Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Vili Niemi

ERP-järjestelmän toiminnallisuuksien testaus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Hyvinvointi- ja Terveysteknologia

Insinöörityö

13.04.2021

Tekijä Otsikko Sivumäärä Aika	Vili Niemi ERP-järjestelmän toiminnallisuuksien testaus 54 sivua 13.04.2021
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja Viestintätekniikka
Ammatillinen pääaine	Hyvinvointi- ja Terveysteknologia
Ohjaaja	Lehtori Juha Havukumpu

Opinnäytetyön aiheena oli testata toimeksiantajan yrityksen ERP-järjestelmän toiminnallisuuksia. Toiminnallisuuksien testaus toteutettiin ison projektin muodossa, jossa päivitettiin potilaille tarkoitettu asiointijärjestelmä. Vanhan järjestelmän skaalautuvuus ja toiminnallisuudet eivät vastanneet organisaation nykypäivän vaatimuksia. Uudistetun järjestelmän olennaisena edellytyksenä on sen mobiilikäyttöisyys, mikä parantaa toiminnan saavutettavuutta. ERP-ohjelmiston testauksella varmistettiin sen laadukkuus, yhtenäisyys ja turvallisuus. Tekstin teoriaosassa käsitellään kaksi isompaa kokonaisuutta: ERP-järjestelmä ja ohjelmistotestaus.

Projektin alkuun perehdyttiin organisaation käytössä olevaan ERP-järjestelmään. Ensin tutkittiin järjestelmän yleistä koostumusta. Myöhemmin syvennyttiin ohjelmiston arkkitehtuuriseen rakenteeseen. Näiden pohjalta luotiin kokonaiskuva käytössä olevasta järjestelmästä. Testausprosessille laadittiin tarkat dokumentit testauksen läpiviemisestä. Näissä dokumenteissa määriteltiin päivitettävän järjestelmän oleelliset vaatimukset.

Testausprosessin aikana kehityksen alla olevasta järjestelmästä löydettiin sekä isoja että järjestelmän toimintaan haitallisesti vaikuttavia virheitä. Myös vakavuusasteeltaan lievempiä ongelmia havaittiin. Kaikki tunnistetut virheet kirjattiin bugiraportteihin, jotka edelleen siirrettiin kehitystiimille korjattavaksi.

Kehitysprojekti saatiin vietyä onnistuneesti loppuun ja loppukäyttäjille julkaistiin päivitetty, toiminnoiltaan modernimpi ja helppokäyttöisempi asiointijärjestelmä. Testausten ansiosta järjestelmän laadukkuus, eheys ja turvallisuus saatiin varmistettua. Näin ollen sekä kehitysprojektin että testausprosessin tavoitteet saavutettiin. Kaiken kaikkiaan opinnäytetyön aikana opittiin paljon projektityölle ominaisia asioita. Projektista omaksuttiin kehitettäviä asioita niin teknisen osaamisen kuin etätyöskentelyn suhteen.

Avainsanat	ERP-järjestelmä, ohjelmistotestaus, kehitysprojekti

Author Title Number of Pages Date	Vili Niemi Testing of ERP Systems Functionalities 54 pages 13.04.2021
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Healthcare and Welfare Technology
Instructor	Juha Havukumpu, Senior Lecturer

The purpose of the study was to test the functionalities of the ERP system of the client company. Functionality testing was carried out in the form of a large project that updated the transaction system for patients. The scalability and functionality of the old system did not meet today's requirements of the organization. An essential prerequisite for the renewed system is its mobile usability, which improves the accessibility of operations. Testing of the ERP software ensured its quality, integrity and security. The theoretical part of the study deals with two larger entities: the ERP system and software testing.

At the beginning of the project, the ERP system used by the organization was studied in detail. First, the general composition of the system was examined. Later, the architectural structure of the software was delved into. Based on this, an overall picture of the system in use was created. Accurate documentation of the testing process was prepared for the testing process. The documents defined the essential requirements for the system to be updated.

During the testing process, both major and adverse errors were found in the system under development. Also less severe problems were observed. All identified errors were recorded in bug reports, which were further forwarded to the development team for correction.

The development project was successfully completed, and an updated transaction system with more modern functions and a system easier to use, was published to the end users. Thanks to the tests, the quality, integrity and security of the system were ensured. Thus, the objectives of both the development project and the testing process were achieved. The project embraced the issues to be developed in terms of both technical competence and teleworking.

Keywords	ERP system, software testing, development project

Sisällys

Lyhenteet

ERP	Enterprise Resource Planning. Toiminnanohjausjärjestelmä. Yrityksen toiminnan, liiketoiminnan sekä resurssien ohjaamiseen suunniteltu tietojärjestelmäkokonaisuus.
CRM	Customer Relationship Management. Asiakkuudenhallintajärjestelmä. Yrityksen asiakassuhteiden hallintaan suunniteltu tietojärjestelmä.
MRP	Material Requirements Planning. Tuotannonohjausjärjestelmä. Yrityksen tuotannon ohjaamiseen suunniteltu tietojärjestelmä.
SOA	Service-Oriented Architecture. Palvelukeskeinen arkkitehtuuri. Ohjelmointitekniikka, jossa käytetään hyödyksi toisistaan eriäviä, itsenäisiä sovelluskomponentteja erilaisten kommunikaatioprotokollien avulla.
SaaS	Software as a Service. Ohjelmisto palveluna. Tunnettu myös yleisesti pilvipalvelu-nimellä. Ohjelmisto, jota käytetään usein verkkoselaimella internetin avulla. Ohjelmisto itsessään sijaitsee palvelimella.
API	Application Programming Interface. Ohjelmointirajapinta. Ohje, jonka mukaan ohjelmat voivat keskustella keskenään ja hyödyntää toistensa tietoja.
SME	Small and Medium Enterprise. PK-yritykset. Kooltaan pienet ja keskisuuret yritykset.
GUI	Graphical User Interface. Graafinen käyttöliittymä. Käyttöliittymäkomponenteista muodostuva kokonaisuus. Tarkoitetaan myös käyttöliittymäelementteihin pohjautuvaa tapaa käyttää tietokonetta tai järjestelmää.
UAT	User Acceptance Testing. Hyväksymistestaus. Testaustyyppi, jossa testataan, että ohjelmistolle määritellyt vaatimukset toimivat niin kuin niiden oletetaan toimivan.

1 Johdanto

Opinnäytetyö tehtiin Suomessa toimivalle terveydenhuoltoalan yritykselle. Testaus ja erityisesti ERP-järjestelmän (*eng. Enterprise Resource Planning*) toiminnallisuuksien testaus sekä yleinen järjestelmän laadunvarmistus nousivat esille tilaavan yrityksen kanssa keskusteltaessa mahdollisesta opinnäytetyöstä. Yritykselle erityisen tärkeänä asiana on ERP-järjestelmän jatkuva testaus, johon opinnäytetyö lopuksi rajattiin. Ohjelmiston testauksen avulla pystytään minimoimaan järjestelmässä tapahtuvat suuremmat virheet ja varmistamaan järjestelmän toimivuus ja eheys jatkuvassa käytössä uusia toiminnallisuuksia lisätessä sekä vanhoja päivitettäessä.

Eritoten teknisen tason testaaminen on ERP-järjestelmää mietittäessä elintärkeässä roolissa, jotta pystytään takaamaan järjestelmän ja liiketoiminnan jatkuva kehitys. Järjestelmän toiminnallisuuksia testataan, jotta pystytään takaamaan järjestelmän laadukkuus, toimivuus ja eheys sen koko elinkaaren aikana. ERP-järjestelmässä ilmenevä vika tai virhe tulee sitä kalliimmaksi, mitä myöhemmässä vaiheessa se huomataan. Pahimmassa tapauksessa virhe voi aiheuttaa koko tuotannon tai liiketoiminnan pysähtymisen.

Opinnäytetyön tavoitteena on ERP-järjestelmän testaaminen, ja yksityiskohtaisemmin järjestelmän koko elinkaaren aikainen testaus. Tavoitteena on siis voida testauksen avulla varmistaa, että ERP-järjestelmä on käytettävyydeltään sekä muilta toiminnallisuuksiltaan laadukas ja toimiva niin asiakkaiden kuin ammattilaisten käyttöön. Suoritettava tutkimustyö on luonteeltaan hyvin käytännönläheistä. Perustava tieto järjestelmästä hankitaan perehtymällä järjestelmän toimintaan yleisellä tasolla, ja myöhemmin syventymällä ERP-järjestelmän arkkitehtuuriin kooditasolla. Opinnäytetyön tavoitteena on pyrkiä vastaamaan tutkimuskysymyksiin ”*Toimiiko ERP-järjestelmä niin kuin halutaan?”, ”Miten ERP-järjestelmän testaus tukee palvelun kehitystä?”* ja ”*Tuottaako ERP-järjestelmä liiketoiminnalle sen tarvitsemat hyödyt?”*.

Tulevassa tekstissä käsitellään ensin isona kokonaisuutena ERP-järjestelmät. Myöhemmin tekstissä toisena isona kokonaisuutena käydään läpi ohjelmistotestaus. Tämän jälkeen käsitellään itse tutkimustyössä suoritetut testaukset toimeksiantajan yrityksessä, sekä näiden aikana ilmenneet tutkimustulokset.

2 ERP-järjestelmä

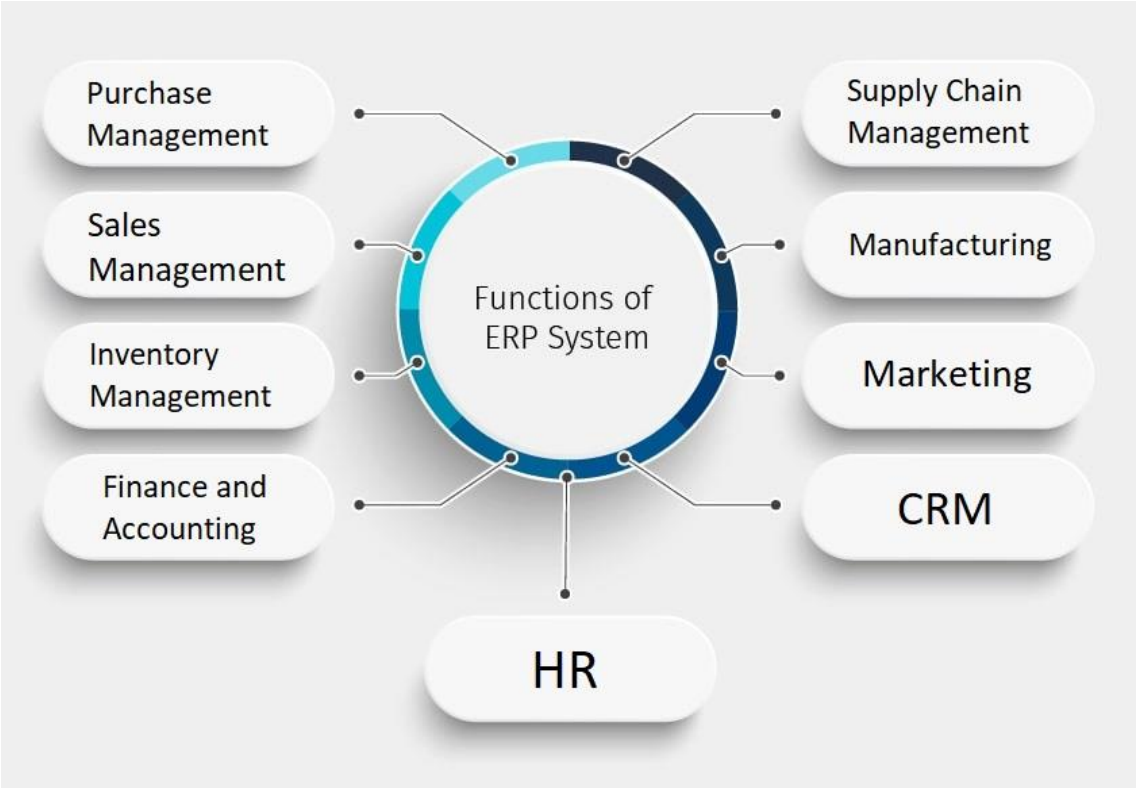
2.1 ERP-järjestelmä yleisesti

ERP-järjestelmällä tarkoitetaan yrityksen toimintaan, liiketoimintaan sekä resurssien hallintaan suunniteltua ohjelmistojärjestelmää. ERP-järjestelmät sisältävät yrityksen toimintaan sopivia elementtejä, kuten esimerkiksi kirjanpitoa, henkilöstöhallintaa, varastonhallintaa, myyntitapahtumienhallintaa ja tuotannonohjausta. ERP-järjestelmä siis yksinkertaistettuna yhdistää useita eri liiketoiminnan kannalta oleellisia toimintoja ja prosesseja ja huolehtii niiden välisestä kommunikaatiosta. [1.]

Moderneissa ERP-järjestelmissä yritykset voivat valita omaan toimintaan ja liiketoimintaan sopivan järjestelmäratkaisun. Eri toiminnot on usein jaettu omiin moduuleihinsa, kuten esimerkiksi laskutuksen tai toiminnanohjauksen moduulit. Yritykset pystyvät myös tarvittaessa päivittämään ERP-järjestelmäänsä uusilla järjestelmämoduuleilla sekä vanhojen moduulien entisöinneillä. [1.]

Kaikkien ERP-järjestelmän toimintojen ei tarvitse olla yhdessä isossa ERP-kokonaisuudessa, vaan ERP-järjestelmän rinnalle voidaan myös kehitellä niin sanottuja erillisjärjestelmiä. Erillisjärjestelmät käyttävät ja jakavat samoja tietoja ERP-järjestelmän kanssa, ja kommunikoivat keskenään tiiviisti. Esimerkki tällaisesta järjestelmästä on CRM-järjestelmä. [2; 3.]

Kuva 1 [4] on alkuperäisestä lähteestä muokattu tekstiin sopivaksi. Kuvassa on kuvattu yleisesti ERP-järjestelmien rakennetta. Kuvan keskellä on visualisoitu isona ympyränä ERP-järjestelmäkokonaisuus, ja reunoille on linkitetty ERP-järjestelmään implementoitavat ominaisuudet erillisinä toimialuekokonaisuuksina. Kaikki ERP-kokonaisuuteen valitut elementit on liitetty toisiinsa ERP-järjestelmäsovelluksen avulla. Myös sen mahdolliset ulkopuoliset kokonaisuudet, esimerkiksi CRM-järjestelmä (*eng. Customer Relationship Management*), ovat tiiviisti yhteydessä itse ERP-järjestelmään.



Kuva 1. ERP-järjestelmien yleinen rakenne. [3]

Yritys voi ERP-järjestelmällä parantaa ja tehostaa yleistä toimintaansa. Liiketoiminnan eri osa-alueet tuottavat tietoa toiminnastaan ja samanaikaisesti tallentavat kyseisiä tietoja ERP-järjestelmän avulla. Tiedot ovat näin ollen haettavissa reaaliaikaisesti yrityksen eri osastojen välillä. Kokonaisuudessaan tämä ERP-järjestelmän toimintaprosessi nopeuttaa toimintaa vanhoihin manuaalisiin tiedonsiirron rakenteisiin verrattuna. [2.]

2.2 ERP-järjestelmien kehitys

Ensimmäiset ERP-järjestelmien edeltäjät löytyvät jo 1960-luvulta, jolloin sovelluksia käyttivät enimmäkseen isot kansainväliset eri tuotannonalojen yritykset varastojen hallinnassa sekä laadunvarmistuksessa. Tällaiset varastonhallinnan järjestelmät rakennettiin yleensä yhtiön sisällä, eivätkä niiden toiminnot olleet kovin laajoja. Alun perin ERP-järjestelmän kaltaisien järjestelmien tarkoituksena olikin kasvattaa juuri isojen tuotantoyritysten kannattavuutta ja tehokkuutta. Tämän lisäksi ohjelmistoilla pyrittiin inhimillisten virheiden minimoimiseen tuotannon eri vaiheissa. [5.]

Varastonhallintaan tarkoitetut sovellukset kehittyivät edelleen 1970-luvulla entistä moderneimmiksi. Yksi isoimmista kehitysaskeleista oli, kun IBM rakensi sovellusarkkitehtuuriltaan entistä monimutkaisemman tuotannon hallinnointiin, suunnitteluun ja ohjaamiseen tarkoitetun järjestelmän. Tätä päivittäiseen käyttöön tarkoitettua, toiminnoiltaan laajempaa järjestelmää kutsuttiin MRP-ohjelmistoksi (*eng. Material Requirements Planning*). Suuri haaste 1970-luvun kehityksessä oli MRP-sovellusten vaatima suuri laitteistokoko sekä korkea hintataso. [5; 6.]

Kun 1980-luvulla laitteistot ja teknologiat muuttuivat yhä yleisimmiksi ja tehokkaammiksi, myös sen aikaiset MRP-järjestelmät alkoivat muotoutua laajemmiksi, koko yritysorganisaation työkaluiksi. Nämä niin kutsutut MRP II -järjestelmät yhdistivät samaan järjestelmään useita yrityksen toimialueita, kuten esimerkiksi varastonhallinnan, henkilöstöhallinnan ja markkinoinnin. [5; 6.]

Itsessään ERP-järjestelmä käsitteenä mainittiin vasta 1990-luvulla. Gartner Groupin ensimmäisenä mainitsemat ERP-järjestelmät lisäsivät yhä edistyneempiä ja monipuolisempia toimintoja yrityksen toiminnan eri prosesseihin. 1990-luvulla valtaosa ERP-järjestelmiä käyttävistä yrityksistä oli suuria. Pienet ja keskisuuret yritykset alkoivat ottaa yhä enemmän käyttöön ERP-järjestelmiä, mutta niiden alkuvaiheen kustannukset olivat vielä korkeat. [7; 8.]

Vuosituhannen alussa ERP-järjestelmät edistyivät etenkin etäyhteyksien sovellettavuuden osalta. Tämä mahdollisti yhteyden ottamisen ohjausjärjestelmään käyttäen internetyhteyttä. Erilaisiin järjestelmäkokonaisuuksiin alettiin myös soveltaa kehittyneempää, niin sanottua SOA-ohjelmointimallia (eng. *Service-Oriented Architecture*). Myös mobiililaitteilla ERP-järjestelmien käyttäminen mahdollistui 2000-luvulla. [8; 9.]

Nykypäivän modernit ERP-järjestelmät ovat täysin integroitavia kokonaisuuksia, jotka muokkautuvat sitä käyttävän organisaation tarpeisiin. Viimeisen vuosikymmenen aikana ERP-järjestelmät ovat siirtyneet entistä enemmän SaaS-pohjaisiksi (*eng. Software as a Service*). ERP-järjestelmiä on nykyään tarjolla pilvipalveluina, perinteisinä paikallissovelluksina, sekä näiden kahden hybridivaihtoehtona. Tämä on antanut mahdollisuuden myös pienille ja keskisuurille yrityksille ottaa käyttöönsä ERP-järjestelmä. Näitä nykypäivänä käytettävien ERP-järjestelmien eroavaisuuksia käydään läpi myöhemmissä luvuissa. [7; 8.]

