – Oikarinen, Anne – Salo, Vesa – Vartiainen, Tuure 2008. Tietoliikenteen turvallisuus. Seminaari. Tampereen teknillinen yliopisto. Saatavilla: http://www.cs.tut.fi/kurssit/TLT-3700/hhh-sem.pdf. Hakupäivä 16.3.2012.

Laine, Harri 2001. Ohjelmistotuotannon-luentomateriaali. Helsinki: Helsingin yliopisto, tietotekniikan osasto.

Marttinen, Viktor 2010. PHP-pohjaisen sovelluksen testaus. Helsinki: Haaga-Helia, Tietojenkäsittelyn koulutusohjelma. Opinnäytetyö.

Pressman, Roger S. 2001. Software engineering, 5th edition. Singapore. McGraw-Hill Book Co.

Sommerville, Ian 2004. Software engineering, 7th edition. United States of America. Pearson Education Limited.

Tersa, Tiina 2002. Testausmenetelmien käyttö sovelluksen systeemitestausvaiheessa. Jyväskylä: Jyväskylän yliopisto, Tietotekniikan laitos. Pro gradu -tutkielma.

Väisänen, Veijo 2010. T241003 Ohjelmistotuotanto, 3 op. Opintojakson opetusmateriaali syksyllä 2010. Oulu: Oulun seudun ammattikorkeakoulu, tekniikan yksikkö.

Wikipedia. 2012. Vapaa tietosanakirja. Saatavilla: http://fi.wikipedia.org/wiki. Hakupäivä 17.3.2012.