Saeed Babai

Vertaileva tutkimus manuaalisesta ja automatisoidusta testauksesta: haasteet ja mahdollisuudet

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

1.2.2024

Tiivistelmä

Tekijä:	Saeed Babai
Otsikko:	Vertaileva tutkimus manuaalisesta ja automatisoidusta testauksesta: haasteet ja mahdollisuudet
Sivumäärä:	32
Aika:	1.2.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaaja:	Osaamisaluepäällikkö Janne Salonen

Tutkielman tarkoituksena on vertailla manuaalisen ja automatisoidun testauksen etuja ja haasteita, soveltaen testausympäristössä käytettyä yleistä näkökulma. Tutkimus perustuu käytännön kokemuksiin dynaamisista testausmenetelmistä, kattaen testauksen periaatteet, prosessit ja eri testityypit. Manuaalista testausta tarkastellaan perinteisenä lähestymistapana, joka vaatii kärsivällisyyttä ja luovuutta. Automatisoitu testaus esitellään tehokkaana menetelmänä, jonka tavoitteena on alentaa kustannuksia ja parantaa testaustaajuutta. Tutkielma tarjoaa näkemyksiä kehittäjille ja testaajille sekä antaa käytännön ohjeita päätöksentekoon testauksen perusteella.

Avainsanat: manuaalitestaus, testiautomaatio, testausprosessi

---

Tämän opinnäytetyön alkuperä on tarkastettu Turnitin Originality Check -ohjelmalla.

Abstract

Author:	Saeed Babai
Title:	A comparative study of manual and automated testing: challenges and opportunities
Number of Pages:	32
Date:	1.2.2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Software Engineering
Supervisor:	Janne Salonen, Competence area manager

The purpose of this thesis is to compare the advantages and challenges of manual and automated testing, adopting a general perspective on the testing environment. The research is based on practical experiences with dynamic testing methods, covering testing principles, processes and various types of tests. Manual testing is examined as a traditional approach that requires patience and creativity. Automated testing is introduced as an efficient method aiming at reducing costs and improving testing frequency. The thesis provides insights for developers and testers and offers practical guidance for decision-making in testing.

Keywords: manual testing, automated testing, testing process

Sisällys

Lyhenteet

Sisällysluettelo

1 Johdanto 1

2 Testaus yleisesti 2

2.1 Miksi testaus on tarpeellista 3
2.2 Virheet, puutteet ja epäonnistumiset ohjelmistokehityksessä 3
2.3 Ohjelmiston testausperiaatteet 4
2.4 Testausprosessi 6
2.4.1 Yksikkötestaus (component testing) 7
2.4.2 Integraatiotestaus (integration testing) 7
2.4.3 Järjestelmätestaus (system testing) 7
2.4.4 Hyväksyntätestaus (acceptance testing) 9

3 Manuaalinen testaus 10

3.1 Testityypit 11
3.1.1 Toiminnallinen testaus (functional testing) 12
3.1.2 Ei-toiminnallinen testaus (non-functional testing) 12
3.1.3 Lasilaatikkotestaus (white-box testing) 13
3.1.4 Muutokseen liittyvää testausta (change-related testing) 14

4 Testausautomaatio 15

4.1 Testasuautomaatio tavoitteet 16
4.2 Testiautomaation edut 16
4.3 Testausautomaation haittoja 18
4.4 Erityyppiset automaatiotestaukset 19

5 Testausoperaatioiden optimointi: roolit, tehokkuus ja kustannukset 21

5.1 Testaajan rooli 21
5.2 Tehokkuus 23
5.3 Manuaalisen ja automaattisen testauksen kustannukset 23

5.3.1 Manuaalisen testauksen kustannukset 23
5.3.2 Automaattisen testauksen kustannukset 24
5.3.3 Kustannusten Vertailu 24

6 Yhteenveto 25

Lähteet 26

Lyhenteet

DevOps:	Development operations (kehitystoiminnot)
DevTestOps:	Development testing operations (kehityksen testaustoiminnot)
QA:	Quality assurance (laadunvarmistus)
QC:	Quality control (laadunvalvonta)

1 Johdanto

Tämän opinnäytetyön taustalla on kokemus testauksesta, joka sisältää sekä manuaalisen että automatisoidun testauksen. Aihe keskittyy näiden kahden testausmenetelmän vertailevaan tutkimukseen, erityisesti niiden haasteisiin ja mahdollisuuksiin.

Idea sai alkunsa työskenneltäessä sovellustestauksen parissa dynaamisessa ympäristössä. Tutkimuksen tavoitteena on selvittää miten nämä kaksi testaustapa kohtaavat erilaiset haasteet ja milloin kumpaakin menetelmää kannattaa hyödyntää. Näiden testausmuotojen syvällinen ymmärtäminen voi auttaa sovelluskehittäjiä ja testaajia tekemään parempia päätöksiä testaussuunnitelmissaan.

Työ on suunnattu erityisesti testausalalla työskenteleville tai siihen tutustuville henkilöille, jotka haluavat hankkia perustiedot manuaalisesta ja automatisoidusta testauksesta. Vaikka tutkimuksessa käsitellään teknisiä näkökulmia, asiat pyritään esittämään selkeästi ja ymmärrettävästi myös vähemmän teknisesti suuntautuneille lukijoille.

Opinnäytetyön tarkoituksena on tarjota käytännönläheinen näkökulma ja ohjeistus testauspäätösten tekemiseen. Lopuksi käydään läpi työn osat ja vaiheet. Tavoitteena on antaa lukijalle perustiedot manuaalisesta ja automatisoidusta testauksesta sekä auttaa heitä tekemään parempia päätöksiä omassa työssään.