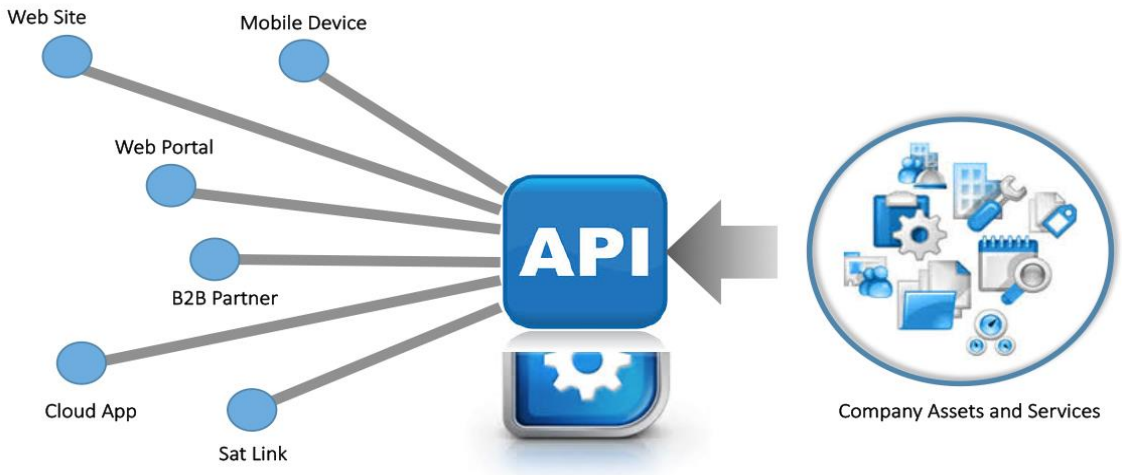
2.3 Sovellusrajapinnat

Ennen kuin voidaan puhua ERP-järjestelmien teknisestä puolesta, täytyy tietää, mitä ohjelmointirajapinnalla tai sovellusrajapinnalla, eli API:lla (eng. *Application Programming Interface*), tarkoitetaan. On myös hyvä tietää, miten erilaiset ERP-järjestelmät hyväksikäyttävät näitä rajapintoja.

API:lla tarkoitetaan ohjelmien, ohjelmakomponenttien tai erilaisten alustojen ohjeita, joiden avulla ne voivat keskustella keskenään. Yksinkertaistettuna API:t ovat sääntöjä, joita noudattamalla ohjelmat vaihtavat ja jakavat tietojaan. [10.]

Rajapintojen tarkoituksena on tehdä erilaisten ohjelmistojen ja järjestelmien rakenteesta selkeitä: vaikka itse järjestelmän arkkitehtuurinen rakenne olisi monimutkainen, sovellusrajapintojen avulla erilaisten palvelupyyntöjen sekä toimintojen suorittaminen on yksinkertaista ja helppoa. Näin ollen rajapintoja voidaan myös pitää ohjelmistojen sekä erilaisten järjestelmien organisointia helpottavina työkaluina. [10.]

Kuvassa 2 [11] on kuvattu, miten yritysten erilaiset järjestelmät hyväksikäyttävät API-rajapintoja. Oikealla puolella on esitetty joukko yrityksen resursseja ja palveluita. Näitä resursseja ja palveluita käytettäessä syötetty tieto- tai palvelupyyntö siirtyy eteenpäin eri sovellusrajapinnoille, jotka edelleen kääntävät tiedon päätesovellukselle ymmärrettäväksi yksinkertaiseksi toimintapyynnöksi.



Kuva 2. Yrityksen palveluiden ja sovellusrajapintojen yhteys päätesovelluksiin. [11]

Ennen kaikkea sovellusrajapintoja käytetään nykyään erilaisten järjestelmien integroinneissa. Alun perin rajapintoja käytettiin erilaisten ohjelmien sisäisten ohjelmointikomponenttien yhdistämiseen. Myöhemmin verkko-ominaisuuksien kehittyessä rajapintojen käyttö siirtyi entistä enemmän erilaisten ulkoisten palveluiden integrointeihin. [10.]

Yleisesti ottaen API:t voidaan jakaa kolmeen pääluokkaan: yksityisiin rajapintoihin, kumppanirajapintoihin sekä avoimiin rajapintoihin. Rajapintojen erottavat tekijät liittyvät niiden näkyvyyteen ja käytettävyyteen. [12.]

Yksityiset rajapinnat ovat ainoastaan organisaation sisäiseen käyttöön tarkoitettuja rajapintoja. Tyypillisesti yksityisiä rajapintoja hyödynnetään yritysten sisäisten palveluiden ja ohjelmistojen kehittämiseen ja integrointiin. Yksityiset API:t ovat siis täysin niitä käyttävien yritysten hallinnoitavia, ja ne harvoin ovat näkyvissä ulkopuolisille osapuolille. [12.]

Kumppanirajapinnat ovat nimensä mukaan näkyvissä ja käytettävissä kahden tai useamman yhteistyöorganisaation välillä. Yritykset siis sopivat näiden rajapintojen yhteiskäytöstä sopimuksen myötä. Tyypillinen käyttötarkoitus kumppanirajapinnoille on yritysten välisten sovellusten tai palveluiden integrointi: tämän avulla yritykset voivat hyödyntää toistensa palveluita ja samalla seurata näiden palveluiden välistä tietoliikennettä. [12.]

Avoimet rajapinnat ovat kaikille osapuolille näkyviä ja käytettäviä rajapintoja. Avoimien API:en hyödyt ovat niiden käytön helppoudessa ja avoimuudessa. Kuitenkin näiden rajapintojen käyttöön liittyy usein myös haasteita yleisen tietoturvan ja tietojen jakamisen suhteen. [13.]

Modernit yritykset käyttävät ERP-järjestelmissään paljon erityyppisiä rajapintoja sovellusten ja palveluiden väliseen kommunikointiin. Tyypillisesti ERP-kokonaisuuksissa käytetään hyödyksi useampia eri rajapintaluokkia. Rajapintojen hyödyntäminen ERP-järjestelmissä näin ollen parantaa ERP-kokonaisuuksien toimivuutta sekä helpottaa sen sisäisten ja ulkoisten palveluiden välistä tiedonkulkua. [14.]

2.4 ERP-järjestelmien jako

Eri ERP-järjestelmät voidaan jakaa useisiin eri kategorioihin. Yritysten tulisikin arvioida omalle toiminnalleen oleelliset toiminnot, ja sen pohjalta valita heidän toimintaansa parhaiten sopiva ERP-järjestelmäkokonaisuus. Yleisimpiä ERP-järjestelmiä jakavia tekijöitä ovat:

- yrityksen koko
- yrityksen toimiala
- palvelutyyppi
- arkkitehtuurirakenne.

Näiden elementtien pohjalta yritykselle juuri sopivan ERP-järjestelmän valinta voi olla iso asia yrityksen toiminnan, kasvun ja sujuvuuden kannalta. Erilaisten ERP-järjestelmien kirjo on nykypäivänä todella laaja. Tämän vuoksi ERP-järjestelmää hankittaessa tai päivitettäessä yrityksen tulisi aina tehdä perusteellinen analyysi sopivan järjestelmästä valinnasta. [15.]

2.4.1 ERP-järjestelmien jako yrityksen koon mukaan

Yrityksen koko vaikuttaa ERP-järjestelmän valintaan suuresti. Erilaisia ERP-järjestelmiä on tarjolla eri kokoisille yrityksille, aina isojen kansainvälisten yritysten laajoista järjestelmistä, pienempien yritysten tarpeet täyttäviin järjestelmiin. Eri kokoisten ERP-järjestelmien etukäteis- ja ylläpitokustannukset ovat isoin peruste järjestelmätarjoajien jakautumiseen. Yleinen ERP-kokonaisuuksien kokoon viittaava jako on tehty isojen organisaatioiden ja niin sanottujen SME (eng. *Small and Medium Enterprise*) organisaatioiden välillä. [15; 16.]

Kooltaan ja liikevaihdoltaan isot yritykset tarvitsevat usein toimintaansa toiminnallisuuksiltaan laajoja ERP-järjestelmiä. Isoille yrityksille tarkoitetut järjestelmät ovat usein myös

kustannuksiltaan hintavia. Markkinoiden isoja ERP-järjestelmien toimittajia ovat muun muassa SAP, Oracle sekä Microsoft. [16.]

Keskisuuret yritykset hyötyvät ERP-järjestelmän implementoinneissa erityisesti liiketoimintaprosessien ja yrityksen eri osastojen toimintojen yhtenäistämisestä. Keskisuurten yritysten järjestelmät eivät ole toiminnoiltaan niin laajoja tai raskaita kuin isojen yritysten. Keskisuurille organisaatioille hyviä ERP-vaihtoehtoja tuottavat muun muassa IFS, Netsuite, Epicor, Sage ja Infor. [16; 17.]

Pienille yrityksille ERP-järjestelmien käytöstä on selkeätä hyötyä koko toiminnan kasvattamisessa. Pieniä ERP-järjestelmätoimittajia on markkinoilla paljon, ja tyypillisesti pienille yrityksille suunnitellut järjestelmät ovat toiminnoiltaan kevyitä ja joustavia kokonaisuuksia. Pienille organisaatioille järjestelmiä tuottavia toimijoita ovat muun muassa Workday, Syspro sekä Intacct. Kotimaisia, toimintaan erikoistuneita ERP-järjestelmätuottajia ovat Visma Nova, Lemonsoft sekä RoimaSoftware Lean System [16; 18; 19; 20.]

2.4.2 Toimialakohtaiset ERP-järjestelmät

Osa ERP-järjestelmätoimittajista on erikoistunut myös toimialakohtaisien ERP-järjestelmien toimittamiseen ja päivittämiseen. Toimialakohtaiset ERP-järjestelmät sisältävät kaikki samat ominaisuudet kuin yleiset ERP-järjestelmät. Näiden lisäksi toimialakohtaiset järjestelmät sisältävät tietyille toimialoille räätälöityjä ominaisuuksia. Nämä lisäominaisuudet tuottavat suuria etuja organisaation toimintaan. Toimialakohtaisten järjestelmien kehittäminen lähtökohtaisesti vaatii erityistä tietoutta ja osaamista kyseisestä alasta, joka usein myös nostaa sen hankintakustannuksia. [21.]

Eri ERP-järjestelmätoimittajat ovat erikoistuneet tiettyihin toimialoihin, aina yhdestä toimialasta useampiin. Usein isoilla järjestelmätuottajilla on laajempi valikoima toimialakohtaisesti räätälöityjä ERP-järjestelmiä johtuen heidän markkinaosaamisestaan. Esimerkiksi isolla ERP-järjestelmien tuottaja SAP:lla on erilaisia toimialakohtaisia ERP-järjestelmiä markkinoilla yli 25. Toimialakohtaisia järjestelmiä on muun muassa terveydenhuollossa, turvallisuuden, lentoteollisuuden, logistiikan ja julkisen sektorin aloille. [16; 22.]

2.4.3 ERP-järjestelmien jako palvelutyypin mukaan

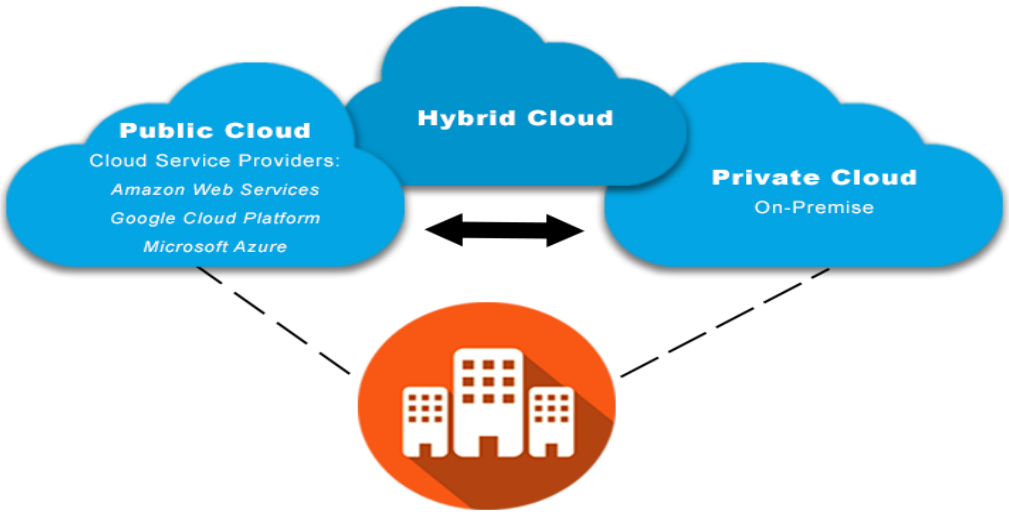
ERP-järjestelmät ovat luonteeltaan järjestelmäsovelluksia. Etenkin isot ja arkkitehtuuriltaan laajat sovellukset tarvitsevat toimiakseen palvelimen, jonka kautta erilaiset toiminnot pyydetään. Palvelimet taas tarvitsevat toimiakseen sille tarkoitetun laitteiston. Yleisesti nämä erilaiset palvelinympäristöt voidaan jakaa kolmeen päätyyppiin: yksityiseen, julkiseen tai näiden kahden hybridiin. [23.]

Yksityisellä, tai paikallisella ERP:llä tarkoitetaan järjestelmäpalvelinta, joka on fyysisesti sijoitettu yrityksen omistamiin tiloihin. Joissakin tapauksissa palvelin voi myös fyysisesti sijaita kolmannen osapuolen tiloissa. Paikalliset ERP-järjestelmät ovat helposti muokattavissa. Monille organisaatioille tällainen skaalautuvuus onkin todella tärkeä osa ERP-järjestelmää. Lisäksi yksityiset ERP-järjestelmät tyypillisesti ovat sijoitusratkaisunsa vuoksi erittäin turvallisia. Paikallisen ERP-järjestelmän hallinta ja vastuu onkin suurimmaksi osaksi sen omistavalla organisaatiolla. Kuitenkin tämä ratkaisu on yleensä hintavampi johtuen palvelinten ja niihin tarvittavien laitteiden kiinteistä ja muuttuvista kuluista. Paikalliset ERP-järjestelmät ovat nykyään yhä vähemmän käytössä hybridi-ERP-järjestelmien muuntautumiskyvyn vuoksi. [23; 24.]

Julkiset tai pilvessä sijaitsevat ERP-järjestelmät ovat vastakohta paikallisiin ERP-järjestelmiin nähden. Julkisia ERP-järjestelmiä operoidaan ja ylläpidetään ulkopuolisen kolmannen osapuolen palvelimilla. Tämä antaa organisaatiolle mahdollisuuden jakaa vastuutaan palveluntarjoajan kanssa, mutta toisaalta antaa palveluntarjoajalle hallinnollisen oikeuden yrityksen erilaisiin tietoihin. Pilvipohjaisten tai niin kutsuttujen SaaS-pohjaisten ERP-ratkaisujen tarjonta on noussut voimakkaasti viimeisen kahdenkymmenen vuoden aikana, ja suurin osa nykypäivän ERP-järjestelmätoimittajista tarjoaa erilaisia pilvijärjestelmävaihtoehtoja. Usein pilvipohjaisia ERP-järjestelmiä pidetään esteettöminä ja helposti käytettävinä. Niiden muokattavuus ja skaalautuvuus on vähäisempää kuin yksityisillä ERP-järjestelmillä. Julkiset ERP-järjestelmät sopivatkin hyvin yrityksille, joilla ei ole erityisempää tarvetta järjestelmän yksilöinnille. [23; 24.]

Kuvassa 3 [25] on esitetty kolmen eri ERP-palvelutyypin eroavaisuuksia. Alhaalla ympyränä on kuvattu ERP-järjestelmää käyttävä organisaatio. Organisaation yläpuolella on esitelty erilaiset palvelutyyppivaihtoehdot pilvinä. Vasemmalla on luonnehdittu pilvipohjainen ERP-järjestelmä ja oikealla paikallinen ERP-järjestelmä. Näiden kahden välissä

on kuvattuna hybridijärjestelmä, jossa hyödynnetään molempien edellä mainittujen järjestelmien parhaita puolia.



Kuva 3. Erilaiset ERP-järjestelmien palvelutyypit. [25]

Hybridi-ERP-järjestelmät ovat paikallisten ja pilvipohjaisten ERP-järjestelmien yhdistelmiä. Järjestelmissä yhdistyy molempien palvelutyyppien parhaat puolet. Hybridi ERP-järjestelmät koostuvat tyypillisesti pilvipohjaisista ERP-ratkaisuista, jotka ovat yhteydessä joukkoon suppeita paikallisia ERP-järjestelmiä. Tämä mahdollistaa näiden kahden palvelutyypin integraation. Näiden järjestelmien etuna on tiedon siirtäminen ja kuljettaminen vaivattomasti, sekä järjestelmän notkeus ja muunneltavuus erityyppisiin käyttötarkoituksiin. Tyypillisesti hybridi ERP-järjestelmiä käyttävät organisaatiot tai yritykset vaativat järjestelmältä erityistä huoltovarmuutta. Hybridi ERP-järjestelmän avulla organisaatiot ovat kykeneviä jatkamaan itsenäisesti toimintaansa, myös mahdollisen laajan häiriötilanteen tapahtuessa. Tyypillisesti organisaatiot tai yritykset, jotka turvaavat yhteiskunnallista toimintakykyä, käyttävät hybridi ERP-järjestelmien paikallisjärjestelmäpuolta kaikkein kriittisimpien toimintojen hallinnoimiseen. Muiden ei niin kriittisten toimintojen hallinnoimiseen voidaan käyttää pilvipohjaista puolta. Tällaisia toimijoita ovat muun muassa energiayhtiöt, valtiolliset toimijat sekä terveydenhuoltoalan suuret toimijat. [23; 26.]

2.4.4 Arkkitehtuurirakenne

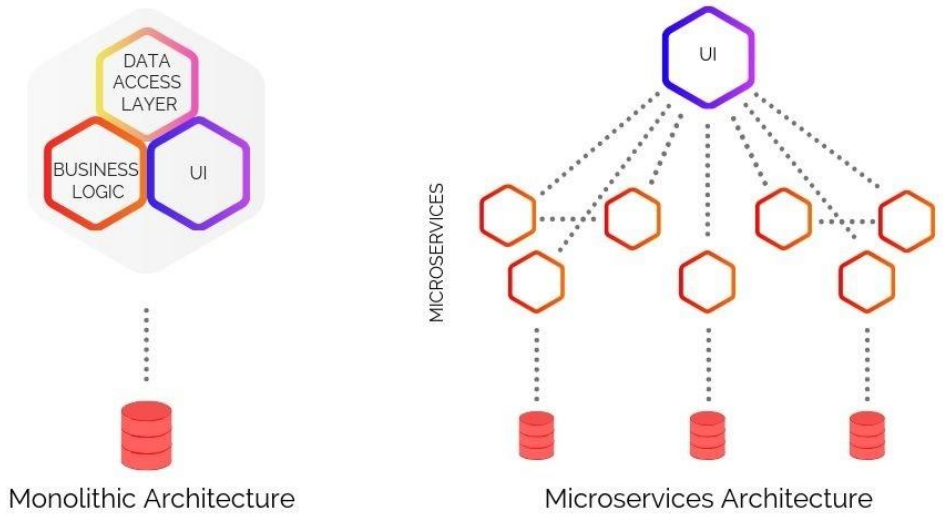
ERP-järjestelmät perustuvat organisaatioiden ydintoimintojen ja prosessien yhdistämiseen yhteen hallinnointijärjestelmään. Näiden järjestelmien tarkoituksena on siis saada paras mahdollinen tehokkuus kaikkiin eri toiminnan prosesseihin. Nykypäivänä eritoten teknologioiden vauhdikas kehitys sekä jatkuva toiminta- ja liiketoimintaprosessien muuttuminen on luonut ERP-järjestelmien arkkitehtuurisille ratkaisuille entistä vaativampia haasteita. Nykyään ERP-järjestelmät voidaankin jakaa arkkitehtuuristen ratkaisujensa pohjalta kahteen eriävään luokkaan: monoliittisiin ja postmoderneihin ERP-järjestelmiin. [27; 28.]