2 Testaus yleisesti

Ohjelmistojärjestelmät ovat nykyään olennainen osa päivittäistä elämäämme, liiketoiminnan sovelluksista kuluttajatuotteisiin. Olemme kaikki kokeneet tilanteita, joissa ohjelmisto ei toimi odotetulla tavalla. Toimimaton ohjelmisto voi aiheuttaa monenlaisia ongelmia, kuten taloudellisia menetyksiä, ajanhukkaa tai liiketoiminnan maineen vahingoittumista. Pahimmillaan toimintahäiriö voi johtaa loukkaantumiseen tai kuolemaan. Ohjelmistotestaus on keskeinen tapa arvioida ohjelmiston laatua ja pienentää toimintahäiriöriskiä.

Yleinen virhekäsitys testauksesta on se, että testaus rajoitu pelkästään ohjelmiston suorittamiseen ja tulosten tarkistamiseen. Testaus on kuitenkin laaja prosessi, joka sisältää monia erilaisia vaiheita. Testaukseen kuuluvat aktiviteetit kuten suunnittelu, analysointi, testien suunnittelu ja toteutus. Tähän kuuluvat vielä testausraportoinnin edistys ja tulosten tarkastelu sekä itse testiobjektin laadun arviointi [1].

Testauksessa on dynaamista testausta, joka liittyy komponentin tai järjestelmän suorittamiseen, ja staattista testausta, joka ei edellytä suoritusta. Testaus ulottuu myös staattiseen testaukseen kuten vaatimusten käyttäjätarinoiden ja lähdekoodin tarkasteluun.

Toinen yleinen virhekäsitys on, että testaus keskittyy vain määrittelyjen kuten vaatimusten tarkistamiseen. Vaikka testaus varmistaa sen, että järjestelmä täyttää määritellyt vaatimukset, se sisältää myös validoinnin – tarkistuksen siitä, täyttääkö järjestelmä käyttäjien ja muiden sidosryhmien tarpeet toimintaympäristössään.

Ohjelmistotestaus on olennainen työkalu ongelmien tunnistamiseksi ja ratkaisemiseksi kehittyneen teknologian maailmassa ja, sen merkitys kasvaa jatkuvasti [1].

2.1 Miksi testaus on tarpeellista

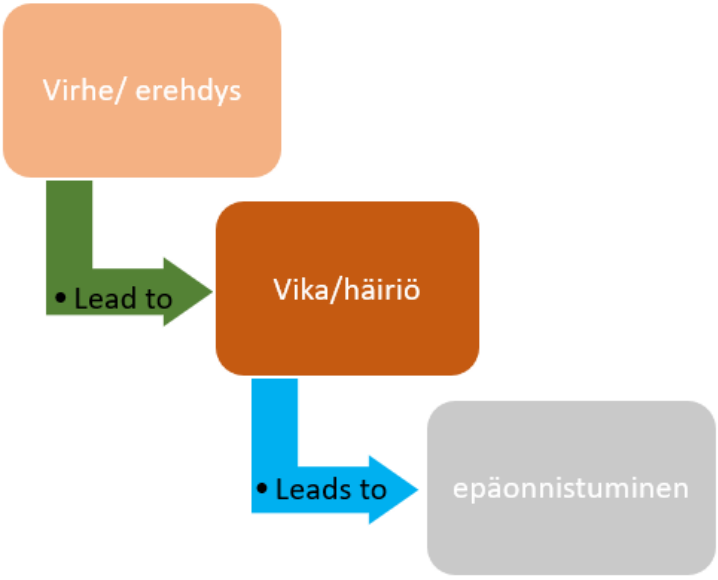
Testaus on välttämätöntä, sillä kaikki tekevät virheitä, jotka voivat olla kalliita tai jopa hengenvaarallisia. Ohjelmistotestauksen tavoitteena on varmistaa, että ohjelmisto täyttää odotetut vaatimukset ja on virheetön. Kunnolla testattu ohjelmisto varmistaa luotettavuuden, turvallisuuden ja korkean suorituskyvyn, mikä johtaa aikasäästöön, kustannustehokkuuteen ja asiakastyytyväisyyteen. Testaus auttaa tunnistamaan virheet varhain ennen ohjelmistotuotteen toimittamista ja vähentää siten toiminnan aikana tapahtuvien vikojen riskiä. Tämä on erityisen tärkeää, kun pyritään täyttämään sopimus- tai oikeudelliset vaatimukset sekä noudattamaan toimialakohtaisia standardeja kuten auto- tai turvallisuuskriittisiä järjestelmiä [2].

2.2 Virheet, puutteet ja epäonnistumiset ohjelmistokehityksessä

Ohjelmistokehityksessä virheet, puutteet ja epäonnistumiset voivat syntyä monista eri tekijöistä. Henkilö voi tehdä virheen, mikä voi johtaa vian tai virheen esiintymiseen ohjelmiston koodissa tai muussa siihen liittyvässä työtuotteessa. Esimerkiksi virhe vaatimusten kartoituksessa voi johtaa vaatimusvirheeseen, joka puolestaan johtaa ohjelmointivirheeseen ja lopulta koodivirheeseen. Virheellinen koodi saattaa aiheuttaa epäonnistumisen, mutta ei välttämättä kaikissa tilanteissa. Esimerkiksi jotkin viat vaativat hyvin tarkkoja syötteitä tai esiehtoja aiheuttaakseen epäonnistumisen, mikä voi tapahtua harvoin tai ei lainkaan.

Virheet, viat ja epäonnistumiset liittyvät toisiinsa monimutkaisella tavalla. Henkilön tekemä virhe voi tuottaa vian, joka voi johtaa epäonnistumiseen, kun järjestelmä ei toimi odotetusti. Inhimillinen virhe voi tapahtua missä tahansa ohjelmistokehityksen vaiheessa. Vika puolestaan on poikkeama odotetuista tuloksista, ja se voi ilmetä testauksen aikana. Epäonnistuminen on seuraus viasta, ja se on havaittavissa, kun järjestelmä ei suorita toivottua toimintoa. Kaikki viat eivät kuitenkaan johda epäonnistumisiin, ja jotkin voivat jäädä huomaamatta. Näiden

käsitteiden ymmärtäminen auttaa ohjelmistokehittäjiä ja testaajia tunnistamaan ja hallitsemaan riskejä ohjelmistoprojekteissa [3].



Kuva 1. Virhe, vika, epäonnistuminen [3].

2.3 Ohjelmiston testausperiaatteet

Ohjelmistotestauksen periaatteet muodostavat olennaisen osan testausprosessia, ja niiden noudattaminen on keskeistä optimaalisten testitulosten saavuttamiseksi. Taustalla olevat periaatteet auttavat ohjaamaan oikeaa testistrategiaa ja varmistamaan, että testit toteutetaan tarkoituksenmukaisesti. Testiperusteiden määrittäminen on tärkeää, jotta voidaan varmistaa se että testit kohdistuvat oikeisiin asioihin ja palvelevat testin tavoitetta.

Näiden periaatteiden taustalla on ymmärrys siitä, että testaamisella on seitsemän perustavanlaatuista periaatetta, jotka ovat olleet havaittavissa yli 40 vuoden ajan. Vaikka näitä periaatteita ei aina ymmärretä tai huomata, ne ovat läsnä useimmissa, elleivät kaikissa projekteissa. Näiden periaatteiden tunnistaminen ja niiden hyödyntäminen auttaa testaajaa toimimaan tehokkaammin. Tuntemalla nämä periaatteet ja osaamalla hyödyntää niitä voimme kehittyä paremmiksi testaajiksi.

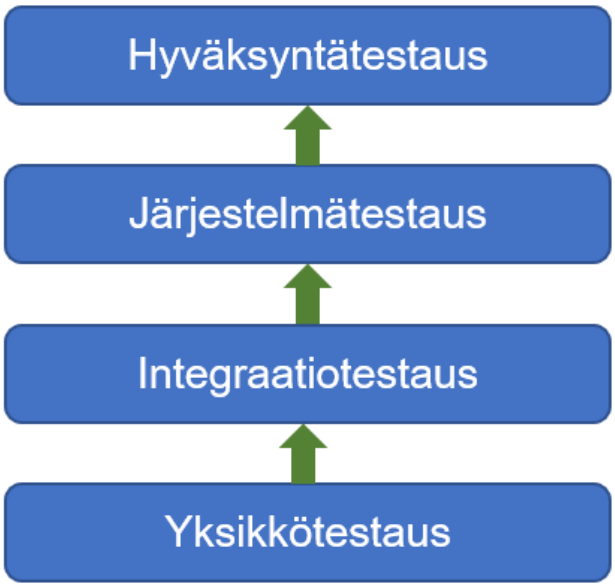
Ohjelmistotestaus on yleisin tapa varmistaa ohjelman virheettömyys ja suorituskyky. Ohjelmiston on oltava virheetön, jotta se voi suorittaa tehtävänsä tehokkaasti. Onnistuneen testauksen tuloksena ohjelmisto on vapaa virheistä, mikä takaa sen sujuvan toiminnan. Tämä korostaa testauksen keskeistä roolia ohjelmiston laadun varmistamisessa ja virheiden tunnistamisessa ennen tuotantoon siirtymistä [4].

Ohjelmiston testausperiaatteet ovat [5]:

- Tyhjentävä testaus ei ole mahdollista: testaus ei pysty osoittamaan ohjelmiston täydellistä virheettömyyttä, ja kattava testaus ei ole mahdollista. Sen sijaan on tärkeää suorittaa optimaalinen määrä tarpeellisia testejä riskianalyysin perusteella.
- Vikakokoonpano: vikakokoonpanon periaate korostaa, että suurin osa vioista kasaantuu pienelle määrälle moduuleja. Tämä perustuu Pareto-periaatteeseen, jossa noin 80 % ongelmista ilmenee 20% moduuleissa.
- Torjunta-aineiden paradoksi: torjunta-aineiden paradoksi varoittaa toistuvien testien rajoituksista ja kannustaa jatkuvasti kehittämään testausmenetelmiä uusien virheiden löytämiseksi.
- Testaus osoittaa, että ohjelmistossa on vikoja: testaus osoittaa vikojen läsnäolon eikä niiden puuttumista, mikä korostaa testauksen tärkeyttä ohjelmiston vikojen vähentämiseksi.
- Virheiden puuttuminen: virheiden puuttuminen ei takaa ohjelmiston käyttökelpoisuutta, ja testauksen tulee keskittyä myös varmistamaan, että ohjelmisto täyttää liiketoiminnan tarpeet.
- Varhainen testi ohjelmistoprosessissa: varhainen testaus säästää aikaa ja rahaa, ja testauksen tulisi alkaa kehitysprosessin mahdollisimman varhaisessa vaiheessa.
- Testaus on asiayhteydestä riippuvainen: testaus on kontekstiriippuvainen, mikä tarkoittaa, että erilaiset ohjelmistot vaativat erilaisia testauslähestymistapoja.