Monoliittisille ERP-järjestelmille on ominaista, että ne ovat tiiviisti yhteen integroituja, eli niiden keskiössä on yksi yhteinen tietokanta ja teknologinen runko, mitä eri järjestelmän osat hyödyntävät. Tämä mahdollistaa yhtenäisen tiedonvälityksen eri järjestelmän osien välillä. Järjestelmät ovat yleisesti optimoitu yrityksen toimintaan ja tarpeisiin sopiviksi. Toiminnan tai liiketoiminnan kasvaessa monoliittisille ERP-järjestelmille on tyypillistä niiden laajennus: uusia liiketoiminta-alueiden prosessityökaluja yhdistetään tai lisätään mukaan jo käytössä olevaan ERP-kokonaisuuteen. Tämä on kuitenkin usein erittäin hintava ja aikaa vievä prosessi. Teknologioiden kehittyessä ja liiketoimintaprosessien laajentuessa monoliittiset ERP-järjestelmät ovat kehittyneet entistä isommiksi ja monimutkaisemmiksi kokonaisuuksiksi. [28; 29; 30.]

Postmodernille ERP-järjestelmälle on tyypillistä, että jokaiselle liiketoiminta-alueelle valitaan siihen sopiva, sen toimintaa ohjaava järjestelmä. Postmoderneissa ERP-järjestelmissä eri moduuleiden prosessit ja tietokannat on hajautettu, ja niiden toimintaa ohjataan keskitetyn integraatioalustan kautta. Ominaista postmodernille ERP-järjestelmillä on sen joustavuus ja muokattavuus. Järjestelmän mahdollisten ominaisuuksien lisääntyessä tai päivittyessä postmodernin ERP-järjestelmän etuna onkin näiden päivitysten vaivaton implementointi, sekä integrointi muihin ERP-järjestelmän moduuleihin. Yleisesti postmodernit järjestelmät jaotellaan kahteen eri alakategoriaan: hallinnollinen ja operatiivinen järjestelmä. Hallinnolliseen järjestelmään kuuluu yleensä vain hallinnollisen puolen moduuleita, kun taas operatiivisiin järjestelmiin liitetään mukaan myös toiminnanohjaamiseen suunnatut moduulit ja työkalut. Postmodernien ERP-järjestelmäratkaisujen kehittä-

misessä on entistä tärkeämpää järjestelmäarkkitehtuurin tarkka määrittely ja toiminta-suunnitelma, sillä aiempaa merkittävämpi osa ERP-järjestelmän sisällöstä siirtyy sen suunnittelu- ja määrittelyvaiheessa järjestelmän tilaajalle. [31; 32; 33.]

Kuvassa 4 [34] on esitettynä sekä monoliittinen arkkitehtuurirakenne että postmoderni mikropalvelutyyppinen arkkitehtuurirakenne. Vasemmalla kuvattu monoliittinen arkkitehtuuri hyödyntää kaikille toiminnoille ja toiminta-alueille yhtenäistä tietokantaa ja teknologista runkoa. Oikealla on kuvattuna postmodernien järjestelmien hyödyntämä mikropalveluarkkitehtuuri, jossa eri palvelut käyttävät hajautettuja tietokantoja. Postmoderneilla ERP-järjestelmillä nämä palvelut ilmenevät usein järjestelmän sisällä erillisinä moduuleina.



Kuva 4. Monoliittisten ja postmodernien järjestelmien arkkitehtuuriratkaisut. [34]

Modernit liiketoiminnan ratkaisut vaativat yhä nopeampaa ja joustavampaa käyttöönottoa sekä moderneja arkkitehtuuriratkaisuja. Tämä on asettanut rajoituksia etenkin monoliittisten järjestelmien käytölle. Postmodernien ERP-järjestelmien ominaisuuksien ansiosta niitä hyödyntävien ja käyttävien organisaatioiden määrä onkin kasvanut viimeisen kahden vuoden aikana huomattavasti. [35; 36; 37.]

3 Ohjelmistotestaus

3.1 Ohjelmistotestaus yleisesti

Ohjelmistotestauksella tai yleisemmin testauksella tarkoitetaan laadunvarmistuksen takaamiseksi suoritettavaa toimintaa, jonka avulla varmistetaan järjestelmän tai tietyn järjestelmän osan laadukkuus, eheys ja toimivuus koko sen elinkaaren aikana. Tarkoituksena on siis selvittää, täyttääkö järjestelmä sille määritellyt vaatimukset. Testaus on erilaisten toimintojen suorittamista etupäässä virheiden, mutta myös järjestelmäaukkojen ja puuttuvien vaatimusten havainnoimiseksi. [38; 39.]

Testaus on elintärkeä osa koko ohjelmiston kehittämistä. Ohjelmiston testauksen avulla myös itse kehittämis- ja ohjelmointiprosessista saadaan eheämpi. Mitä nopeammin järjestelmän kehityksessä huomataan virheitä, sitä nopeammin ja helpommin ne voidaan korjata. Pahimmassa tapauksessa kustannustehokas ja järjestelmällinen testauksen puuttuminen voi johtaa ohjelmiston tai järjestelmän käytön estymiseen. Testauksen avulla voidaan arvioida ohjelman tai järjestelmän

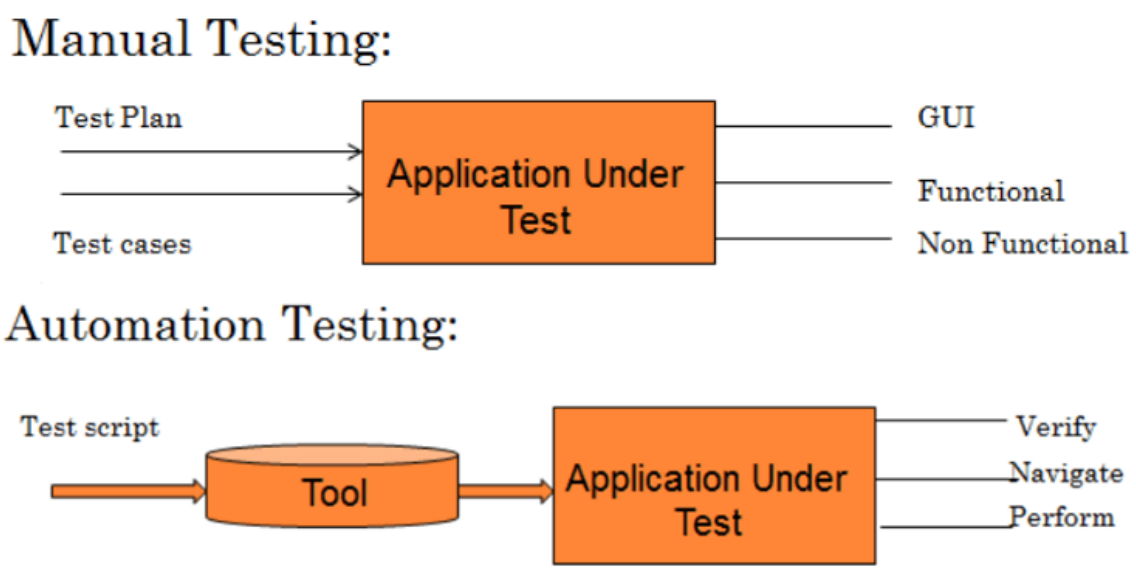
- toimivuutta
- toimintavarmuutta
- tehokkuutta
- käytettävyyttä
- ylläpidettävyyttä
- siirrettävyyttä. [38.]

Ohjelmistotestaus voidaan jakaa kahteen peruskategoriaan: automaatiotestaukseen ja manuaalitestaukseen. Nämä testausmuodot eroavat toisistaan erityisesti testauksen työstämisen näkökulmasta. Kuitenkaan nämä kaksi eri testausmuotoa eivät sulje toisiaan pois, vaan tyypillisesti ne suoritetaan yhdessä maksimaalisen testaustehokkuuden saamiseksi. [40; 41.]

Manuaalinen testaus edustaa ohjelmistotestauksen perinteistä muotoa. Sen avulla ohjelmistoa tai järjestelmää testataan manuaalisesti, eli se on ihmisen tekemää testausta. Manuaalisessa testauksessa itse testausta suorittavan henkilön tiedoilla ja taidoilla on iso merkitys onnistuneen testaustapahtuman takaamiseksi. Tyypillisesti manuaalinen testaaminen vaatii enemmän aikaa ja resursseja, mutta sen avulla voidaan saada testaustuloksia, joita ei automaatiotestauksen avulla kyetä saamaan. Manuaalisen testauksen suorittaminen on myös välttämätöntä, jotta kyetään arvioimaan automaatiotestauksen soveltuvuutta testausprosessiin. [41; 42; 43.]

Automaatiotestauksessa käytetään erilaisia testausta varten suunniteltuja automaatiosovelluksia ja työkaluja, joiden avulla testaustapahtumat suoritetaan. Automaatiotestaus on siis yksinkertaistettuna tietokoneen suorittamaa testausta. Automatisoidut, etukäteen suunnitellut testaustapahtumat, eli skriptit, seuraavat tarkoin määriteltyä testauksen kulkua: todellisten tulosten suhdetta odotettuihin tuloksiin. Automaatiotestausta tehdään erityisesti toistuvien toimintojen ja tapahtumien suorittamiseen, ja sen tarkoituksena on vähentää testauksen toteutumiseen käytettäviä henkilöresursseja sekä aikaa. [41; 44.]

Kuvassa 5 [41] on kuvattu manuaalisen ja automatisoidun testauksen eroja. Ylempänä kuvatussa manuaalisen testauksen kaaviokuvassa on esitetty vasemmalla erilaiset testisuunnitelmat ja testitapahtumat, jotka suoritetaan halutulle ohjelmistolle. Oikealla on esitettynä osa-alueet, joihin testauksella voidaan vaikuttaa. Alempana on kuvailtuna sama kaaviokuva automatisoidun testauksen näkökulmasta. Automatisoidussa testauksessa suoritetaan ennalta määriteltyjä testiskriptejä testaustyökalujen tai ohjelmistojen avulla.



Kuva 5. Manuaalisen testauksen ja automaatiotestauksen erot. [41]

Automatisoidussa testauksessa testaustyökaluilla ja ohjelmistoilla on merkittävä rooli testauksen suorittamisessa. Eri työkalut ja ohjelmistot eheyttävät testausprosessia. Ne auttavat testausdatan ja testitapahtumien luomisessa, erilaisten epäkohtien kirjaamisessa sekä kirjallisessa ja visuaalisessa dokumentoinnissa. Testaustyökaluja on markkinoilla satoja, joista testaajat voivat valita sopivan vaihtoehdon kunkin testausprosessin toteuttamiseen. [44; 45.]

3.2 Testausmenetelmät

Testausprosessit jaetaan tyypillisesti testauksen läpiviemisen suhteen eri metodeiksi sekä lähestymistapojen mukaan eri kategorioihin. Ohjelmistotestausta voidaan suorittaa monesta eri lähtöpisteestä ja näkökulmasta katsoen. Edellä mainittujen metodien ja lähestymistapojen avulla luodaan siis koko testausprosessin pohja. [46.]

3.2.1 Testausmetodit

Testausprosessit ovat usein hyvin toisistaan poikkeavia. Tämän vuoksi erilaisten testausmetodien käyttäminen on suotavaa eri ohjelmistojen testauksissa. Erilaisilla testausmetodeilla saadaan toisistaan poikkeavia testaustuloksia. Testausmetodien käytön vaih-

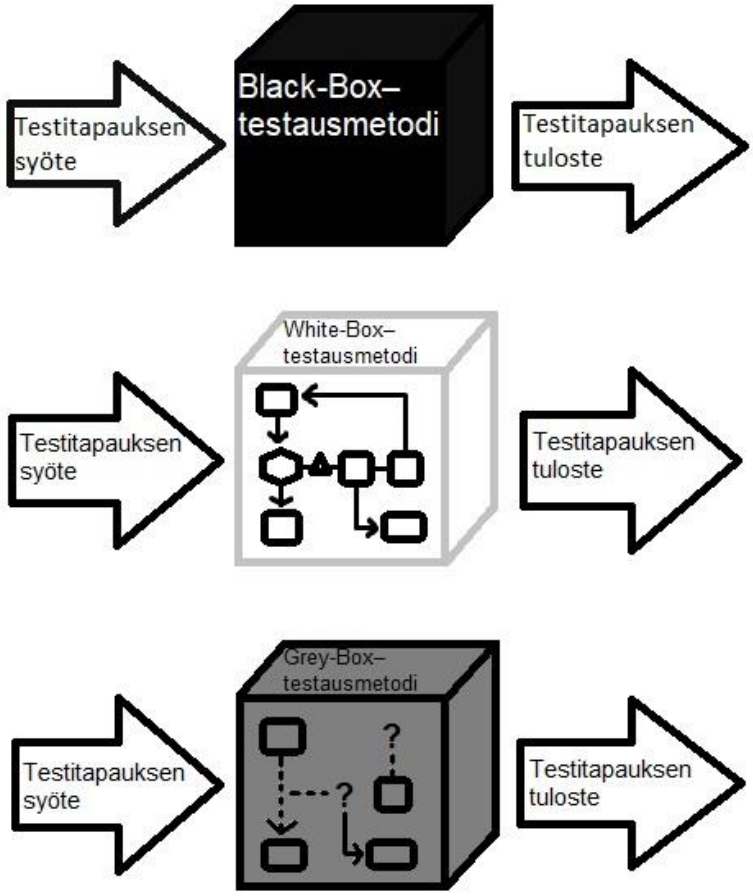
televuuteen vaikuttaa erityisesti testaavien henkilöiden tietotekninen osaaminen ja järjestelmän tai ohjelmiston sisäisen rakenteen asiantuntemus. Kolme käytetyintä ja tunnetuinta ohjelmistotestauksen testausmetodia ovat:

- Black-Box, eli musta laatikko – metodi
- White-Box, eli valkoinen laatikko – metodi
- Grey-Box, eli harmaa laatikko – metodi. [46.]

Black-Box-testausmetodilla tarkoitetaan järjestelmään tai ohjelmistoon tehtävää pintatestausta. Nimi ”Black-Box” viittaa mustaan laatikkoon, missä kuvaannollisesti järjestelmälle tehtävän testauksen sisään testaajat eivät näe. Tätä testausmetodia käyttävillä testaajilla ei tarvitse olla erityistä tuntemusta järjestelmän toiminnasta tai syvemmästä sovellusarkkitehtuurista. Testaavilla henkilöillä ei myöskään tarvitse olla erityistä ohjelmointiosaamista. Black-Box-testausmetodin avulla testataan etupäässä ohjelmiston yleisiä toiminnallisuuksia ja käyttöliittymän käyttökelpoisuutta. Tyypillisesti suoritettavat testaukset tehdään loppukäyttäjän toimintatapoja ja näkökulmaa silmällä pitäen. Black-Box-metodin luonteen vuoksi sitä voidaan käyttää lähes kaiken tasoisissa testausprosesseissa. Black-Box-metodille luodaan usein joukko yksinkertaisia testitapauksia, joiden mukaan testauksesta saatavia tuloksia verrataan oletettuihin tuloksiin. Tämän vuoksi Black-Box-metodi on tehokkaimmillaan käytettynä useille pienille paloille ohjelmistoa ison kokonaisuuden sijasta. [46; 47.]

White-Box-testausmetodi on Black-Box-metodin vastakohta. Testausmetodin avulla pyritään saamaan syvempää analyysia ohjelmiston tai järjestelmän sisäisestä arkkitehtuurista ja sen takaisesta logiikasta. White-Box-metodin käyttäjä tuntee laajasti ja syvällisesti ohjelmiston arkkitehtuurin, sekä omaa hyvät ohjelmointitaidot. White-Box-testausmetodi vaatii enemmän henkilöresursseja, sillä kyseiset testaukset vievät usein kauemmin aikaa kuin muut testausprosessit. Tämän tyyppinen syvällinen järjestelmän testaus vaatii myös erikoistyökaluja tai ohjelmistoja, joiden avulla koko testausprosessi suoritetaan ja dokumentoidaan. Edellä mainittujen syiden vuoksi White-Box-testaus on muihin testausmetodeihin verrattuna kalliimpaa toteuttaa. White-Box-testauksesta saatavat tulokset ovat niin ikään suoraan yhteydessä kirjoitettuun koodiin, joten suoritettavan testauksen virhemarginaali on melko korkea. [46; 48.]

Kuvassa 6 on esitetty kolme toisistaan eriävää testausmetodia näiden nimien mukaisina laatikoina. Ylimpänä on kuvattu mustana laatikkona Black-Box-testausmetodi, jossa ohjelmistolle tehdään pinnallista testausta loppukäyttäjän näkökulmasta. Keskellä on kuvattu valkoinen laatikko eli White-Box-testausmetodi, jossa testaaja tuntee testattavan ohjelmiston syvällisesti. White-Box-testauksessa testataan ohjelmiston syvempää arkkitehtuuria ja sen takaista logiikkaa. Alimpana on kuvattu harmaana laatikkona Grey-Box-testausmetodi. Tämä testausmetodi on Black- ja White-Box-metodien välimuoto, jossa hyödynnetään molempien edellä mainittujen metodien parhaita puolia. [46.]



Kuva 6. Black-, White- ja Grey-Box-testausmetodien toimintaperiaatteet ja erot kuvattuina.

Kahden edellä mainitun metodin välimuotoa kutsutaan Grey-Box-testausmetodiksi. Testausmetodissa hyödynnetään molempien Black- ja White-Box–metodien toimintatapoja sekä poistetaan näiden välisiä puutteita. Metodi näin ollen soveltaa Black- ja White-Box-testauksissa käytettävien tapojen parhaita puolia. Grey-Box-testauksessa testaajalla on

jonkin verran tietoa ohjelmiston sisäisestä arkkitehtuurista ja logiikasta. Tyypillisesti Grey-Box-metodissa testataan kaikkia ohjelmiston eri tasoja, sekä pinnallisia että sisäisiä. Metodilla on myös ominaista käsitellä monimutkaisia järjestelmän osia suoraviivaisella lähestymistavalla. Grey-Box-testausmetodissa poikkeuksellisten testitapausten määrittäminen on isossa roolissa ohjelmiston laadukkuuden ja toimivuuden varmistamisen sekä testausprosessin onnistumisen kannalta. Ominaisin Grey-Box-metodin käyttökohde on integraatiotestaus, jossa testataan ohjelmiston eri osien välisten integraatioiden toimivuutta. [46; 49.]