2.4 Testausprosessi

Ohjelmistonkehityksen testivaiheet edustavat selkeästi määriteltyjä vaiheita, joilla on omat tavoitteensa ja vaatimuksensa. Niihin kuuluvat komponenttitestaus, integraatiotestaus, järjestelmätestaus ja hyväksyntätestaus. Keskeinen osa on testiympäristö, joka voi olla jaettu tai omistettu. Esimerkiksi hyväksyntätestaus vaatii usein ympäristön, joka vastaa mahdollisimman tarkasti tuotantoympäristöä. Komponenttitestauksessa kehittäjät voivat käyttää kehitysympäristöään, kun taas järjestelmätestaus saattaa tarvita ympäristön, joka jäljittelee tuotantoa tiettyjen ulkoisten liitäntöjen kanssa. Näiden näkökohtien selkeä määrittely koko projektiryhmälle varmistaa tehokkaan yhteistyön ja estää ongelmat ympäristöriippuvuuksien osalta. Testisuunnittelun aikana vastuussa olevien johtajien tulisi harkita, miten he aikovat testata järjestelmän kokoonpanoa, jos tällaiset tiedot ovat osa järjestelmää. Tavoitteena on estää päällekkäisyyksiä ja toistoa eri vaiheiden välillä kehityssyklin aikana edistää tehokkuutta ja perusteellista testauskattavuutta [6].



Kuva 2. Testausprosessin vaiheet [7].

2.4.1 Yksikkötestaus (component testing)

Komponenttitesti, jota kutsutaan myös yksikkö- tai moduulitestaukseksi, on ohjelmistotestauslähestymistapa, joka keskittyy yksittäisten osien (komponenttien/moduulien) arviointiin suuremmassa ohjelmistojärjestelmässä. Sen tavoitteena on löytää virheet, varmistaa toiminnallisuus ja se, että kukin komponentti suorittaa ainutlaatuisen tehtävänsä tehokkaasti. Tämä testivaihe suoritetaan erikseen jokaiselle komponentille, jotta voidaan varmistaa sen toimivuus ja käytettävyys eristyksissä ennen integrointia muiden komponenttien kanssa. Komponenttitestauksella varmistetaan ohjelmistosovellusten luotettavuus ja vahvuus, koska siinä käsitellään ongelmia yksittäisen moduulin tasolla [8].

2.4.2 Integraatiotestaus (integration testing)

Integraatiotestaus varmistaa erilaisten ohjelmistoyksiköiden tai komponenttien sujuvan yhteistyön, vaikka ne olisivatkin eri ohjelmoijien laatimia. Pääpaino on näiden komponenttien välisissä liittymissä, mahdollisten virheiden paljastamisessa integroinnin aikana. Päämääränä on varmistaa, että integroidut komponentit toimivat saumattomasti yhtenäisenä kokonaisuutena ja ylläpitävät odotettua suorituskykyä. Tämä lähestymistapa auttaa tunnistamaan mahdolliset ongelmat kehityksen varhaisessa vaiheessa ja varmistaa vahvat ja virheettömät vuorovaikutukset komponenttien välillä. Automaattiset integraation regressiotestit antavat luottamusta siihen, etteivät muutokset ole häirinneet olemassa olevia liittymiä, komponentteja tai järjestelmiä. Siten ohjelmistoon luotettavuus paranee kokonaisvaltaisesti [8].

2.4.3 Järjestelmätestaus (system testing)

Ohjelmistokehitysprosessissa keskeinen vaihe on järjestelmätestaus, joka arvioi koko järjestelmän tai tuotteen käyttäytymistä. Tämä testausvaihe ottaa huomioon sekä toiminnalliset että ei-toiminnalliset käyttäytymiset eri järjestelmän toiminnoissa. Järjestelmätestauksen ensisijaiset tavoitteet ovat riskin

vähentäminen, varmistaminen siitä, että järjestelmä on kokonainen ja toimii odotetusti, sekä luottamuksen rakentaminen järjestelmän kokonaislaatuun.

Yksi keskeinen osa järjestelmätestausta on datan laadun varmistaminen, mikä voi olla ratkaisevan tärkeää tietyille järjestelmille. Testausprosessi sisältää perusteellisen tarkastelun järjestelmän käyttäytymisestä riskianalyysiraporttien, järjestelmävaatimusten, liiketoimintaprosessien ja muiden korkean tason kuvauksien perusteella. Se käsittelee myös vuorovaikutuksia käyttöjärjestelmän ja järjestelmäresurssien kanssa sekä oikeudellisten tai sääntelyvaatimusten ja ulkoisten standardien noudattamista.

Järjestelmätestauksessa automatisoinnilla on merkittävä rooli regressiosarjaa luottaessa. Regressiosarjan avulla voidaan varmistaa, etteivät muutokset vaikuta negatiivisesti olemassa olevan järjestelmän toiminnallisuuteen. Järjestelmätestauksesta johdettu tieto on korvaamaton osakkeenomistajille, sillä he tekevät päätökset saadun informaation pohjalta. Siten järjestelmätestaus toimii kriittisenä vaiheena ennen käyttäjän hyväksymistestauksen aloittamista ja määrittää järjestelmään valmiuden käyttöönottoa varten.

sidosryhmien omassa rollissaan tulisi painottaa sellaisen testiympäristön luomista, joka vastaa mahdollisimman tarkasti lopullista tuotantoympäristöä. Testausprosessi on sitä tehokkaampi mitä tarkemmin testaus olosuuteet heijastavat todellisia skenaarioita [8].

2.4.4 Hyväksyntätestaus (acceptance testing)

Hyväksyntätestaus, ohjelmistotestauksen viimeinen vaihe, ratkaisee ohjelmiston valmiuden ennen käyttöönottoa. Toisin kuin aiemmat testausvaiheet, jotka keskittyivät virheiden tunnistamiseen, hyväksyntätestaus arvioi, validoi ohjelmiston soveltuvuutta tuotantoon ja sen käyttäjille.

Tämä viimeinen testi vaihe on ohjelmistotuotteen musta laatikko -testaus (black -box), jossa toiminnallisuus tarkistetaan perusteellisesti, jotta voidaan varmistaa sen täyttävän ennalta määrätyt hyväksyntäkriteerit. Painopiste on luottamuksessa järjestelmän kokonaislaatuun ja täydellisyyteen sekä odotetussa toiminnallisuudessa. Vaikka vikoja voidaan havaita tässä prosessissa, ensisijainen tavoite ei ole virheiden havaitseminen vaan järjestelmän valmiuden varmistaminen todelliseen käyttöönottoon.

Järjestelmätestausta seuraavan hyväksyntätestauksen aikana kehitysorganisaatio korjaa tunnistetut virheet. Siirtyminen hyväksyntätestaukseen merkitsee kriittistä vaihetta, jossa selvitetään, onko järjestelmä sopiva julkaisuun loppukäyttäjille tai asiakkaille. Painopiste laajenee koskemaan toiminnallisten näkökohtien lisäksi myös ei toiminnallisia käyttäytymismalleja, jotka on määritelty järjestelmälle.

Hyväksyntätestaus toimii viimeisenä tarkastuspisteenä ennen ohjelmiston saatavuutta varsinaiseen käyttöön. Siinä varmistetaan sääntelyyn, standardien ja lainsäädännöllisten vaatimusten noudattamista. Tämän vaiheen aikana havaitut virheet voivat viitata merkittäviin ongelmiin koko järjestelmässä ja aiheuttaa mahdollisia projektiriskejä [8].

3 Manuaalinen testaus

Manuaalinen testaus edustaa perinteisintä testausmenetelmää ja sen keskeinen tehtävä on varmistaa ohjelmistonlaatu. Tässä lähestymistavassa testit suoritetaan huolellisesti pyrkien löytämään sekä selkeät että piilevät puutteet ohjelmistosta ilman automaattisten työkalujen käyttöä. Testikattavuuden varmistaminen vaati jatkuvaa ylläpitoa ja panostusta aikaresursseihin.

Testausprosessi vaatii kärsivällisyyttä ja luovuutta. Koska sovellukset on suunniteltu ihmisten käyttöön, niitä testaavat ihmiset, ja tarkastelu suoritetaan loppukäyttäjän näkökulmasta.

Testit suoritetaan manuaalisesti ilman automaattisia työkaljua. Jokainen testitapaus käydään läpi huolellisesti loppukäyttäjän näkökulma huomioiden. Siten voidaan varmistaa se, että ohjelmisto noudattaa vaatimusasiakirjaa.

Tämä prosessi on usein ensimmäinen vaihe ennen automaattista testausta, ja se vaatii huomattavia ponnisteluja ja aikaresurssien panostusta. Samalla voidaan varmistua siitä, ettei ohjelmistossa ole virheitä. Manuaalisen testauksen vääjäämättömyyden tunnustaminen liittyy ohjelmistotestauksen perusperiaatteeseen, joka korostaa "100 prosentin automaation" saavuttamisen mahdottomuutta.

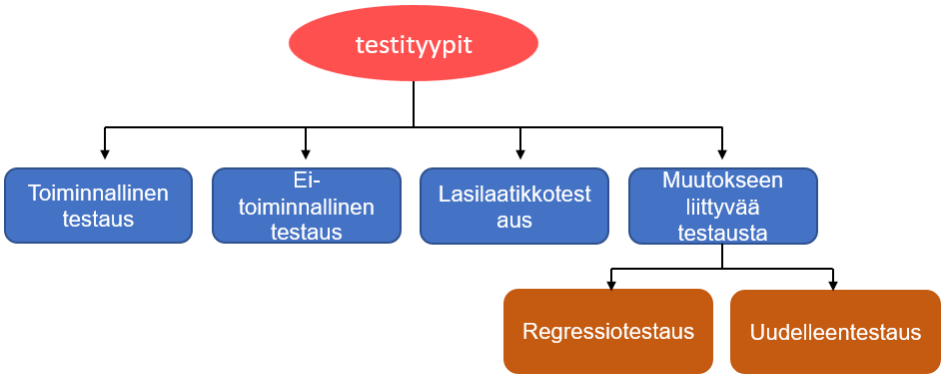
Suorittamalla manuaalista testausta mahdolliset ongelmat ja virheet havaitaan varhain sovelluksen elinkaaren aikana. Tämä tuo vakautta ja toimittamalla laadukkaan tuotteen loppukäyttäjille. Testaajat voivat tarkastella sovellusta loppukäyttäjän näkökulmasta ja helpottaa tarkkojen testitapausten laatimista ja nopean palautteen saamista sovelluksen suorituskyvystä. Ytimekkäästi sanottuna manuaalinen testaus ei ole pelkästään testausmenetelmä. Se on huolellinen tutkimus, joka varmistaa ohjelmistotuotteiden luotettavuuden ja laadun.