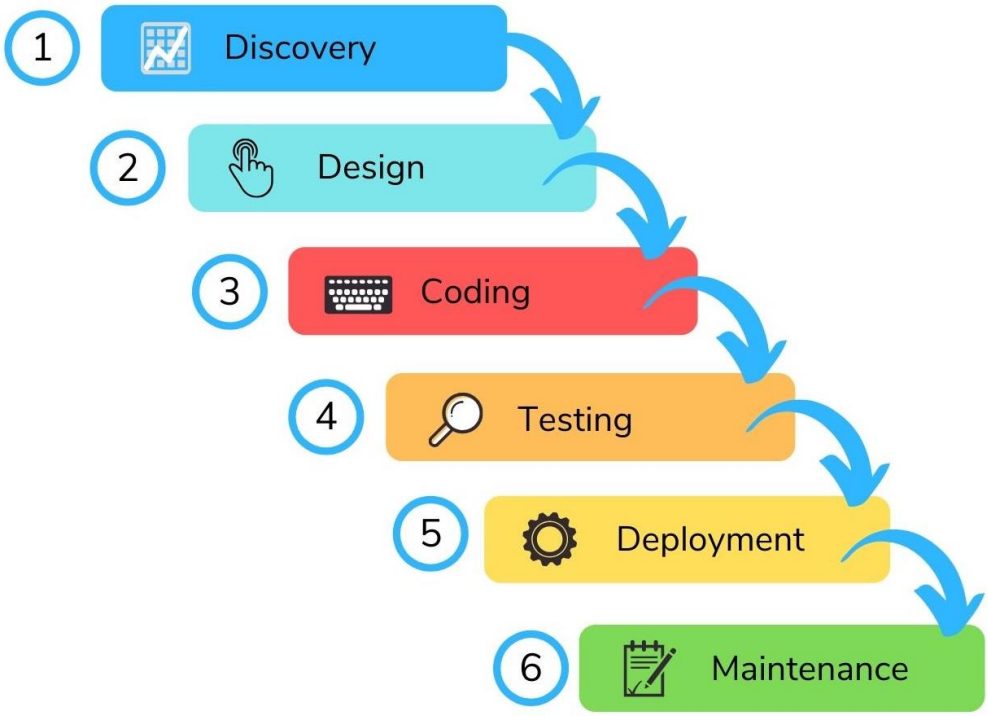
3.2.2 Testauksen lähestymistavat

Tyypillisesti kehittämisprojektia aloitettaessa tulee projektissa mukana olevien jäsenten valita projektille sopiva lähestymistapa. Lähestymistapaa valittaessa päätetään myös, miten kehittämisprojektin työt jakautuvat niin ajallisesti kuin eri tiimin jäsenten välillä. Pääasiallisesti lähestymistapastrategioita on kaksi: reaktiivinen ja ennakoiva, jotka tunnetaan myös englanninkielisillä nimillä ”Waterfall” ja ”Agile”. Molemmilla lähestymistavoilla on omat hyödyt ja haitat projektin läpiviemisen kannalta. Lähestymistavan valintaan vaikuttaa etupäässä projektin luonne, sekä vallitsevat olosuhteet projektin aloitushetkellä. Isoimmat eroavaisuudet näiden kahden lähestymistavan välillä liittyvät testauskertojen määrään sekä testausprosessin ajalliseen sijoittumiseen projektin aikana. [50; 51.]

Perinteisempää testauksen lähestymistapaa kutsutaan reaktiiviseksi menetelmäksi, joka on tunnettu myös englanninkielisellä nimellä ”Waterfall method”, suomennettuna vesiputousmenetelmä. Nimi vesiputous tulee sille ominaisen projektin läpiviemisen tyylistä: vesiputousmenetelmässä projektit jaetaan selvästi tiettyihin vaiheisiin, jotka viedään läpi peräkkäin yksi iso kokonaisuus kerrallaan. Vaiheet seuraavat tarkasti toinen toistaan, ja seuraavaa vaihetta ei voida aloittaa ennen kuin edellinen on viety loppuun. Tyypillisesti reaktiivisessa lähestymistavassa ohjelmiston testaus suoritetaan projektin loppuvaiheessa. Reaktiivinen lähestymistapa jaetaan usein kuuteen tai useampaan erilliseen vaiheeseen. [50.]

Kuvassa 7 [52] on kuvattu reaktiiviselle lähestymistavalle tyypillinen ohjelmiston kehittämisprojektin eteneminen. Kukin vaihe suoritetaan yhtenä isona kokonaisuutena. Ensim-

mäisessä kohdassa on esitetty ideointivaihe. Vaiheessa kaksi usein suunnitellaan ohjelmiston rakenne ja se, miten ohjelmiston halutaan toimivan. Kolmas vaihe on tyypillisesti työvaihe, eli vaihe, jossa rakennetaan ohjelmistolle suunnitellut vaatimukset. Ohjelmiston testaaminen suoritetaan kohdassa neljä, jolloin varmistetaan ohjelmiston toimivuus ja laadukkuus sekä koodipohjalta että loppukäyttäjän näkökulmasta. Tämän jälkeiset vaiheet ovat tyypillisesti valmiin tuotteen tai demoversion julkaiseminen ja tarvittavan ohjelmiston ylläpidon toimeksi paneminen.



Kuva 7. Reaktiivisen lähestymistavan tyypillinen eteneminen projektissa. [52]

Reaktiivisen projektin lähestymistavan hyötyihin kuuluu selkeä suunnitelma ohjelmiston vaatimuksista sekä halutusta projektin lopputuloksesta. Kaikki suurimmat projektiin liittyvät tavoitteet ja ohjelmistoon halutut ominaisuudet päätetään tässä projektin lähestymistavassa jo ennen varsinaista työvaihetta. Tyypillisesti myös projektin jokaisen erillisen vaiheen lopputulos dokumentoidaan laajasti ja tarkasti sekaannusten välttämiseksi projektin myöhemmissä vaiheissa. Reaktiivisessa lähestymistavassa on tapana jakaa työmäärää vaiheiden välillä, joten eri tiimit keskittyvät ainoastaan omaan työvaiheeseensa. Lähestymistavan haitoissa mainitaan usein vähäinen asiakkaan osallistuminen. Joissakin projekteissa on tyypillistä, että asiakas haluaa olla projektissa vahvasti mukana. Täl-

laisiin projekteihin reaktiivinen lähestymistapa voi olla haastava. Myös muutosten tekeminen ohjelmistoon projektin myöhemmässä vaiheessa voi osoittautua vaikeaksi. Kuitenkin isoin haittapuoli reaktiivisen lähestymistavan projekteissa on ohjelmiston testaukseen käytettävä aika. Usein reaktiivisessa lähestymistavassa projektin testausvaihe jää ohueksi, eikä testausta ehditä suorittaa halutulla volyymillä tai resursseilla. [50.]

Toista kehittämisprojekteissa laajasti käytettyä lähestymistapaa kutsutaan ennakoivaksi lähestymistavaksi. Lähestymistavasta käytetään usein myös englanninkielistä nimeä ”Agile method”. Suurin eroavaisuus reaktiivisen ja ennakoivan lähestymistavan välillä liittyy kehittämisprojektin aikaiseen mukautumiskykyyn sekä asiakkaiden ja työntekijöiden osallistumismäärään. Tyypillisesti ennakoivassa lähestymistavassa kehittämisprojekti on jaettu pieniin osiin. Nämä osat, joita kutsutaan myös nimellä sprintit, sisältävät kaikki oleelliset kehittämisprojektin elementit. Kun yksi sprintti on viety loppuun, siitä saadut kehitysehdotukset, testauksen tulokset sekä yleinen palaute käydään läpi. Tämän jälkeen suunnitellaan ja siirrytään seuraavaan sprinttiin, jossa korjataan edellisessä sprintissä ilmenneet ongelmat ja hyödynnetään sen aikana saatu palaute. Sprinttejä suoritetaan niin paljon kuin on tarpeellista laadukkaan ja toimivan ohjelmiston tuomiseksi loppukäyttäjille. [50.]

Kuvassa 8 [53] on kuvattu tyypillinen ennakoivan kehittämisprojektin läpikulku. Projekti jaetaan sprintteihin, jotka suoritetaan yksi kerrallaan. Sprintit koostuvat tyypillisistä kehittämisprojektin elementeistä, joita ovat suunnittelu, työvaihe, testaus, läpikäynti, julkaisu ja ylläpito, sekä mahdolliset jatkokehitysvaiheet. Edeltävästä sprintistä saatu tietosisältö käytetään hyväksi aina seuraavan sprintin suunnittelussa ja läpiviennissä. Sprinttejä on lähes aina kahdesta useampaan kappaletta.

Kuva 8. Ennakoivan lähestymistavan tyypillinen eteneminen projektissa. [53]

Ennakoivan lähestymistavan ehdottomiin hyötyihin kuuluu asiakkaiden tiivis osallistuminen ohjelmiston kehitysprojektiin sekä sen mukautuvuuskyky erilaisiin poikkeaviin tilanteisiin. Kehitysprojekteissa ennakoivan lähestymistavan valinta antaa asiakasyritykselle enemmän kontrollia projektin lopputuloksesta, lähestymistavan jaksottuneisuuden ja yhteistyön ansiosta. Lähestymistavan joustavuuden ansiosta asiakas voi tehdä ohjelmiston haluttuihin ominaisuuksiin muutoksia helpommin. Riski suurempien ohjelmistovirheiden ilmaantumiselle on pieni. Myös lähestymistavalle tyypillisellä laajemmalla ja useammin tehtävällä ohjelmiston testauksella on merkittävä rooli kehittämisprojektin lopputulokseen. Yhtenä haasteena ennakoivassa lähestymistavassa voidaan pitää kommunikaation tasoa. Ennakoivan lähestymistavan projekteissa viestinnän taso ja tärkeys korostuu onnistuneen lopputuloksen takaamiseksi. Isoin haittapuoli ennakoivassa lähestymistavassa on kuitenkin projektiin liittyvä mahdollinen budjetin kasvu ja aikataulun venyminen. Uusien ei-suunniteltujen sprinttien täydentäminen kehittämisprojektiin voi nostaa kokonaiskustannuksia ja venyttää valmiin ohjelmiston toimittamista loppukäyttäjille. [50.]

3.3 Testaustyypit

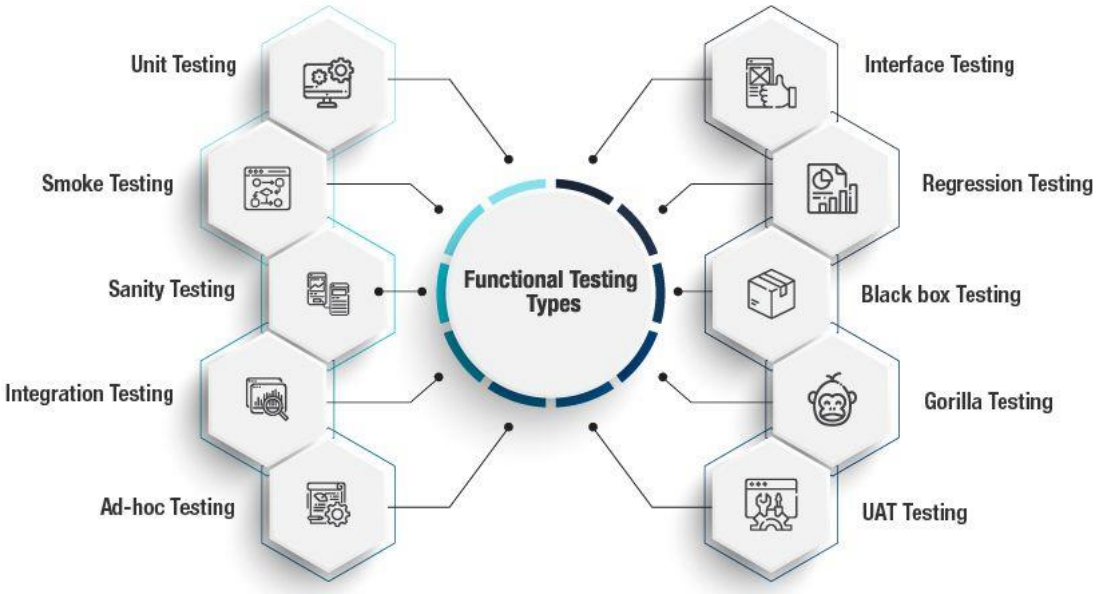
Erilaisten ohjelmistojen testaus vaatii erilaisia variaatioita ja tyylejä testausprosessin onnistumiseksi. Näitä variaatioita kutsutaan testaustyypeiksi. Jokaiselle omalle testaustyypille on määritelty sille ominainen tavoite, strategia sekä suoritustapa. Testaustyyppien avulla siis validoidaan testauksessa oleva ohjelmisto sille määriteltyä päämäärää varten. Testaustyyppejä on eri tasoisia, ja ne tyypillisesti kategorioidaan ylä- ja alatyyppeihin. [54.]

Testaustyyppejä on olemassa yli sata erilaista. Selkeyden ja rajaamisen vuoksi tekstissä käsitellään testaustyypeistä käytetyimmät ja yleisimmät. Itse ERP-järjestelmän testauksen oleellisimmat testaustyypit käsitellään myöhemmissä osioissa. [54.]

3.3.1 Funktionaalinen testaus ja sen alatyyppejä

Funktionaalinen testaus on yläluokan testaustyyppi. Testaustyypissä keskitytään ohjelmiston toiminnallisuuksien testaamiseen. Tarkoituksena on siis testata jokainen ohjelmiston toiminto ja varmistaa, että kaikki toiminnot vastaavat niille asetettuja vaatimuksia. Testauksia voidaan suorittaa käyttöliittymälle, tietokannoille, tietoturvallisuudelle, rajapinnoille, palvelinyhteyksille, liiketoimintaelementeille sekä muille olennaisille toiminnallisuuksille. Tyypillisesti funktionaalisessa testauksessa ei paneuduta syvemmin ohjelmiston tai järjestelmän sisäiseen rakenteeseen tai koodiin. Funktionaalinen testaus voi olla sekä manuaalista että automatisoitua. [55.]

Kuvassa 9 [56] on esitetty funktionaalisen testaustyypin alatyyppejä. Testaustyyppejä, joita funktionaalisessa testauksessa voidaan suorittaa, on paljon. Osaa kuvassa esitetyistä testaustyypeistä voidaan luonnehtia myös metodeiksi. Jokaisen alatyypin avulla paneudutaan tietyn ohjelmiston osa-alueen testaukseen.



Kuva 9. Funktionaalisen testauksen alatyyppejä, joiden avulla testataan tiettyjä ohjelmiston osa-alueita. [56]

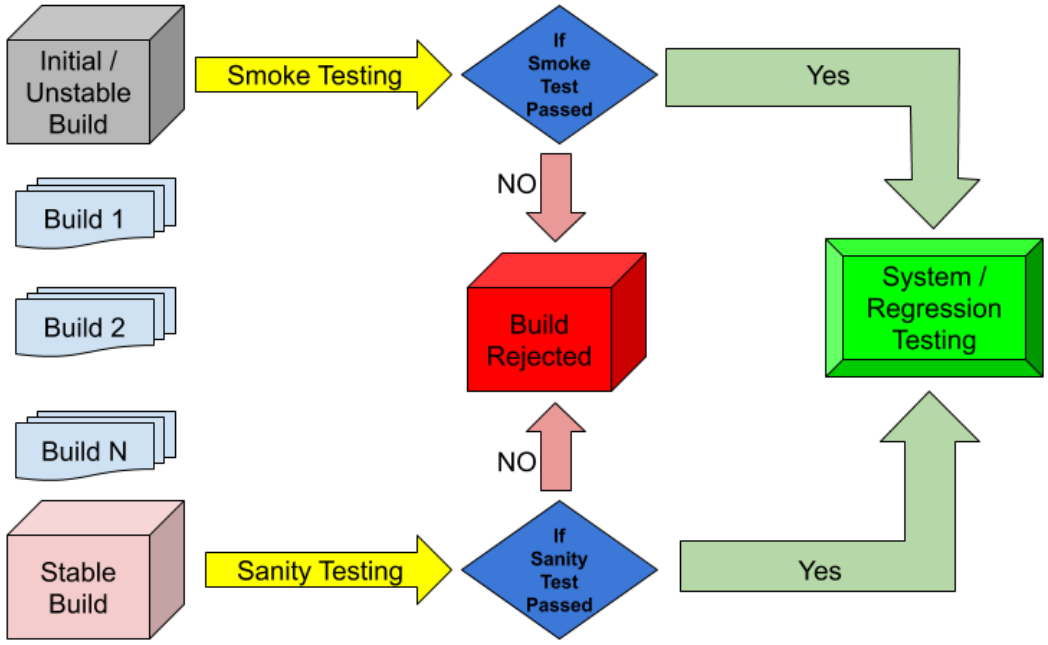
Funktionaalisen testauksen alatyyppejä ovat muun muassa yksikkötestaus, savutestaus, järkevyystestaus, integraatiotestaus, rajapintatestaus ja hyväksymistestaus. Kukin testaustyyppi keskittyy määrättyyn ohjelmiston osan ja version testaamiseen, kuten esimerkiksi integraatiotestaus ohjelmiston sisäisten osien integraation testaukseen. Edellä mainitut testaustyypit käsitellään myöhemmin luvussa laajemmin. [55; 57.]

Yksikkötestaus on ensimmäinen laajemmin käsiteltävistä alatyypeistä. Yksikkötestauksella viitataan testaustyyppiin, jossa ohjelmiston yksittäisiä osia tai soluja testataan niiden toimivuuden osalta. Se on luonteeltaan funktionaalista testausta. Tyypillisesti yksikkötestaus suoritetaan ohjelmoijien toimesta, ennen kuin varsinainen testausprosessi aloitetaan. Testauksen tarkoituksena on varmistaa, että jokainen ohjelmiston rakenneosa toimii niin kuin sen oletetaan toimivan. Yksikkötestaus keskittyy ohjelman pienimpien osien, kuten esimerkiksi ohjelmakoodin, toiminnan testaukseen. Testaukset suoritetaan tarkastelemalla ohjelmiston rakenneosia irrottamalla ne muista rakenneosista. Yksikkötestaus voidaan suorittaa joko manuaalisesti tai automatisoidusti. Yksikkötestauksen avulla ohjelmiston mahdolliset virheet löydetään aikaisessa vaiheessa, joka tehostaa kehittämisprosessia ja helpottaa muita myöhempiä testaus- ja kehittämisprosessin vaiheita. Huolellisesti suoritetulla yksikkötestauksella luodaan myös suotuisa pohja ohjelmiston kehittämisen jatkolle. [58.]

Savu-, järkevyys- ja regressiotestaus käydään läpi samassa jaottelussa, sillä ne liittyvät vahvasti toisiinsa ja ovat tyyleiltään samanlaisia keskenään. Näiden testaustyyppien suoritusajankohdat ovat hyvin lähellä toisiaan, ja näin ollen vaikuttavat toinen toisiinsa vahvasti. Kaikissa testaustyypeissä pyritään arvioimaan ohjelmiston julkaisukelpoisuutta erilaisten muutosten ja lisäysten jälkeen. Savutestauksella, joka on tunnettu myös nimellä koontiversion vahvistustestaus, varmistetaan, että ohjelmiston ydintoiminnot toimivat niille asetettujen vaatimusten mukaisesti. Tarkoituksena ei siis ole paneutua ohjelmiston syvempiin toiminnallisuuksiin tai muihin sivutoimintoihin. Savutestauksen perinpohjaisena tavoitteena on hylätä vakavasti puutteellinen ohjelmisto tai sen osa niin, että testaajat saavat keskittyä oleellisiin testauksiin. Tyypillisesti se suoritetaan testausprosessissa ennen täsmällisempien toiminnallisuustestauksien aloittamista tai silloin, kun ohjelmistosta on toimitettu uusi versio. Järkevyystestaus on luonteeltaan hyvin lähellä savutestausta. Sen tarkoituksena on varmistaa ohjelmiston toimivuus, kun jo olemassa olevaan ohjelmistotuotteeseen lisätään merkittäviä elementtejä tai toimintoja. Erona siis savu- ja järkevyystestauksen välillä on, että savutestaus suoritetaan alustavalle ohjelmistolle, ja järkevyystestaus suoritetaan valmiille ohjelmistolle. Järkevyystestaus toteutetaan vakaalle ohjelmiston versiolle tai koontiversiolle, johon on tehty pieniä koodimuutoksia tai toiminnallisuusmuutoksia. Regressiotestauksessa puolestaan varmistetaan, että ohjelmistoon tehdyt muutokset ovat virhevapaita ja vastaavat ohjelmistolle asetettuja vaatimuksia. Testauksella varmennetaan myös, että tehdyt muutokset eivät vaikuta

muihin ohjelmiston osiin haitallisesti. Savu- ja järkevyystestauksessa hyväksyttyjen muutosten tai lisäysten vaikutukset isompaan ohjelmistokokonaisuuteen näin ollen varmistetaan regressiotestauksen avulla. Tyypillisesti regressiotestaus suoritetaan ohjelmiston vaatimusmäärittelyn muututtua tai ohjelmistoon tehtyjen virhekorjausten jälkeen. [59.]

Kuvassa 10 [59] on kuvattu ohjelmiston kehittämisprosessin aikana tehtävien savu-, järkevyys- ja regressiotestauksen eroavaisuudet, sekä niille tyypillisen läpiviemisen ajankohta ohjelmiston kehittämisen elinkaaressa. Kuvan yläreunassa on kuvattu alustavan tai uudelle ohjelmistolle tehtävän savutestauksen läpivieminen, ja alareunassa vastaavasti jo valmiille ohjelmistolle suoritettavan järkevyystestauksen toteutus. Oikeassa reunassa on esitetty regressiotestaus, jonka avulla vielä testataan tehtyjen muutosten vaikutus koko ohjelmistoon tai ohjelmiston osaan.



Kuva 10. Savu-, järkevyys- ja regressiotestauksen eroavaisuudet ja ilmeneminen ohjelmiston kehittämisprosessissa. [59]

Integraatiotestauksella tarkoitetaan testaustyyppiä, jossa varmistetaan integroidun ohjelmiston toimivuus ja eheys. Ohjelmiston eri osat ovat usein toisiinsa linkitettyjä tai yhdistettyjä. Integraatiotestauksen avulla näiden yhdistettyjen osien toimivuutta testataan, joko keskenään isompana kokonaisuutena, tai kokonaisena järjestelmänä. Tyypillisesti testausta voidaan suorittaa sekä ohjelmistojen pintapuolen integraatioiden toimivuuden

varmistamiseksi että syvempien arkkitehtuuriyhteyksien varmistamiseksi. Integraatiotestaus suoritetaan tavallisesti yksikkötestauksen loppuun viemisen jälkeen. Testauksen avulla voidaan myös varmistaa kahden tai useamman kokonaisen järjestelmän yhteiselo ja toimivuus keskenään. Usein tämän tyyppinen testaus suoritetaan testaamalla eri järjestelmien yksittäisten moduulien toimivuus, ja sen jälkeen moduulien toimivuus yhdessä. Esimerkkinä tällaiselle tapaukselle on ohjelmiston ja sen laitteiston integraatiotestaus. Integraatiotestauksessa käytetään myös erilaisia läpiviemisen tyylejä. Ylhäältä alas -tyylissä ohjelmiston integraatiotestaus suoritetaan kukin ohjelmistotaso kerrallaan, pinnallisesta rakenteesta syviin ohjelmiston rakenteisiin. Tämän tyylin vastakohdassa alhaalta ylös -tyylissä integraatioiden testaus aloitetaan syvistä ohjelmiston rakenteista, ja päätetään pinnallisiin rakenteisiin. Alkuräjähdys-tyylissä kukin erillinen ohjelmiston osion tai moduulin toimivuus testataan yhdessä. [60.]

Yhteyttä, joka yhdistää kaksi tai useampaa eri osaa kutsutaan rajapinnaksi. Rajapintatestaus on ohjelmistotestauksen tyyppi, jossa näiden kahden ohjelmisto-osan välisen kommunikaation toimivuus ja oikeellisuus varmistetaan. Rajapinnat, joihin usein tällaisilla testauksilla viitataan, ovat verkkopalveluja tai ohjelmointi- ja sovellusrajapintoja. Rajapintatestaus jaetaan kahteen testaussegmenttiin. Näitä ovat verkkopalvelujen ja sovelluspalvelinrajapintojen välinen testaus sekä sovelluspalvelinrajapintojen ja tietokantarajapintojen välinen testaus. Rajapintatestauksia suoritetaan muun muassa loppukäyttäjille suotuisan käyttökokemuksen ja käyttäjäystävällisyyden takaamiseksi, turvallisuusvaatimusten varmistamiseksi sekä mahdollisten yhteysvirheiden tarkastamiseksi. Rajapintatestaus voidaan toisaalta jakaa vielä pienempiin tyyppeihin, jotka painottuvat tiettyjen rajapinnan ominaisuuksien testaukseen. Työnkulun testauksella varmistetaan, että rajapinnan koneisto suoriutuu sille syötetystä normaalista työmäärästä. Tämän lisäksi rajapinnan suorituskykyä voidaan testata erilaisilla kuormitustesteillä. [61.]

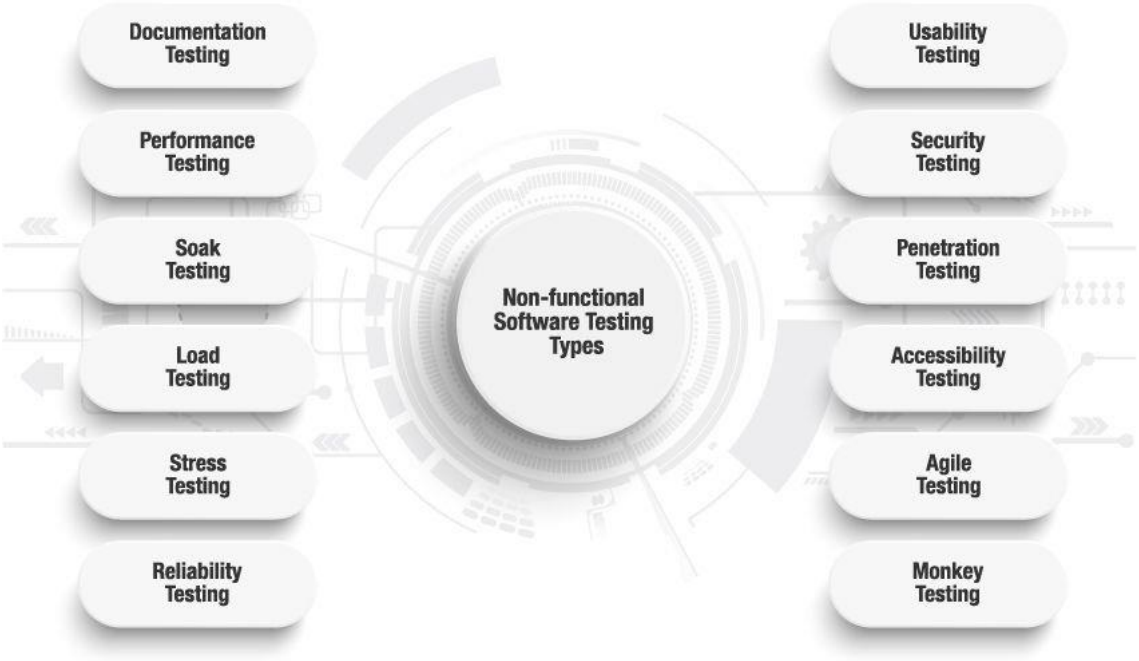
Hyväksymistestaus on testausta, jossa päätetään, onko ohjelmisto täyttänyt sille annetun vaatimusmäärittelyn. Toisin sanoen se on nimensäkin mukaan ohjelmiston lopullisen tuotoksen hyväksymistä. Hyväksymistestaus suoritetaan tyypillisesti juuri ennen kuin ohjelmisto viedään tuotantoympäristöön, eli varsinaiseen käyttöön. Testauksen päätarkoituksena on arvioida ohjelmiston käyttöä liiketoimintaprosessien ja loppukäyttäjien näkökulmasta. Tämän lisäksi testauksen avulla asiakkaalle annetaan mahdollisuus antaa mahdollisia viime hetken palautteita ja muutosehdotuksia ohjelmasta. Hyväksymistes-

taus jaetaankin eri näkökulmien puolesta neljään erilliseen kategoriaan: käyttäjien hyväksyntätestaukseen, liiketoimintahyväksyntätestaukseen sekä Alpha- ja Beta-testaukseen. Käyttäjien hyväksyntätestaus on usein asiakkaan puolesta toteutettua testausta. Liiketoimintahyväksymistestaus on liiketoimintaprosessin onnistumisen arvioimista, jonka usein suorittavat asiakasyrityksen liiketoiminnan ammattilaiset. Alpha-testauksella viitataan ohjelmiston testausvaiheeseen, jossa vielä viimeisiä varsinaisia virheitä korjataan, ennen ohjelmiston julkaisemista loppukäyttäjille. Beta-testauksella tarkoitetaan testausta, joka suoritetaan organisaation ulkopuolisten loppukäyttäjien toimesta. [62; 63; 64.]

3.3.2 Ei-funktionaalinen testaus ja sen alatyyppejä

Funktionaalisen testauksen vastakohta on ei-funktionaalinen testaus. Sillä viitataan testauksen ylätyyppiin, jossa testataan, kuinka hyvin ohjelma toimii. Toisin sanoen, kun funktionaalisessa testauksessa testataan ohjelmiston toiminnallisuuksia, niin ei-funktionaalisessa testauksessa testataan näiden eri toiminnallisuuksien suorituskykyä erilaisissa tilanteissa ja ympäristöissä. Tyypillisesti laadukkaassa ja laajassa ohjelmiston testauksessa hyödynnetään molempia testaustyyppejä. Tämän avulla ohjelmiston laadukkuus ja toiminta voidaan varmistaa myös eriävissä toimintaympäristöissä. [65.]

Kuvassa 11 [56] on kuvattuna ei-funktionaalinen testaus ylätyyppinä sekä sen tyypillisiä alatyyppejä. Kaikilla ei-funktionaalisen testauksen alatyypeillä pyritään arvioimaan ja varmistamaan ohjelmiston osien toimintavarmuutta erilaisissa tilanteissa ja toimintaympäristöissä. Osaa kuvassa esitetyistä alatyypeistä voidaan myös luonnehtia lähestymistavoiksi tai testausmenetelmiksi.



Kuva 11. Ei-funktionaalinen testaus, sekä sen tyypillisiä alatyyppejä. Jokaisen avulla testataan ohjelmiston tietyn osan toimintavarmuutta. [56]

Ei-funktionaalisen testauksen alatyyppeihin kuuluu muun muassa toimintakykytestaus, käytettävyystestaus sekä turvallisuustestaus. Testaustyyppien avulla pyritään arvioimaan ja varmistamaan ohjelmiston eri elementtien toimintavarmuutta ja suorituskykyä. Edellä mainitut testaustyypit ovat yleisimmin käytettyjä, sekä tärkeimpiä ohjelmiston onnistuneen lopputuloksen kannalta. Nämä testaustyypit käydään laajemmin läpi tekstissä. [65.]

Toimintakykytestaus on ohjelmistotestauksessa käytettävä tyyppi, johon kuuluu useita eri ohjelmiston toimintakyvyn arviointiin liittyviä testauksia. Testauksen tarkoituksena ei siis ole löytää virheitä ohjelmistosta tai sen suorituskyvystä, vaan pikemminkin testata ohjelmiston suoriutumista erilaisista työmääristä ja kuormasta. Toimintakykytestauksesta saadaan arvokasta diagnostiikkatietoa, jota voidaan edelleen hyödyntää havaittujen puutteiden korjaamiseen tai parantamiseen. Toimintakykytestauksen avulla voidaan niin ikään helposti kertoa, mikäli ohjelmiston tai sen laitteiston asetukset tai vaatimukset

eivät ole riittävät toimintojen suorittamiseen. Käytetyimpiä toimintakykytestejä, joita ohjelmistolle tai järjestelmälle voidaan tehdä, ovat kuormitustestaus, stressitestaus, kestävyystestaus sekä volyymitestaus. Kuormitustestauksella viitataan järjestelmän normaalin, mutta suhteessa kasvavan, kuormituksen asettamista ohjelmistolle ja ohjelmiston suoriutumisen arvioimista tässä tilanteessa. Stressitestaus kuuluu muuten samaan kategoriaan kuormitustestauksen kanssa, mutta siinä kuormituksen määrä viedään ohjelmiston normaalin sietokyvyn yli. Stressitestauksen tarkoituksena on arvioida, milloin ohjelmisto kaatuu, sekä miten ohjelmiston palautumista ongelmatilanteesta. Kestävyystestauksella arvioidaan ohjelmiston suoriutumista normaalista kuormitusmäärästä pitkällä aikavälillä. Tämän testauksen tarkoituksena on varmistaa, että ohjelmistossa ei tapahdu mahdollisia muistivuotoja. Volyymitestauksen tarkoituksena on kyetä arvioimaan ohjelmiston suoriutumista, kun siihen tuodaan isoja määriä erilaista dataa. [66.]

Käytettävyystestauksella tarkoitetaan testaustyyppiä, jossa pyritään löytämään mahdollisia virheitä sekä parantamismahdollisuuksia ohjelmiston käyttöliittymästä. Tämän lisäksi käytettävyystestauksella saadaan arvokasta tietoa loppukäyttäjien toimintatavoista ohjelmistoa käytettäessä. Usein käytettävyystestaus muodostuu kolmesta pääelementistä: fasilitaattorista, tehtävistä ja osanottajista. Fasilitaattori on henkilö, joka toimii koko käytettävyystestausprosessin ajan osanottajien ja muiden osallisten apuna sekä ohjelmiston asiantuntijana. Fasilitaattorin pääasiallinen toimenkuva on avustaa käytössä ja vastata ohjelmistoon liittyviin kysymyksiin vaikuttamatta osanottajien toimintaan tai mielipiteisiin ohjelmistosta. Tehtävät ovat ohjelmiston käyttöön liittyviä todenmukaisia aktiviteetteja. Ne voivat olla luonteeltaan joko tarkasti määriteltyjä tai avoimia. Tehtävien dokumentaation tulee olla hyvin selkeä, jotta testausprosessi olisi onnistunut, ja saadut testitulokset hyödyllisiä ja luotettavia. Käytettävyystestauksen osanottajiksi valitaan tyypillisesti ohjelmiston loppukäyttäjiä. Tämän avulla taataan todenmukaiset testaustulokset. Usein osallistujia pyydetään myös kertomaan testausprosessin aikana ajatusmaailmastaan, sekä valinnoistaan mahdollisimman tarkasti. Tämän avulla saadaan mahdollisimman kattava käsitys loppukäyttäjien käyttäytymisestä, ajatuksista, motivaatiosta ja tavoitteista. [67.]

Turvallisuustestauksella viitataan ohjelmistotestauksen tyyppiin, jossa pyritään löytämään ohjelmistosta mahdollisia haavoittuvaisuuksia, joita väärinkäyttäjät voivat käyttää hyödyksi tietomurtoihin. Turvallisuustestauksen avulla pyritään ennen kaikkea minimoimaan tietoturvaan ja tietosuojaan liittyviä heikkouksia mahdollisimman aikaisessa vaiheessa ohjelmiston kehitystä. Turvallisuustestauksessa käytetään kuuden pääperiaatteen mallia. Nämä kuusi pääperiaatetta ovat luottamuksellisuus, koskemattomuus, todentaminen, oikeuttaminen, saatavuus, sekä kieltämättömyys. Nämä kaikki pitää ottaa huomioon ohjelmiston kehityksessä ja testauksessa. Isoja kokonaisuuksia turvallisuustestauksessa ovat verkon turvallisuus, järjestelmäohjelmiston turvallisuus, asiakaspään ohjelman turvallisuus sekä palvelinpään ohjelman turvallisuus. Tyypillisesti turvallisuustestaus jaetaan pienempiin kokonaisuuksiin, jotka keskittyvät ohjelmiston turvallisuuden eri osa-alueisiin. Näitä kokonaisuuksia ovat muun muassa haavoittuvuus- ja turvallisuusskannaus, tunkeutumistestaus sekä turvallisuustarkastus. Haavoittuvuus- ja turvallisuusskannauksen avulla käydään ohjelmisto kokonaan läpi mahdollisten turvallisuusriskien varalta. Usein haavoittuvuustestaus suoritetaan automatisoidusti ja siinä käydään ohjelmisto läpi tunnettujen haavoittuvuusmallien löytämiseksi. Tunkeutumistestauksen tarkoituksena on simuloida oikeata tietoturvahyökkäystä ohjelmistoon. Testauksessa käydään läpi ohjelmiston mahdollisia heikkouksia, jonka jälkeen niitä yritetään käyttää hyväksi ohjelmiston tietomurtosimulaatiossa. Turvallisuustarkastuksella tyypillisesti tarkoitetaan ohjelmiston sisäistä tarkastelua turvallisuuspuutteiden löytämiseksi. Usein turvallisuustarkastuksen jälkeen suoritetaan löydettyjen riskien arvioiminen, jossa tarkastellaan löydettyjen turvallisuusaukkojen vakavuutta. [68.]

3.3.3 Tutkiva testaus

Tutkivaa testausta voidaan luonnehtia usein eri tavoin, mutta usein se luokitellaan testaustyypiksi. Tutkivaa testausta voidaan kuvailla ohjelmiston opetteluksi, testien suunnitteluksi ja testauksen suorittamiseksi samanaikaisesti. Testausprosessi keskittyy epäkohtien havaitsemiseen ja luottaa yksittäisen testaajan kykyihin löytää uniikkeja tapoja testata ohjelmistoa, ja näin ollen löytää mahdollisia virheitä ja puutteita perinteisten testaustyyppien skaalan ulkopuolelta. Testausprosessissa ei siis seurata ennalta määrättyjä testitapauksia. Tutkivan testauksen avulla testausprosessiin saadaan uusi ulottuvuus jäsentyneen testauksen lisäksi. Testaus on hyvin satunnaista ja epämuodollista, joka voi paljastaa virheitä, joita ei jäsennellyssä testauksessa huomattaisi. [69.]

3.3.4 Mobiilitestaus

Mobiilitestaus on nimensä mukaan mobiilisovellukselle tehtävää testausta. Mobiilisovellusten testauksessa käytetään hyödyksi samoja ohjelmistotestauksen toimintatapoja, menetelmiä ja tyyppejä, joita muidenkin ohjelmistojen testauksessa, mutta isoimmat erot mobiilitestauksessa tulevat esille mobiililaitteille tyypillisten elementtien välillä. Mobiililaitteille tyypillisiä yksilöiviä elementtejä ovat muun muassa:

- erilaiset mobiililaitteet ja niiden käyttöliittymät
- mobiililaitteiden vaihtelevat komponentit
- eri kokoiset näytöt
- puhelutoiminto
- verkkoyhteyksien käyttö, kuten 3G, 4G ja Wi-fi
- tiedonsiirtonopeuksien vaihtelu. [70]

Kaikki nämä yksityiskohdat täytyy huomioida mobiilisovellusta kehitettäessä ja testattaessa, jotta ohjelmiston laadukkuus ja eheys voidaan taata. Mobiilitestauksessa käytetään niin ikään monia hyödyllisiä työkaluja ja ohjelmistoja. Näiden apuvälineiden avulla pystytään muun muassa imitoimaan erilaisia käyttöliittymiä ja mobiililaitteiden kokoonpanoja. [70.]