3.1 Testityypit

Manuaalisessa testauksessa useiden testityyppien käyttäminen varmistaa ohjelmistojärjestelmien vahvuuden ja luotettavuuden. Tulevissa luvuissa syvennymme näistä muutamiin kuten toiminnallinen testaus (functional testing), ei-toiminnallinen testaus (non-functional testing), valkoinen laatikko -testaus (white-box -testaus) ja muutoksiin liittyvä testaus (change-related testing).

Testityyppien monimuotoisuus on välttämätöntä, sillä pelkkä toiminnallinen tarkastelu ei riitä tuottamaan tavoiteltuja kokonaistestituloksia. Kukin testityyppi palvelee erityisiä tavoitteita, joiden avulla voidaan varmentaa ohjelmiston suorituskyky.

Sopivimman testityypin valinta yhtenäistää testitavoitteita, helpottaa päätöksentekoa ja tehostaa viestintää testauskehyksessä. Seuraavassa pyritään avaamaan näiden testityyppien ainutlaatuisia metodologioita, monimutkaisuuksia ja käytännön näkökohtia [9].



Kuva 3. Testityypit [10].

3.1.1 Toiminnallinen testaus (functional testing) [11]

Funktionaalinen testaus on tapa arvioida ohjelmiston toimintoja ja varmistaa, että se toimii odotetulla tavalla. Kun testaamme ohjelmistoa funktionaalisesti, tarkastelemme, miten eri osat, kuten käyttöliittymä, tietokanta ja turvallisuus, suoriutuvat niille asetetuista tehtävistä.

Testit perustuvat ohjelman toiminnallisiin vaatimuksiin, jotka voivat olla liiketoimintavaatimuksia, käyttötapauksia tai muita toiminnallisia määrittelyjä. Näiden vaatimusten avulla luomme testitapauksia ja suoritamme ne joko manuaalisesti tai automaattisesti.

Toiminnallinen kattavuus on tapa mitata, kuinka monta ohjelman toiminnallisuutta testeissä on käsitelty. Tämä auttaa varmistamaan, että ohjelmisto toimii monipuolisesti ja vastaa liiketoiminnan tarpeisiin.

Kaiken kaikkiaan funktionaalinen testaus on keskeinen osa ohjelmiston laadun varmistamista. Se auttaa löytämään ja korjaamaan mahdolliset virheet sekä varmistaa sen, että ohjelmisto täyttää käyttäjien tarpeet luotettavasti ja tehokkaasti.

3.1.2 Ei-toiminnallinen testaus (non-functional testing)

Ei-toiminnallinen testaus on testaustyyppi, joka arvioi ohjelmiston suorituskykyä, käytettävyyttä, luotettavuutta ja muita ei- toiminnallisia ominaisuuksia. Sen tarkoituksena on testata järjestelmän valmiutta ei- toiminnallisten kriteerien mukaisesti, joita funktionaalinen testaus ei koskaan huomioi.

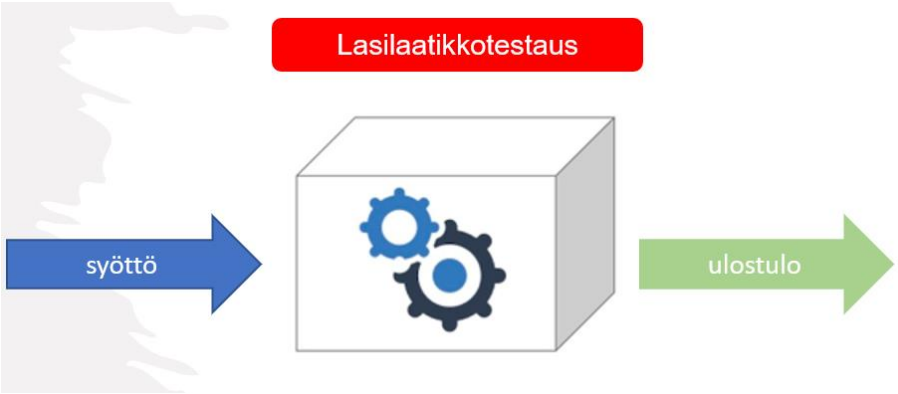
Ei-toiminnallinen testaus arvioi sen kuinka hyvin järjestelmä käyttäytyy. Näitä kriteereitä ovat käytettävyys, suorituskyky ja turvallisuus. Vaikka yleinen käsitys on, että tätä testausta suoritetaan vain myöhemmissä vaiheissa, sitä tulisi itse asiassa tehdä kaikilla testauksen tasoilla ja mahdollisimman varhain. Ei-toiminnallisten virheiden myöhäinen havaitseminen voi olla erittäin vaarallista projektin onnistumiselle [12].

Ei-toiminnallisen testauksen perusteella voidaan mitata testauksen kattavuutta. Se kertoo, miten kattavasti tiettyä ei-toiminnallista elementtiä on testattu. Se voisi tarkoittaa esimerkiksi yhteensopivuustestauksen osalta sitä, kuinka monta erilaista laitetta on testattu, ja onko testaus kattanut kaikki oleelliset osat.

Ei-toiminnallisen testauksen suunnittelu ja suorittaminen voivat vaatia erityistietoa tai -taitoja, kuten ymmärrystä suunnittelun tai teknologian sisäisistä heikkouksista (esim. tietoturvahaavoittuvuudet tiettyihin ohjelmointikieliin liittyen) tai käyttäjäkunnan erityispiirteistä (esim. terveydenhuollon hallintajärjestelmien käyttäjäprofiileista).