Tämän lisäksi myös mobiilille tehtävän ohjelmiston tyypillä on suuri merkitys testausprosessille. Nämä mobiilisovellustyypit jaetaan kolmeen päätyyppiin, jotka ovat mobiiliverkkosovellus, paikallissovellus sekä näiden kahden hybridi. Mobiiliverkkosovellus toimii puhelimen selaimen kautta, eikä tämän vuoksi tarvitse erillistä asentamista. Verkkosovelluksen huono puoli on se, että sitä ei voi käyttää, mikäli verkkoyhteyttä ei ole, tai yhteys on liian heikko. Paikallissovellus on puolestaan ainoastaan yhdelle tietylle käyttöliittymälle kehitetty sovellus. Paikallissovellusten erityinen hyöty on, että ne voivat käyttää kaikkia mobiililaitteiden toimintoja hyväkseen. Ongelmana tässä ovat kuitenkin sen kor-

keat kehitys ja ylläpitokustannukset. Hybridisovelluksella viitataan kahden edellä mainitun yhdistelmäsovellukseen. Tyypillisesti se esiintyy paikallissovelluksen muodossa, mutta hyödyntää myös selaimen toiminnallisuuksia. Hybridisovelluksen etuina ovat sen edullisemmat kehityskustannukset suhteessa paikallissovelluksiin ja sen helppo jakaminen loppukäyttäjille. Yleensä hybridisovelluksen haittapuolena on sen hitaus sekä käyttöliittymän ja mobiililaitteen huonompi yhteistyö. [70.]

3.4 Testausdokumentaatio

Testausprosessin aikana tuotettu dokumentaatio ja sen laatu on erittäin tärkeätä. Testausdokumentaation avulla saadaan hyvä määritelmä siitä mitä ollaan testaamassa, miten testataan, sekä missä järjestyksessä testaukset suoritetaan. Tehdystä työstä jää selkeä validointi, jota osalliset voivat myöhemmin tarkastella. Näiden lisäksi testauksen aikana saadaan paljon erilaista tietoa testausten kulusta, sekä mahdollisista virheistä, joita testausprosessin aikana ilmenee. Dokumentaatio auttaa myös ymmärtämään koko ohjelmiston kehitysprosessin edistymistä. [71.]

Testaussuunnitelma on laaja kuvaus tulevasta testausprosessista. Dokumentin alkuun hahmotellaan projektin yleinen kuvaus sekä kaikki testattavaan ohjelmistoon liittyvät yksityiskohdat. Usein myös ohjelmiston testattavat osat ilmoitetaan sekä testausprosessin aloituskriteerit hahmotellaan. Tämän jälkeen testaussuunnitelmassa kuvataan testausprosessin läpivienti, aina käytettävistä lähestymistavoista, menetelmistä ja tyypeistä testausprosessin lopetuskriteereihin. Testauksen alustava aikataulu pyritään myös arvioimaan. Tämän avulla annetaan organisaatiolle arvio siitä, milloin ohjelmiston lopullinen käyttöönotto tapahtuu. [72.]

Bugiraportti on testausprosessin aikana tärkein yksittäinen kirjallinen dokumentaatio. Bugiraportin avulla testausprosessin aikana todettu virhe kirjataan tarkasti ylös. Mitä kattavampi ja selkeämpi bugiraportti on, sitä helpompi virhe on löytää ja korjata. Tyypillisesti hyvään bugiraporttiin kirjataan ylös tarkka tekninen kuvaus löydetystä virheestä ja sen tapahtumisympäristöstä, löydetyn virheen vakavuusaste ja korjauksen tärkeysjärjestys, virheen uudelleenluontiohjeet sekä tapahtuman todellinen tulos suhteessa odotettuun tulokseen. Bugiraporteista korkeimman vakavuusasteen virheet korjataan ensimmäisten

joukossa, sillä ne voivat estää kokonaan ohjelmiston käytön, tai vaikuttaa sen käyttökokemukseen negatiivisesti. Vakavuusasteeltaan matalat virheet voidaan korjata myös ohjelmiston julkaisemisen jälkeen, mikäli niillä ei ole merkittävää vaikutusta ohjelmiston käyttökokemukseen. [72.]

Testitapaukset ja testiskenaariot ovat testausprosessin alkuun luotavia testausdokumentteja, joita seuraamalla kukin testi viedään läpi. Näiden luomiseen vaaditaan ohjelmiston sen aikaisen version analysoimista, jotta tiedetään, miten kunkin elementin ja toiminnon tulisi toimia. Analysoinnin pohjalta luodaan testiskenaariot, jotka ovat usein tiiviitä yhden lauseen kuvauksia testattavasta kokonaisuudesta. Esimerkkinä tällaiselle on ”Kirjautuminen järjestelmään”. Testiskenaarioiden sisälle luodaan joukko testitapauksia. Testitapaukset ovat pieniä yksinkertaisia askeleita, jotka suoritetaan peräkkäin. Testitapauksia voidaan luonnehtia malleiksi, joita tulee seurata, jotta testiskenaario saadaan vietyä läpi onnistuneesti. Tyypillisesti testitapauksissa seurataan kolmea pääelementtiä jotka ovat toimenpide, tapahtuman oletettu tulos ja tapahtuman todellinen tulos. [72.]

4 Toiminnallisuuksien testaus toimeksiantajan yrityksessä

4.1 ERP-testauksen taustaa

Opinnäytetyön toimeksiantajan yrityksessä ERP-järjestelmän tekniselle testaukselle oli erityistä tarvetta, etenkin henkilöresurssien puitteissa. Yrityksen kasvaessa myös järjestelmän laajuuden, laadun ja eheyden täytyy pysyä ajantasaisena jatkuvan kehityksen takaamiseksi. Myöhemmin tekstissä käsiteltäviin testauksiin toimeksiantajayritys tarvitsi varsinkin lisää henkilöstöä ohjelmiston testaukseen, sillä ne olivat kooltaan laajempia kehitysprojekteja. Kehitysprojektin tarkoituksena on luoda järjestelmää käyttäville asiakkaille ja ammattilaisille entistä helppokäyttöisempi, saavutettavampi ja turvallisempi alusta.

ERP-järjestelmää pääsääntöisesti käyttävät organisaation sisällä terveydenhuollon asiantuntijat, kuten lääkärit ja hoitajat. Myös markkinoinnin ja myynnin ammattilaiset käyttävät ERP-järjestelmää jokapäiväisessä toiminnassaan. Yrityksen ulkopuolella järjestelmää käyttävät pääosin yksityisasiakkaat sekä organisaation asiakasyritykset.

Toimeksiantajayrityksen ERP-järjestelmä on sekä asiakkaiden että alan ammattilaisten toiminnanohjaamista helpottava järjestelmä. ERP-sovellukseen sisältyy ajanvarausjärjestelmä, asiointijärjestelmä, hoitoprosessin hallinnointijärjestelmä, sekä markkinoinninhallintajärjestelmä. ERP-järjestelmässä on kaikki oleelliset asiakkaiden toimintaa helpottavat toiminnallisuudet sekä ammattilaisten jokapäiväistä työtä parantavat ominaisuudet.

Toimeksiantajan ERP-kokonaisuus on rakennettu hyväksikäyttäen modernia low-code ohjelmointityökalua. Low-codella viitataan visuaaliseen ohjelmointikieleen. Low-code työkaluissa kehittämistyö tehdään valtaosin visuaalisessa käyttöliittymässä, jossa low-code-syntaksi tuottaa halutun ohjelmakoodin taustalla samanaikaisesti. Ohjelmoitavat kokonaisuudet luodaan erilaisten logiikkakaavioiden ja toisiinsa linkittyvien ohjelmointielementtien avulla. Low-code-ohjelmointityökalun avulla myös erilaiset integraatio-ohjelmiston eri osien välillä luodaan kehittämistyön ohessa. Low-coden ehdottomana etuna on kehitysprojektien nopeutuminen. Tämän kautta myös kehitettävien ohjelmistojen testaus- ja palauteprosessit tehostuvat huomattavasti. Muuttuvien asiakastarpeiden ja toimintaympäristöjen vuoksi low-code on myös hyvä ohjelmointiratkaisu sen sopeutumiskyvyn ansiosta. [73.]

Toimeksiantajan yrityksessä kehitystyökaluksi on valikoitunut low-code-ohjelmointityökalu juuri sen loogisuuden ja helposti ymmärrettävyyden vuoksi. Low-coden visuaalinen ja selkeä käyttöliittymä antaa mahdollisuuden myös muulle henkilöstölle ymmärtää ohjelmointiratkaisujen takaista logiikkaa. Kuitenkin ennen kaikkea sen avulla organisaation sisäiset eri toimijat pystyvät muodostamaan helpommin kokonaiskuvan toimintojen teknisestä puolesta. Toimeksiantajan yrityksen kehitystiimi käyttääkin low-code-ohjelmointityökalua nimenomaan erilaisiin ERP-järjestelmän pienkehitysprojekteihin. Tavallisesti suurempiin ja monimutkaisempiin kehittämisprojekteihin käytetään kolmannen osapuolen tukea.

Kuvassa 12 on esitetty tyypillisen low-code-ohjelmointialustan käyttöliittymää. Low-code-alustassa iso osa ohjelmoinnista tapahtuu graafisen käyttöliittymän avulla. Visuaalisuuden ja selkeytensä puolesta uusien ohjelmien kehittäminen sekä vanhojen ohjelmien päivittäminen on nopeaa ja kustannustehokasta. Low-coden käyttöön on niin ikään helpompaa kouluttaa uusia työntekijöitä, sillä se ei vaadi koulutettavalta henkilöltä erityistä ohjelmointitaustaa. Kuvassa esitetyssä logiikkakaaviossa on kuvattuna jonkun tietyn toiminnallisuuden tai tapahtuman läpivienti ohjelmointitasolla. Toiminnon alkuun määritellään lähtökohdat ja vaadittavat syötteet. Tämän jälkeen toimintoon voidaan luoda erilaisia ehtoja, joiden avulla toimintoa ohjataan. Tällaisia voivat olla esimerkiksi erilaiset vaatimuspykälät, kuten ”jos” ja ”tai”. Myös erilaisia tietokantakomentoja voidaan suorittaa käyttöliittymässä. Lopuksi vielä koodissa määritellään toiminnon lopetusehdot, eli se, miten kyseinen toiminto päättyy. Low-code luo automaattisesti ohjelmakoodin visuaalisen käyttöliittymän pohjalta.

Kuva 12. Tyypillinen low-code-ohjelmointialustan visuaalinen logiikkakaavio.

Yrityksen ERP-järjestelmän historia on hyvin tyypillinen nykypäivän yritysmaailmassa: kasvava yritys on ottanut toimintansa parantamiseksi ja helpottamiseksi käyttöön toiminnoiltaan kevyen ERP-järjestelmän, joka on yrityksen kasvun ohessa laajentunut toiminnoiltaan mittavaksi ja perusteelliseksi koko organisaation toimintaa ohjaavaksi järjestelmäksi. Yrityksen kasvun tärkeänä edellytyksenä on, että sen toimintaa ohjaava ERP-järjestelmä on toiminnaltaan luotettava ja turvallinen. Tästä johtuen myös järjestelmän jatkuva testaaminen ja laadunvarmistus on tärkeässä roolissa kasvun turvaamiseksi.

Testattava ERP-järjestelmä on arkkitehtuuriratkaisultaan postmoderni. Siinä useat eri toiminnalliset kokonaisuudet, kuten esimerkiksi ajanvarausjärjestelmän tai hoitoprosessin hallintajärjestelmän toiminta ja tietokannat, ovat hajautettuina erillisiin itsenäisiin moduuleihin. Näitä ohjataan integroidun alustan avulla. ERP-kokonaisuuden kukin itsenäinen moduuli voidaan näin ollen luonnehtia erilliseksi mikropalveluksi.

4.2 Potilaalle tarkoitetun verkkomobiilisovelluksen testaus

4.2.1 Testausprosessin lähtökohdat

Opinnäytetyössä toteutettiin yksi iso projekti. Kyseinen kehitysprojekti aloitettiin yhdessä kolmannen osapuolen tuella. Projektin tarkoituksena oli luoda mobiilisovellus, jolla potilaat pääsisivät helpommin käyttämään yrityksen tarjoamia palveluita. Samanaikaisesti potilaille tarkoitettu vanha asiointijärjestelmä päivitettiin. Asiointijärjestelmä yhtenäistettiin mobiilisovelluksen kanssa yhdeksi eheäksi järjestelmäksi. Kehitettävä mobiilisovellus on toiminnoiltaan tiiviisti yhteydessä muihin ERP-järjestelmän osiin. Koska kehitysprojektissa luotiin vanhojen moduulien ja toiminnallisuuksien pohjalta uusi verkkomobiilisovellus, valikoitui parhaaksi projektin lähestymistavaksi ennakoiva ”Agile”-käytäntö. Koko projektin aikana sekä ohjelmointitiimi, testaustiimi että muut projektin osalliset olivat tiiviisti yhteydessä toisiinsa.

Projektin aluksi luotiin karkea suunnitelma ohjelmiston vaatimuksista. Erilaiset toiminnot ja niiden käyttäytyminen ohjelmassa kuvattiin alustavasti. Myös ohjelman graafinen ulkoasu ja tiedon esiintymistapa luonnehdittiin projektin suunnitelmaan. Testaussuunnitelma oli melko kevyt kuvaus tulevasta testausprosessista, sillä testaus suoritettiin valtaosin organisaation sisällä. Kuitenkin testiskenaariot ja tapaukset luotiin laajoiksi ja kattaviksi dokumenteiksi testausprosessin alkuun. Testiskenaariot ja tapaukset dokumentoitiin tarkistuskorttipohjaiseen projektinhallintajärjestelmään. Testiskenaarioita ja tapauksia seurattiin läpi testausprosessin, jotta ohjelman versiointia ja toimivuutta pystyttiin tarkasti valvomaan. Projektin alkuun päätettiin myös, että suoritettavat ohjelmiston testaukset tehdään manuaalisesti, sillä erityisempää tarvetta automaatiotestaukselle ei ollut.

Valtaosa kehitysprosessin testauksesta suoritettiin organisaation sisäisen testaustiimin puolesta. Testausprosessin aikana käytettiin useita eri testiympäristöjä testausten suorittamiseen. Ohjelmoijat suorittivat ohjelman eri koontiversioiden yksikkötestaukset White-Box-testauksina ennen testaustiimin laajempaa toiminnallisuuksien testausta. Laajempi toiminnallisuuksien testaus suoritettiin joko Black- tai Grey-Box-menetelminä. Eri menetelmiä käyttäen päivitettävän ohjelman koko testausskaala pystyttiin hyödyntämään, aina teknisen puolen testitapauksista käyttöliittymän visuaalisuuden validointiin.

Mobiilisovelluksen testauksessa tuli ottaa huomioon mobiililaitteille ominaiset seikat. Jo testausprosessin alussa päätettiin testaustiimin jäsenten vastuualueet käyttöliittymien ja verkkoselainten testauksen suhteen. Valtaosa tiimin jäsenistä käytti testauksessa iOS (*Apple-puhelinvalmistajan käyttämä käyttöliittymä*) -käyttöliittymällä varustettuja työpuhelimia. Henkilökohtainen Android-käyttöliittymällä varustettu puhelimeni näin ollen täytti testausten skaalan mainiosti. Tämän lisäksi henkilökohtaiseksi vastuualueen testausselaimeksi valikoitui Edge (*Microsoftin kehittämä verkkoselain*).

4.2.2 Käytettävyystestaus

Mobiilisovelluksen käytettävyystestaus suoritettiin pandemiatilanteen vuoksi etänä organisaation sisäisten asiantuntijoiden toimesta. Käytettävyystestauksen onnistumiselle tärkeää oli testiskenaarioiden ja tapausten tarkka dokumentointi. Näiden testausta avustavien dokumenttien selkeys ja ymmärrettävyys varmistettiin, ennen kuin ne annettiin testaavien henkilöiden läpikäytäviksi.

Samanaikaisesti testausta suorittavien henkilöiden antama palaute kerättiin myöhempää analysointia varten. Käytettävyystestauksen aikana löytyneet bugit myös huomioitiin ja pyrittiin korjaamaan heti. Käytettävyystestauksen lopuksi annettu palaute käytiin läpi ja arvioitiin mobiilisovelluksen kehityksen suhteen. Muun muassa palautetta saatiin erilaisista sovelluksen sisäisistä kielivalinnoista sekä tiettyjen asioiden tai toimintojen näkyvyyksistä sovelluksen sisällä. Testauksen palautteista saadut kehitysehdotukset priorisoitiin vakavuusasteittain ja siirrettiin kehitysjonoon.

4.2.3 Toimintakyvyn testaus mobiililaitteella

Testausprosessin aikana testattiin mobiililaitteille hyvin oleellista osaa: toimintakykyä. Testauksien aikaisessa vaiheessa tutkittiin verkkomobiilisovelluksen käytön vaikutusta mobiililaitteen akun kestoon. Akun kestävyyden testaus hyväksyttiin hyvin nopeasti, sillä eri mobiililaitteiden ja verkkoselainten todettiin käyttävän ohjelmaa ilman erityisen korkeaa virtatason kulutusta.

Erityisesti toimintakyvyn testauksessa otettiin huomioon sovelluksen sisäinen toimintavarmuus. Näihin testauksiin kuului painallusten responsiivisuuden mittaaminen, ohjel-

man sisäisten sivujen latausaikojen mittaaminen ja erilaisten ohjelman sisäisten palvelupyyntöjen vasteaikojen testaaminen. Kaikki testaukset suoritettiin sprinttien muodossa, joten samat toimintavarmuustestaukset suoritettiin aina ohjelman uuden version siirryttyä testaukseen. Tämän avulla pystyttiin varmistamaan ohjelman eheys sekä sovelluksen toimintavarmuus mahdollisten ohjelmointivirheiden kannalta.

Toimintakykytestausten aikana todettiin muutamia merkittäviä virheitä. Nämä kuitenkin esiintyivät etenkin ohjelmiston kehittämisen aikaisessa vaiheessa. Testauksia suoritettaessa huomattiin muun muassa, että jotkin toiminnot potilaiden tutkimustiedoissa eivät lataantuneet niin kuin niiden olisi pitänyt, tai eivät ollenkaan. Testausten aikana huomattiin myös eri käyttöjärjestelmien ja verkkoselainten välillä eroja. Esimerkiksi Chromella (*Googlen kehittämä verkkoselain*) testatessa tietyt tiedot latautuivat heikommin kuin Safarilla (*Applen kehittämä verkkoselain*). Nämä virheet ilmoitettiin ohjelmointitiimille, joka korjasi ilmenneet viat ja puutteet seuraavaan testaussprinttiin. Myöhemmissä testaussprinteissä ei erityisempiä toimintakyvyn puutteita ohjelmistossa esiintynyt.

Ohjelma on luonteeltaan verkkomobiilisovellus, joten sen toimintakyky täytyi testata myös normaalia matalammissa yhteysnopeuksissa. Sovelluksen yleistä toimintaa testattiin 4G-, 3G- ja noin 10 Mb/s:n (*Megabittiä per sekunti*) Wi-fi-yhteyden kanssa. Ohjelman todettiin toimivan kaikkien eri yhteysvarianttien kanssa hyvin. Erilaisten yhteysnopeuksien lisäksi sovelluksen reagointi äkilliseen yhteyden katkeamiseen testattiin tietovuotojen ja virheiden varalta. Testausten aikana ei esiintynyt minkäänlaisia tietovuotoja tai tietojen korruptoitumista.

4.2.4 Tutkiva testaus

Tavanomaisempiin testiskenaarioihin ja tapauksiin perustuvan ohjelmiston testauksen lisäksi tutkivaa testausta suoritettiin läpi testausprosessin. Tutkivan testauksen avulla pyrittiin löytämään normaalien käyttötapausten ja tapojen ulkopuolelta ohjelmistovirheitä. Etenkin testausprosessin lopussa hyödynnettiin tutkivaa testausta, jotta lähes valmiin ohjelman kaikki erilaiset käyttöön liittyvät virheet saataisiin minimoitua.