3.1.3 Lasilaatikkotestaus (white-box testing)

Lasilaatikkotestaus, jota on usein kutsuttu rakenteelliseksi testaukseksi sukeltaa järjestelmän tai komponentin sisäiseen rakenteeseen tai toteutukseen. Tutkittavia näkökohtia ovat koodi, arkkitehtuuri, liiketoimintaprosessit ja tiedonkulku järjestelmässä. Nimi "white-box" kuvastaa kiinnostustamme ymmärtää, mitä tapahtuu metaforisen laatikon sisällä.



Kuva 4. Lasilaatikkotestaus [13.]

Tämä testauslähestymistapa keskittyy pääasiassa testauksen perusteellisuuden mittaamiseen erilaisten rakenteellisten elementtien tai kattavuuskohteiden kautta. Toisin kuin joissakin muissa testausmenetelmissä, white-box -testaus tarjoaa näkyvyyden ohjelmiston sisäiseen logiikkaan, rakenteeseen ja

koodaukseen. Testaajilla on kattava tietämys sekä pääsy lähdekoodiin ja suunnitteludokumentteihin, mikä mahdollistaa ohjelmiston sisäisten toimintojen ja integraatioiden tarkastelun ja varmennuksen.

Eri testitasoilla white-box -testaus voi saada moninaisia muotoja. Komponenttitestauksessa painopiste on usein koodin kattavuudessa, jolloin arvioidaan testatun komponenttikoodin prosenttiosuutta. Siirryttäessä kohteen integraatiotestaukseen white-box -testaus voi siirtää painopistettään järjestelmän arkkitehtuuriin, jolloin se mittaa rakenteellista kattavuutta komponenttien välillä.

Vaikka white-box -testaus on sovellettavissa missä tahansa testitasossa, sitä käytetään yleisimmin komponenttitestauksessa ja komponenttien integraatiotestauksessa. Työkalutuki on vahva näillä tasoilla, erityisesti koodikattavuuden mittaamisessa. Kattavuuden mittausvälineet arvioivat suoritettujen elementtien prosenttiosuutta. Mittaus kohteena ovat esim. lauseet tai päätöstulokset, joita testiryhmä on käyttänyt. Jos kattavuus jää alle 100 prosentin, lisätestit saattavat olla tarpeen kattamattomien osien osalta, riippuen määritellyistä poistumiskriteereistä [13].

3.1.4 Muutokseen liittyvää testausta (change-related testing)

Kun järjestelmään tehdään muutoksia, oli kyseessä vian korjaus tai uuden toiminnallisuuden toteutus, testaaminen on välttämätöntä, jotta voidaan varmistaa muutosten oikeellisuus ja estää mahdolliset ennalta arvaamattomat haitalliset seuraukset. Tällöin käytetään uudelleentestausta (re-testing) ja regressiotestausta (regression testing).

Uudelleentestaus: Kun vika on korjattu, ohjelmisto testataan kaikilla niillä testitapauksilla, jotka aiemmin epäonnistuivat vian takia. Nämä testitapaukset suoritetaan uudelleen päivitetyn ohjelmistoversion kanssa. Lisäksi voidaan ottaa käyttöön uusia testejä, jotka kattavat vian korjaamiseen tehtyjä muutoksia. Vähintäänkin vian aiheuttaman epäonnistumisen toistamiseksi tarvittavat vaiheet on

suoritettava uudella ohjelmistoversiolla. Vahvistustestauksen tarkoituksena on varmistaa, että alkuperäinen vika on korjattu onnistuneesti.

Regressiotestaus: Yhden osan koodiin tehtävät muutokset (korjaus tai muu muutos) voivat tahattomasti vaikuttaa muiden koodiosien toimintaan. Testauksella varmistetaan, että muutokset eivät ole tuottaneet tai paljastaneet uusia vikoja ohjelmiston alueilla, joita ei ole muutettu. Testaus suoritetaan, kun ohjelmisto tai sen ympäristö muuttuu [14].

4 Testausautomaatio

Ohjelmistotestauksessa on kaksi pääasiallista tapaa: manuaalinen ja automaattinen. Testaustomaatio tarkoittaa testien tekemistä tietokone avusteisesti. Manuaalisessa testauksessa sen erottaa juuri automaattisten työkalujen käyttö. Se auttaa hallitsemaan testitietoa ja käyttämään tuloksia ohjelmiston parantamiseksi. Testausautomaatio on kuin joukkuepeli, johon osallistuvat kaikki liiketoiminnan analyytikoista kehittäjiin ja DevOps-insinööreihin.

Automaattinen testaus Se sopii suuriin projekteihin tai testeihin, jotka toistuvat usein. Sen edut tulevat esiin, kun on kyse toistuvista testeistä kuten vanhojen virheiden tarkistamisesta. Vaikka testien tekemiseen tarvitaan huolellista suunnittelua, se auttaa lisäämään ohjelmiston laatua ja testikattavuutta. Siinä käytetään erilaisia työkaluja, kuten automaatio- ja testikehyksiä. Työkalut valitaan riippuen siitä, millainen järjestelmä on kyseessä ja millaisia testejä haluamme tehdä. Joten, tiivistäen, testausautomaatio ei ole vain testien suorittamista. Se on koko prosessi, joka sisältää suunnittelun, suorittamisen ja tulosten tarkistamisen [15].