Tutkivaa testausta tehtäessä ohjelmiston kehityksen aikaisessa vaiheessa löydettiin muutamia virheitä, jotka vaikuttivat oleellisesti ohjelmiston toimintaan. Nämä määriteltiin

vakavuusasteiltaan vakaviksi. Muun muassa sovelluksen sisäisten istuntojen, eli sen miten kauan käyttäjä pidetään sovelluksessa sisäänkirjautuneena, pituudet vaihtelivat paljon, vaikka ohjelmoinnin aikana istunnoille oli määritelty tietty vakiopituus. Tutkivan testauksen avulla löydettiin myös virheitä ohjelmiston lokalisoinnista. Lokalisoinnilla tarkoitetaan sovelluksen sisäisen kielen kääntämistä ja sovittamista, esimerkiksi eri sivujen välillä. Nämä virheet huomioitiin ja siirrettiin edelleen savutestaukseen.

Testausprosessin tutkivan testauksen aikana ilmeni muitakin vakavuusasteeltaan lieviä virheitä. Tällaisia olivat muun muassa puhelimen näppäimistön aktivoituminen ajanvaraussivun kalenterinäkymää painettaessa sekä esitietolomakkeen jähmettyminen tietyn toiminnan johdosta. Edellä mainitut viat korjattiin heti niiden havaitsemisen jälkeen, eivätkä ne aiheuttaneet isompaa haittaa ohjelmiston käytettävyydelle.

4.2.5 Turvallisuustestaus

Verkkomobiilisovelluksen turvallisuustestauksessa suoritettiin kaikki tavanomaisten ohjelmistojen tietoturvaan sekä tietosuojaan liittyvät testaukset sekä mobiilisovelluksille ominaiset turvallisuustestaukset. Valtaosa testauksista suoritettiin testausprosessin alkupäässä, jotta selkeät tietoturvaan liittyvät tapaukset saataisiin heti optimoitua ohjelmiston vaatimusten mukaisiksi. Turvallisuustarkastukset suoritettiin aina, jos sovellukseen lisättiin tai päivitettiin tarkastuksia vaativia toiminnallisuuksia.

Valtaosa turvallisuusskannauksesta suoritettiin jo ohjelmointivaiheessa. Ohjelmointityökaluun on generoitu tieto tyypillisistä tietoturvaan liittyvistä yksityiskohdista, joista työkalu ilmoittaa, jos ohjelmoija vastaavia virheitä tekee. Ohjelmointitiimi kuitenkin suoritti myös itsenäisesti turvallisuusskannauksen. Itse testausprosessin alkuun suoritettiin tavallisimmat tietomurtoihin liittyvät testaukset. Sovellukseen muun muassa yritettiin päästä sisään väärällä käyttäjänimellä ja salasanalla sekä kirjautumissivun stressitestauksella. Testaukset olivat onnistuneita kehitysprojektin hyvin aikaisessa vaiheessa, joten tarvetta myöhemmälle turvallisuustestaukselle kirjautumisen näkökulmasta ei ollut.

Noin puolessavälissä ohjelmiston kehitystä suoritettiin testausprosessin aikana ohjelmistolle kevyt tunkeutumistestaus. Testauksessa simuloitiin tiedonurkintayritys kirjautumis-

sivulle ja sovelluksen sisäisiin tietoihin. Tiedonurkintayrityksessä käytettiin tietokantakyselyjä, joiden avulla pyrittiin saamaan tietoja sovelluksen tietokannasta. Tunkeutumistestaus sujui onnistuneesti, ja ohjelmistosta ei saatu urkittua minkäänlaisia tietoja.

Ohjelman mobiililaitteille tyypillisiä testauksia suoritettiin, jotta voitiin varmistaa sovelluksen turvallinen käyttö myös mobiililaitteilla. Testauksia tehtiin dokumenttien lataamiseen ja datan tallentamiseen liittyen. Tutkimuksiin ja maksuihin liittyviä dokumentteja ladattaessa testattiin latausprosessin katkaisua sekä yllättävää yhteyden poistamista latausprosessin aikana. Dokumenttien lataaminen keskeytyi suunnitellusti, eikä tietoja dokumenteista levinnyt ohjelman ulkopuolelle. Myös mahdollista arkaluontoisen datan tallentamista testattiin. Sovellus ei tallentanut arkaluontoisia tietoja puhelimen sisäiseen muistiin eikä verkkoselaimen välimuistiin. Sovellus ei niin ikään jakanut mitään arkaluontoisia tietoja verkkoselaimen yli muille avoinna oleville mobiilisovelluksille.

Viimeisimpänä turvallisuustestauksissa varmistettiin mobiililaitteen puhelutoiminnon käytettävyys. Mobiililaitteen pääkäyttötarkoitus on toimittaa ja vastaanottaa puheluita. Tämän vuoksi testattiin, että ohjelma siirtyy taka-alalle puhelun ajaksi. Testauksen aikana käytettiin usean eri sovelluksen puhelutoimintoja. Kaikkien puhelutapahtumien aikana ohjelmisto jäi taustalle puhelun ajaksi.

Projektiin sisältyi myös turvallisuustestauksen tekeminen teknisen kumppanin toimesta. Lisäksi yrityksessä päätettiin katselmoida järjestelmään tehdyt muutokset kolmannen osapuolen toimesta, jotta tietosuoja- ja turva tulee varmistettua.

4.2.6 Rajapintojen testaus

Verkkomobiilisovelluksen sisällä käytettiin paljon erilaisia rajapintoja. Näiden rajapintojen toimivuus ja eheys täytyi testata, jotta kaikki ohjelmiston oleellisten toimintojen käytettävyydet voitiin varmistaa. Testattavia rajapintoja oli niin ERP-järjestelmän eri osien välillä, kuin ERP-järjestelmän ulkopuolisten toimintojen välillä. Sovelluksen rajapintojen toimivuuksia testattiin etenkin testausprosessin loppupuolella.

Potilaalle kehitetyn uuden asiointijärjestelmän täytyy olla yhteydessä ERP-järjestelmän muihin moduuleihin ja toiminnallisuuksiin. Rajapintoja näiden eri toiminnallisten kokonaisuuksien välillä täytyi näin ollen testata. Ajanvarausjärjestelmän ja asiointijärjestelmän

välisten rajapintojen kommunikaation toimivuus varmistettiin aina, kun ohjelmasta tuotiin uusi toiminnallinen versio testattavaksi. Ajanvarausjärjestelmän rajapinta toimi vaatimusten mukaisesti jo hyvin aikaisesta kehityksen vaiheesta lähtien, eikä siinä ilmennyt erityisiä vikoja tai yhteysongelmia myöhemminkään.

Projektin kannalta erittäin tärkeänä toiminnallisuutena oli potilaan asiointijärjestelmän ja hoitoprosessin hallinnointijärjestelmän rajapinnan toimivuus ja eheys. Toiminnallisuuden käytettävyys tuli varmistaa aina, kun ohjelmistoon tehtiin isompia muutoksia. Hoitoprosessin hallintajärjestelmän ja asiointijärjestelmän kommunikaation oikeellisuus varmistettiin molemmista moduuleista, jotta pystyttiin todentamaan rajapinnan toimivuus. Varmistaminen tehtiin siten, että asiointijärjestelmässä näennäinen potilas siirrettiin tutkimukseen. Tämän jälkeen varmistettiin, että tutkimus näkyy hoitoprosessin hallintajärjestelmässä. Sieltä se edelleen siirrettiin tutkimuksen päättymisvaiheeseen ja maksun suorittamistilaan. Lopuksi vielä asiointijärjestelmässä varmistettiin, että tutkimus oli siirtynyt onnistuneesti loppupisteeseen, ja että maksutapahtuma voitiin suorittaa. Edellä mainittu testausprosessi suoritettiin useita kertoja, eikä sen toiminnassa esiintynyt virheitä.

Myös potilaan tietoja säilyttävän järjestelmän ja asiointijärjestelmän välisen rajapinnan toimivuus tuli varmistaa. Tällä todennettiin, että potilaan tieto säilyy autenttisena ja muuttumattomana näiden järjestelmien välisessä tietoliikenteessä. Testauksia rajapinnalle suoritettiin niin ikään läpi testausprosessin. Testausten aikana ilmeni virhe, jossa asiointijärjestelmässä tehty potilaan tietojen muutos ei välittynyt tietoja säilyttävään järjestelmään. Viasta ilmoitettiin bugiraportissa, ja ohjelmointitiimi korjasi sen seuraavaan testaussprinttiin mennessä.

Asiointijärjestelmään kehitettiin tutkimusten maksamiseen käytettävä palvelu. Tämän toiminnallisuuden avulla potilaat voivat maksaa tutkimuksensa suoraan järjestelmän kautta sekä tietokoneella että mobiililaitteella. Järjestelmässä käytettäviä maksutapoja ovat korttimaksu, eri pankkien kautta maksaminen, PayPal ja MobilePay. Kaikkien näiden maksutapojen toimivuus ja toimintavarmuus piti varmentaa, jotta sovelluksen käytettävyys säilyisi vaatimusten mukaisena. Maksutapahtumien testaamisessa ilmeni paljon ongelmia. Rajapinnan toimivuus oli kauan tilanteessa, että tutkimusten maksaminen ei onnistunut. Tähän vaikuttivat erilaiset palveluntarjoajien yhteysongelmat. Myös rajapintojen virheellinen määrittely vaikutti oleellisesti maksutapahtumien epäonnistumisiin. Ra-

japintatestausta tehtiin lähes koko kehitysprojektin ajan, jotta pystyttiin varmasti toteamaan, että maksut onnistuvat kaikilla eri maksuvälineillä ja tapahtumilla. Testausprosessin varhaisessa vaiheessa maksutapahtumia testattiin testiasetuksilla, jossa oikeata rahaa ei käytetty. Myöhemmin testaukset vielä toistettiin oikeilla maksutapahtumilla. Näin varmistuttiin siitä, että testauksen varhaisemmassa vaiheessa havaitut ongelmat eivät toistuisi loppukäyttäjillä.

4.2.7 Savu- ja regressiotestaus

Kehitysprojektin alussa projektin lähestymistavaksi valittiin ennakoiva ”Agile”-metodi. Lähestymistavalle ominaisesti kehitystyö jaettiin sprinteiksi, joihin sisältyi ohjelmointi, testaus, sekä muut suunnittelu- ja palautteenkeruuvaiheet. Jokaisen testaussprintin lopussa suoritettiin savutestaus. Testauksen avulla pyrittiin saamaan yhtenäinen käsitys siitä, voiko nykyistä ohjelman versiota julkaista loppukäyttäjien käyttöön vai tarvitaanko kehittämiselle lisää sprinttejä ja aikaa.

Savutestauksissa käytiin läpi kunkin testaussprintin aikana havaitut, vakavuusasteeltaan vakavat ongelmat. Viat olivat luonteeltaan sen laatuisia, että ne estäisivät tai vaikuttaisivat huomattavasti sovelluksen normaaliin käyttöön. Tällaisia ongelmia olivat muun muassa maksujärjestelmiin liittyvät ongelmat, lokalisoinnin epävakaa toiminta ja ohjelman istuntojen toimivuuteen liittyvät virheet. Maksujärjestelmien viat estäisivät loppukäyttäjiä joko maksamasta tutkimuksia, tai antaisivat maksutapahtumista virheellistä tietoa. Tällaiset viat aiheuttaisivat isoja ongelmia ohjelmiston luotettavuuden kannalta, joten ne piti korjata ennen ohjelmiston julkaisua loppukäyttäjille. Lokalisointiin liittyvät ongelmat vaikuttavat erityisesti ohjelmiston käytön mukavuuteen. Jos sovellus ei käännä ja sovita tekstejä oikein, esimerkiksi englanniksi, vaikeuttaa se sovelluksen englanninkielisten käyttäjien toimintaa. Ohjelman istuntoprosessissa oli ohjelmistovirhe, joka eräännytti istunnon ja kirjasi käyttäjän ulos. Tällainen virhe vaikuttaisi loppukäyttäjien haluun käyttää sovellusta. Ongelma piti siis saada poistettua ennen tuotantoon julkaisua.

Kun savutestauksien aikana pystyttiin varmentamaan, että edellä mainitut toiminnallisuudet olivat vaatimusten mukaisia, voitiin ne siirtää regressiotestaukseen, jossa niiden vaikutus muuhun ohjelmistoon varmistettiin. Toisin sanoen regressiotestauksessa testattiin kaikkien ohjelmiston osien toiminta yhdessä, ja näin ollen yleisesti koko ohjelmiston toi-

mivuus. Regressiotestauksen aikana ei enää ilmennyt ohjelmiston eri osien vuorovaikutuksesta aiheutuvia virheitä, ja koko ohjelmisto toimi niin hyvin, että se voitiin julkaista loppukäyttäjien käyttöön.

5 Testausten tulosten analysointi ja johtopäätökset

Projektin alussa kehittämiselle määriteltiin tiukka aikataulu. Tästä huolimatta testausprosessi onnistui erittäin hyvin. Testausten aikana löydettiin virheitä niin järjestelmän ohjelmakoodista, turvallisuudesta kuin toimintakyvystä. Järjestelmän julkaiseminen loppukäyttäjille myöhästyi hieman suunnitellusta julkaisupäivästä teknologia-alustan ongelmista johtuen. ERP-järjestelmän potilaille tarkoitetun asiointijärjestelmän eheys, toimivuus ja laadukkuus saatiin testausten avulla kuitenkin varmistettua. Vaikka kehitysprojekti on viety loppuun, on järjestelmää ylläpidettävä ja monitoroitava ongelmien ennaltaehkäisemiseksi.

Projektissa käytettiin kolmannen osapuolen tukea etenkin palvelujen ohjelmoinnissa. Organisaation sisäinen kehitystiimi teki suurimman osan järjestelmän testauksesta, joka tehosti projektin kulkua huomattavasti. Pelkästään organisaation omilla resursseilla näin laajan kehittämisprojektin läpivieminen olisi ollut erittäin haastavaa. Kokonaisuudessa hankkeen roolitus ja jako oli optimaalinen. Organisaation sisäinen kehitystiimi varmisti projektissa kehitettävien toiminnallisuuksien käytettävyyden. Samanaikaisesti tiimi piti huolen käytössä olevasta järjestelmästä. Julkaisun jälkeen organisaation vastuulle jäi järjestelmän toimivuuden monitorointi, mahdollisen pienjatkokehityksen tekeminen sekä vakavuusasteeltaan matalien virheiden korjaaminen. Vakavuusasteeltaan korkeiden virheiden korjaamisessa otetaan yhteys myös ohjelmoinnista vastuussa olevaan toimijaan ongelmien korjaamiseksi.

Projektin aikana käytettiin monia eri teknologisia ratkaisuja, jotka olivat allekirjoittaneelle ennestään tutkimattomia. ERP-järjestelmän modulaarinen rakenne helpotti ison kokonaisuuden hahmottamista. Perehtymällä yhteen moduuliin kerrallaan saatiin selkeä kuva yhden osa-alueen toiminnoista ja toiminnallisuuksista. Tämän jälkeen oli helppo luoda selkeä näkemys näiden osa-alueiden yhteyksistä toisiinsa. Testausprosessiin nähden arkkitehtuurisen rakenteen jäsentyneisyys, ja selkeys mahdollisti erityisesti tehokkaamman tiimityöskentelyn. Järjestelmän kehittämiseen käytettävä ohjelmointityökalu oli niin

ikään helppokäyttöinen. Eri ohjelmakoodien logiikkaa oli selkeästi helpompi ymmärtää työkalun visuaalisuuden ansiosta. Perinteistäkin ohjelmakoodia jouduttiin projektin aikana tutkimaan. Nähtävissä olevia vuokaavioita tarkastelemalla perinteisen ohjelmakoodin toiminnot oli helpompi ymmärtää. Näistä teknologisista ratkaisuista oli selkeätä hyötyä projektin onnistuneelle läpiviemiselle ja henkilökohtaiselle kehitykselle.

Testausprosessin tavoitteena oli varmistaa, että ERP-järjestelmään päivitettävän potilasasiointijärjestelmän kehityksessä ei ilmene ongelmia tai vikoja, jotka voisivat estää sen julkaisun. Yleisesti ottaen koko kehitysprojektin tavoitteena oli luoda loppukäyttäjille uusi, modernimpi ja helppokäyttöisempi asiointijärjestelmä. Myös saavutettavuus oli olennaista: asiointijärjestelmä tuli olla mobiilikäyttöinen. Loppukäyttäjille kehitettiin päivitetty organisaation toimintaa ohjaava järjestelmä, jonka toiminta on sujuvaa, aukotonta ja turvallista. Näin ollen projektin tavoitteet saavutettiin.

Johdannossa esitettiin itse opinnäytetyön tavoitteet kolmena selkeänä tutkimuskysymyksenä: ”*Toimiiko ERP-järjestelmä niin kuin halutaan?”, ”Miten ERP-järjestelmän testaus tukee palvelun kehitystä?”* sekä ”*Tuottaako ERP-järjestelmä liiketoiminnalle sen tarvitsemat hyödyt?”*. Tehdyn projektin näkökulmasta organisaation käytössä oleva ERP-järjestelmä toimii kaikkien organisaation vaatimusten ja tarpeiden mukaan. Myös projektissa päivitetyn potilaan asiointijärjestelmän testaus antoi selvää tukea ERP-järjestelmän kehitykselle. Mikäli testausta ei olisi toteutettu tai testaus olisi suoritettu vähäisemmillä resursseilla, olisi järjestelmän käytettävyys ja toimintavarmuus ollut selvästi heikompaa. Testausprosessien avulla ERP-järjestelmän toimintaan vaikuttavia virheitä ja puutteita löydettiin monia. Jos nämä virheet olisi huomattu vasta varsinaisessa käytössä, olisivat niiden vaikutukset olleet haitallisia organisaation liiketoiminnalle. Kokonaisuudessaan ERP-järjestelmän käytöstä on ehdotonta hyötyä yrityksen liiketoiminnalle. Järjestelmä helpottaa kokonaisuuden hallinnoimista. ERP-järjestelmän kautta myös asiakkaiden toiminta helpottuu, joka edelleen tarkoittaa asiakastyytyväisyyden parantumista. Kaiken kaikkiaan opinnäytetyön tavoitteet saavutettiin onnistuneesti.

Opinnäytetyön ja projektin aikana opittiin monia projekteille ominaisia asioita. Yhtenä merkittävänä aiheena oli tiimityöskentely etänä. COVID-19-pandemian vuoksi valtaosa tehdystä työstä toteutettiin etätyöskentelynä. Kyseinen seikka loi uusia työskentelytapoja niin tiimin jäsenten välille kuin myös projektin toteuttamiselle. Perustavampi tiimin jäsen-

ten koulutus olisi parantanut ja helpottanut projektin läpiviemistä testausprosessin näkökulmasta. Kuitenkin tällaiselle koulutukselle olisi vaadittu aikaa ja resursseja, jotka olisivat olleet kehitysprojektista pois. Kaiken kaikkiaan toteutettu projekti antoi erittäin hyvät lähtökohdat tiimin tuleville kehitysprojekteille.

6 Yhteenveto

Opinnäytetyön tavoitteena oli varmistaa, että toimeksiantajan yrityksen ERP-järjestelmä oli toiminnoiltaan laadukas, eheä ja turvallinen. Toiminnallisuuksien testaus toteutettiin isona kehitysprojektina. Projektissa päivitettiin ERP-järjestelmän potilaille tarkoitettu asiointijärjestelmä, joka on myös mobiilikäyttöinen.

Projektin alussa tutustuttiin käytössä olevaan ERP-järjestelmään niin pintatasolla kuin syvällisemmin ohjelmakoodin pohjalta. Ensin perehdyttiin ERP-järjestelmän toiminnallisiin kokonaisuuksiin. Näihin kuului muun muassa ajanvarausjärjestelmä, asiointijärjestelmä, hoitoprosessin hallinnointijärjestelmä sekä markkinoinninhallintajärjestelmä. Kuhunkin kokonaisuuteen perehtymällä luotiin yleiskuva koko järjestelmän toiminnasta tulevaa kehitysprojektia varten.

Kehitysprojektin alkuun luotiin suunnitelma halutusta asiointijärjestelmästä. Ohjelmiston toiminnalliset vaatimukset määriteltiin, ja sen yleisestä rakenteesta luotiin suunnitelma. Myös käytettävä projektin lähestymistapa päätettiin. Projektin luonteen kannalta sopivimmaksi käytännöksi valikoitui ennakoiva ”Agile”-menetelmä. Testaustiimi loi kevyen testaussuunnitelman tulevista testauksista sekä kattavat dokumentit erilaisista testiskenaarioista ja tapauksista. Testauksen dokumentit kirjattiin tarkistuskorttipohjaiseen projektinhallintajärjestelmään.

Testauksia suoritettiin koko ERP-järjestelmän laajuudelta päivitetyn asiointijärjestelmän laadun takaamiseksi. Eri metodien avulla ohjelmiston koko testausskaala saatiin hyödynnettyä. Testausprosessissa toteutettiin muun muassa käytettävyystestaus, toimintakykytestaus, tutkiva testaus, turvallisuustestaus sekä savu- ja regressiotestaus. Testauksia suoritettaessa havaittiin niin järjestelmän toimintaan laajasti vaikuttavia virheitä kuin vakavuusasteeltaan lievempiä ongelmia. Kaikki havaitut virheet dokumentoitiin bugiraporteiksi ja siirrettiin korjattavaksi.

Projektin julkaisu myöhästyi alun perin suunnitellusta aikataulusta. Tähän vaikuttivat odottamattomat ongelmat teknologia-alustassa. Haasteista huolimatta projekti saatiin vietyä onnistuneesti loppuun ja päivitetty asiointijärjestelmä julkaistiin loppukäyttäjille. Päivitetty järjestelmä helpottaa loppukäyttäjien asiointia, erityisesti mobiilikäyttöisyyden ansiosta. Kehitysprojektissa siis saavutettiin sille määritellyt tavoitteet.

Testausprosessin näkökulmasta tavoitteena oli varmistaa kehitettävän ohjelmiston toimivuus, eheys ja turvallisuus. Ilman laajaa ohjelmiston testausta sovelluksen laatua ei olisi saatu varmistettua yhtä hyvin. Järjestelmään olisi pahimmassa tapauksessa jäänyt virheitä, jotka olisivat vaikuttaneet yrityksen toimintaan haitallisesti. Näin ollen myös testausprosessin tavoitteet täytettiin.

Projekti saatiin suoritettua onnistuneesti loppuun. ERP-järjestelmän ylläpito kuitenkin jatkuu myös tuotannon aikana. Järjestelmää tullaan päivittämään tulevaisuudessa, sillä yrityksen kasvaessa myös ERP-järjestelmän täytyy pysyä ajantasaisena. Erityisesti erilaisille pienkehitysprojekteille on suurta tarvetta. Tulevia projekteja ja testauksia silmällä pitäen henkilöstön teknistä osaamista voitaisiin kehittää vielä paremmaksi. Järjestelmän asiantuntemus helpottaisi tiimin työskentelyä yhdessä ja tehostaisi myös yleistä toimintaa. Projekti toteutettiin valtaosin etätyöskentelynä. Etätyöskentelyn ei todettu tuottavan erityistä haittaa kehitysprojektin kululle. Näin ollen myös tulevaisuudessa vastaavantyyppiset projektit on mahdollista toteuttaa etätyöskentelynä.

Lähteet

1 ITE Wiki, ”Mikä on ERP-järjestelmä?”. Verkkoaineisto <https://www.ite-wiki.fi/p/mika-on-erpjarjestelma> Luettu 16.1.2021.

2 Taimer Ltd, ”Mikä on ERP? Kuinka ERP toimii? Aloittelijan opas”, 4.12.2018. Verkkoaineisto <https://taimer.com/fi/toiminnanohjaus-erp/mika-on-erp-kuinka-erp-toimii/> Luettu 16.1.2021.

3 Logistiikan maailma, Toiminnanohjausjärjestelmä. Verkkoaineisto <https://www.logistiikanmaailma.fi/logistiikka/ohjausjarjestelmat/toiminnanohjausjarjestelma/> Luettu 16.1.2021.

4 Tally Solutions, What is Enterprise Resource Planning System, 19.10.2019. Verkkoaineisto <https://tallysolutions.com/erp-software/what-is-enterprise-resource-planning-system/> Luettu 18.1.2021.

5 Learning Hub, Piper Thomson, The Complete History of ERP: Its Rise to a Powerful Solution, 23.1.2020. Verkkoaineisto <https://learn.g2.com/history-of-erp> Luettu 19.1.2021.

6 Investopedia, Will Kenton, Material Requirements Planning (MRP), 27.6.2020. Verkkoaineisto <https://www.investopedia.com/terms/m/mrp.asp> Luettu 19.1.2021.

7 Genius ERP, Anne Mulvenna, A Brief History of ERP, 10.7.2018. Verkkoaineisto <https://www.geniuserp.com/blog/a-brief-history-of-erps> Luettu 19.1.2021.

8 ERP Information, A Brief History of ERP. Verkkoaineisto <https://www.erp-information.com/history-of-erp.html> Luettu 19.1.2021.

9 IBM, SOA (Service-Oriented Architecture), 17.7.2019. Verkkoaineisto <https://www.ibm.com/cloud/learn/soa#:~:text=SOA%2C%20or%20service%2Doriented%20architecture,perform%20deep%20integration%20each%20time.> Luettu 23.1.2021.

10 Info World, Jonathan Freeman, ”What is an API? Application programming interfaces explained”, 8.8.2019. Verkkoaineisto <https://www.infoworld.com/article/3269878/what-is-an-api-application-programming-interfaces-explained.html> Luettu 23.1.2021.

11 Forum Systems, API Security Management. Verkkoaineisto <https://www.forumsys.com/product-solutions/api-security-management/> Luettu 23.1.2021.

12 AltexSoft, ”What is API: Definition, Types, Specifications, Documentation”, 18.6.2019. Verkkoaineisto <https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/> Luettu 24.1.2021.

13 TechTarget, Margaret Rouse, ”Open API (public API)”, 1.4.2020. Verkkoaineisto <https://searchapparchitecture.techtarget.com/definition/open-API-public-API> Luettu 24.1.2021.

14 Distribution One, ”API and ERP–What You Need to Know”, 22.3.2017. Verkkoaineisto <https://distone.com/api-erp-need-to-know/#:~:text=An%20Application%20Program%20Interface%20(API,and%20a%20program%20requesting%20data.> Luettu 24.1.2021.

15 IDAP Group, ERP Systems Classification by Types with Examples, 19.12.2020. Verkkoaineisto, <https://idapgroup.com/blog/erp-systems-classification-by-types-with-examples/> Luettu 24.1.2021.

16 Technology Advice, Enterprise Resource Planning Software Buyer's Guide, 21.1.2021. Verkkoaineisto <https://technologyadvice.com/erp/> Luettu 25.1.2021.

17 G2, Best ERP Systems for Medium-Sized Businesses. Verkkoaineisto <https://www.g2.com/categories/erp-systems/mid-market> Luettu 25.1.2021.

18 Visma, Toiminnanohjaus-järjestelmä (ERP). Verkkoaineisto <https://www.visma.fi/toiminnanohjausjarjestelma/> Luettu 25.1.2021.

19 Lemonsoft, Complete Solution For Enterprise Resource Planning. Verkkoaineisto <https://www.lemonsoft.fi/en/> Luettu 25.1.2021.

20 Roima, RoimaSoftware Lean System. Verkkoaineisto <https://www.roimaint.fi/roimasoftware-lean-system/> Luettu 25.1.2021.

21 Datacor, Caitlin O'Donnell, Industry Specific vs Generic ERP Differences & Buying Advice, 3.6.2020. Verkkoaineisto <https://www.datacor.com/the-datacor-blog/erp-selection-generic-erp-vs.-industry-specific-erp> Luettu 26.1.2021.

22 Crystol Tech, SAP Industry Solutions. Verkkoaineisto <https://www.crystol-tech.com/industries-and-solutions> Luettu 26.1.2021.

23 Solution Dot, Private Vs Public Cloud Vs Hybrid Cloud, 25.4.2018. Verkkoaineisto <https://solutiondots.com/blog/technology-blog/private-vs-public-cloud-vs-hybrid-cloud/> Luettu 26.1.2021.

24 Software Advice, Zach Hale, Toby Cox, ”Cloud ERP vs. On-Premise ERP”, 4.12.2020. Verkkoaineisto <https://www.softwareadvice.com/resources/cloud-erp-vs-on-premise/> Luettu 26.1.2021.

25 Avi Networks, Hybrid Cloud Definition. Verkkoaineisto <https://avinetworks.com/glossary/hybrid-cloud/> Luettu 26.1.2021.

26 Solutions Review, Elizabeth Quirk, ”Why You May Want to Opt for Hybrid ERP Solutions”, 10.5.2018. Verkkoaineisto <https://solutionsreview.com/enterprise-resource-planning/why-you-may-want-to-opt-for-hybrid-erp-solutions/> Luettu 26.1.2021.

27 Management Study Guide, Prachi Juneja, Technical Foundations of ERP Architectures. Verkkoaineisto <https://www.managementstudyguide.com/erp-architecture.htm> Luettu 26.1.2021.

28 Diceus, ”Types of ERP Systems: ERP System Architecture and Modules”, 9.10.2020. Verkkoaineisto <https://diceus.com/types-of-erp-systems/> Luettu 26.1.2021.

29 Talend, ”Monolithic vs. Microservices: a guide to application architecture”. Verkkoaineisto <https://www.talend.com/resources/monolithic-architecture/> Luettu 27.1.2021.

30 Wayne L. Staley, The New ERP vs. the Monolith, 2015. PDF-artikkeli <https://www.competitiveamerica.us/publications/The_New_ERP_vs_The_Monolith.pdf> Luettu 27.1.2021.

31 Gartner, Postmodern ERP. Verkkoaineisto <https://www.gartner.com/en/information-technology/glossary/postmodern-erp#:~:text=Postmodern%20ERP%20is%20a%20technology,integration%20against%20business%20flexibility%20and> Luettu 27.1.2021.

32 Kook.com, The evolution of the "Postmodern ERP". Verkkoaineisto <https://www.kook.com.au/software-integration/postmodern-erp-solutions/> Luettu 27.1.2021.

33 Deloitte, Welcome to the postmodern era for public sector ERP, 2017. PDF-artikkeli <https://www2.deloitte.com/content/dam/Deloitte/us/Documents/public-sector/us-fed-postmodern-era-for-public-sector-erp.pdf> Luettu 27.1.2021.

34 Boost High, ”Monolithic vs. Microservices Architecture”, 26.8.2019. Verkkoaineisto <https://boosthigh.com/monolithic-vs-microservices-architecture/> Luettu 27.1.2021.

35 Gartner, 2.3.2016. Verkkouutinen <https://www.gartner.com/en/newsroom/press-releases/2016-03-02-gartner-says-through-2018-90-percent-of-organizations-will-lack-a-postmodern-application-integration-strategy> Luettu 27.1.2021.

36 Epicor, Postmodern ERP is Trending: What It Is and What It Can Do for You. Verkkoaineisto <https://www.epicor.com/en/blogs/erp/postmodern-erp-is-trending-what-it-is-and-what-it-can-do-for-you/> Luettu 27.1.2021.

37 Ascent Solutions, The End of Monolithic ERP, 10.9.2019. Verkkoaineisto <https://ascenterp.com/the-end-of-monolithic-erp/> Luettu 27.1.2021.

38 Tutorials Point, Software Testing Tutorial. Verkkoaineisto <https://www.tutorialspoint.com/software_testing/index.htm> Luettu 30.1.2021.

39 ITE Wiki, Vala Group, Laadunvarmistus ja ohjelmistotestaus. Verkkoaineisto <https://www.itewiki.fi/opas/laadunvarmistus-ja-ohjelmistotestaus/> Luettu 30.1.2021.

40 ATR, Ohjelmistotestaus parantaa laatua, 4.1.2018. Verkkoaineisto <https://www.atrsoft.com/2018/01/04/ohjelmistotestaus-parantaa-laatua/> Luettu 30.1.2021.

41 Software Testing Help, ”Manual Testing Vs Automation”, 18.1.2021. Verkkoaineisto <https://www.softwaretestinghelp.com/manual-testing-vs-automation-testing/> Luettu 30.1.2021.

42 Tutorials Point, Manual Testing. Verkkoaineisto <https://www.tutorialspoint.com/software_testing_dictionary/manual_testing.htm> Luettu 30.1.2021.

43 Guru99, Manual Testing Tutorial: What is, Concepts, Types & Tool. Verkkoaineisto <https://www.guru99.com/manual-testing.html> Luettu 30.1.2021.

44 Guru99, ”Automation Testing Vs. Manual Testing: What’s the Difference?”. Verkkoaineisto <https://www.guru99.com/difference-automated-vs-manual-testing.html> Luettu 31.1.2021.

45 Tutorials Point, Automated Software Testing. Verkkoaineisto <https://www.tutorialspoint.com/software_testing_dictionary/automated_software_testing.htm> Luettu 31.1.2021.

46 DZone, Andrew Smith, ”Difference Between Black-Box, White-Box, and Grey-Box Testing”, 15.5.2020. Verkkoaineisto <https://dzone.com/articles/difference-between-black-box-white-box-and-grey-bo#:~:text=While%20black%2Dbox%20test-ers%20make,in%20a%20non%2Dintrusive%20manner.> Luettu 10.2.2021.

47 Software Testing Fundamentals, Black Box Testing, 17.9.2020. Verkkoaineisto <https://softwaretestingfundamentals.com/black-box-testing/> Luettu 11.2.2021.

48 Software Testing Fundamentals, White Box Testing, 17.9.2020. Verkkoaineisto <https://softwaretestingfundamentals.com/white-box-testing/> Luettu 11.2.2021.

49 ProfessionalQA.com, Grey Box Testing, 16.2.2019. Verkkoaineisto <https://professionalqa.com/gray-box-testing> Luettu 13.2.2021.

50 Trust Radius, Brooklin Nash, ”Agile vs Waterfall: Learn the Differences in 5 Minutes”, 7.4.2020. Verkkoaineisto <https://www.trustradius.com/buyer-blog/difference-between-agile-vs-waterfall> Luettu 13.2.2021.

51 Tutorials Point, Test Approach. Verkkoaineisto <https://www.tutorialspoint.com/software_testing_dictionary/test_approach.htm#:~:text=A%20test%20approach%20is%20the,before%20the%20build%20is%20created.> Luettu 13.2.2021.

52 Mind K, Pavlo Zinchenko, Agile vs Waterfall. Verkkoaineisto <https://www.mindk.com/blog/agile-vs-waterfall/> Luettu 13.2.2021.

53 Medium, Nicholas Ivanecky, Crash Article in Agile Development, 6.9.2016. Verkkoaineisto <https://medium.com/open-product-management/crash-article-in-agile-development-da960861259e> Luettu 19.2.2021.

54 Guru99, ”Types of Software Testing: 100 Examples of Different Testing Types”. Verkkoaineisto <https://www.guru99.com/types-of-software-testing.html> Luettu 20.2.2021.

55 Guru99, ”What is Functional Testing? Types & Examples”. Verkkoaineisto <https://www.guru99.com/functional-testing.html> Luettu 20.2.2021.

56 TestingXperts, ”Types of Software Testing You Should Know”, 23.7.2020. Verkkoaineisto <https://www.testingxperts.com/blog/types-of-software-testing> Luettu 20.2.2021.

57 Software Testing Help, ”Complete Functional Testing Guide With Its Types And Example”, 18.2.2021. Verkkoaineisto <https://www.softwaretestinghelp.com/guide-to-functional-testing/> Luettu 20.2.2021.

58 Guru99, ”Unit Testing Tutorial: What is, Types, Tools & Test EXAMPLE”. Verkkoaineisto <https://www.guru99.com/unit-testing-guide.html> Luettu 20.2.2021.

59 Testsigma, Vijay Kumar, ”Smoke Testing vs Sanity Testing vs Regression Testing Explained”, 7.2.2020. Verkkoaineisto <https://testsigma.com/blog/smoke-testing-vs-sanity-testing-vs-regression-testing-explained/#:~:text=Smoke%20testing%20is%20executed%20at,main%20functionalities%20of%20an%20application.> Luettu 21.2.2021.

60 Guru99, ”What is System Integration Testing (SIT) with Example”. Verkkoaineisto <https://www.guru99.com/system-integration-testing.html> Luettu 22.2.2021.

61 Guru99, ”What is Interface Testing? Types & Example”. Verkkoaineisto <https://www.guru99.com/interface-testing.html#:~:text=Interface%20Testing%20is%20defined%20as,API's%2C%20web%20services%2C%20etc.> Luettu 22.2.2021.

62 Tutorialspoint, Acceptance Testing. Verkkoaineisto <https://www.tutorialspoint.com/software_testing_dictionary/acceptance_testing.htm> Luettu 24.2.2021.

63 Guru99, ”What is User Acceptance Testing (UAT)? with Examples”. Verkkoaineisto <https://www.guru99.com/user-acceptance-testing.html> Luettu 24.2.2021.

64 GeeksForGeeks, Difference between Alpha and Beta Testing, 30.4.2019. Verkkoaineisto <https://www.geeksforgeeks.org/difference-between-alpha-and-beta-testing/#:~:text=Alpha%20Testing%20is%20a%20type,of%20the%20user%20acceptance%20testing.&text=Alpha%20testing%20is%20performed%20by,internal%20employees%20of%20the%20organization.> Luettu 24.2.2021.

65 Perfecto, Eran Kinsbruner, ”What Is Non Functional Testing?”, 16.4.2020. Verkkoaineisto <https://www.perfecto.io/blog/what-is-non-functional-testing> Luettu 25.2.2021.

66 Stackify, Alexandra Altvater, ”The Ultimate Guide to Performance Testing and Software Testing”, 26.4.2017. Verkkoaineisto <https://stackify.com/ultimate-guide-performance-testing-and-software-testing/> Luettu 2.3.2021.

67 Nielsen Norman Group, Kate Morgan, ”Usability Testing 101”, 1.12.2019. Verkkoaineisto <https://www.nngroup.com/articles/usability-testing-101/> Luettu 2.3.2021.

68 GeeksForGeeks, Security Testing, 10.5.2019. Verkkoaineisto <https://www.geeksforgeeks.org/software-testing-security-testing/> Luettu 3.3.2021.

69 Atlassian, Deepak Parmar, Exploratory testing. Verkkoaineisto <https://www.atlassian.com/continuous-delivery/software-testing/exploratory-testing> Luettu 3.3.2021.

70 EasyQA, ”How to test mobile application”. Verkkoaineisto <https://geteasyqa.com/qa/mobile-apps-testing/> Luettu 4.3.2021.

71 Guru99, Test Documentation in Software Testing. Verkkoaineisto <https://www.guru99.com/testing-documentation.html> Luettu 4.3.2021.

72 QA Madness, Yana Andyol, ”Software Testing Documentation: Overview”. Verkkoaineisto <https://www.qamadness.com/software-testing-documentation-overview/> Luettu 6.3.2021.

73 ITE Wiki, Low-code-ohjelmistokehitys. Verkkoaineisto <https://www.ite-wiki.fi/opas/low-code-ohjelmistokehitys/> Luettu 10.3.2021.