4.1 Testasuautomaatio tavoitteet

Ohjelmistotestauksen ydintarkoitus on tuottaa korkealaatuista ohjelmistoa, ja automaatio on keskeisessä roolissa tämän tavoitteen saavuttamisessa. Testausautomaatiota pyritään parantamaan selkeillä tavoitteilla, joiden tarkoituksena on tehostaa testausta ja helpottaa kokonaiskustannuksia. Vaikka se ei ole päämäärä sinänsä, ensisijainen tavoite on vähentää kustannuksia parannetun testitehokkuuden avulla. Ohjelmiston laadun ylläpitäminen tai jopa nostaminen on yhtä tärkeää. Testausautomaatio parantaa laatua koska se laajentaa toiminnallista kattavuutta ja toteuttaa testejä, jotka olisivat manuaalisesti hankalia resurssirajoitusten vuoksi. Esimerkkejä ovat monipuolisten datamääritysten perusteellinen testaaminen, vikasietoisuuden arviointi ja erilaisten suorituskykytestien suorittaminen. Lisäksi automaatio mahdollistaa laajojen testipakettien yhtenäisen ja toistuvan suorituksen eri ohjelmistoversioilla tai eri ympäristöissä niin, että testaus on taloudellistesi toteutettavissa [16].

Automaatio säästää myös aikaa ja vapauttaa QA-ammattilaiset keskittymään monimutkaisempiin tehtäviin, joissa tarvitaan inhimillistä arviointia. Automaatio vähentää merkittävästi paitsi aika myös vaivaa, erityisesti tilanteissa joissa manuaalinen testaus vie paljon aikaa. Automaatio yksinkertaistaa monimutkaisten sovellusten testausta ja tarjoaa tehokkaan tavan havaita ja poistaa taantumia, regressioita. Laadunvarmistusinsinöörit voivat määrittää automatisoituja järjestelmiä suorittamaan testejä ja helpottaa siten toistettavien testien suorittamista.

4.2 Testiautomaation edut

Automaattinen testaus tuo mukanaan monia etuja, joista yksi keskeisimmistä on automatisoitujen regressiotestisarjojen luominen. Nämä sarjat mahdollistavat suuremman määrän testitapauksia ohjelmiston julkaisua kohti manuaaliseen regressiotestauksen verrattuna. Manuaaliseen testauksen tehokkuus vähenee ajan myötä heikkenevään keskittymisen ja motivaation vähenemiseen vaikutuksesta. Automaattiset testit tarjoavat nopeamman suorituksen ja ovat vähemmän

alttiita virheille ja kykenevät toistamaan monimutkaisia testiskenaarioita vaivattomasti [17].

Automatisoidun testaukseen hyödyt tulevat esiin kuormitus-, suorituskyky- ja rasitustestaus, jotka ovat suhteellisen yksinkertaisia automatisoida. Automaattisten testitapausten muodollinen ja yhtenäinen kuvaus määritellyssä viitekehyksessä eliminoi tulkinnanvaraisuuden ja parantaa siten testien yhdenmukaisuutta, toistettavuutta ja kokonaisluotettavuutta.

Projektinäkökulmasta katsottuna automaattisella testauksella on merkittäviä etuja. Testattavan järjestelmän laadusta saatava välitön palaute kiihdyttää projektin etenemistä, koska mahdolliset ongelmat kyetään tunnistamaan ja ratkaisemaan nopeasti. Tämä tehokkuus ulottuu testausresursseihin, teknisiin infrastruktuureihin ja testausryhmiin. Testaajat voivat silloin käyttää enemmän aikaa virheiden löytämiseen kokeellisen testauksen tai erilaisten dynaamisten, manuaalisten testausmenetelmien kohdennetun käytön kautta.

Yhteenvetona automaattisen testauksen eduista voidaan esittää seuraava [18]:

1. Aikasäästö: Automaatio nopeuttaa testausprosessia ja varmistaa uusien ominaisuuksien nopeamaan validoinnin kuin manuaalinen testaus, missä lukuisien testitapausten kirjoittaminen voi olla aikaa vievää.

2. Tuottavuuden paraneminen: Automaattiset testit, joissa ei tarvita ihmisen väliintuloa suorituksen aikana (esim. myöhään illalla). Automatiikan ansiosta kyetään optimoimaan ohjelmistokehittäjien ja testaajien tuottavuus.

3. Tarkkuuden paraneminen: Automaatio vähentää virheiden mahdollisuutta ja varmistaa yhdenmukaiset ja tarkat tulokset joka kerta samaa testitapausta suoritettaessa.

4. Testiskenaarioiden uudelleenkäytettävyys: Automaattisia testiskriptejä voidaan käyttää uudelleen, jolloin uusia skriptejä ei tarvitaa, mikä vähentää kustannuksia ja inhimillisen virheen riskiä.

5. Mahdollisuus testata eri alustoilla: Automaattinen testaus mahdollistaa sovelluksen testauksen eri verkkoselaimilla ja käyttöjärjestelmillä.

6. Testien suoritus 24/7: Automaattisessa testauksessa voimme aloittaa testausprosessin missä ja milloin tahansa. Se voidaan myös suorittaa etäältä, jos meillä ei ole monia lähestymistapoja tai mahdollisuutta ostaa lisää testauslaitteita.

4.3 Testausautomaation haittoja

Vaikka testausautomaatio tarjoaa etuja, se tuo mukanaan myös haasteita. Organisaatioiden tulee tiedostaa seuraavaa [19]:

1. Alkuinvestoinnit ja jatkuvat kustannukset: Automaation käyttöönotto vaatii rahaa työkaluihin ja koulutukseen. Päivityksiin ja ylläpitoon liittyvät jatkuvat kulut lisäävät kustannuksia.

2. Taidot ja koulutus: Automaatioinsinööreiltä vaaditaan erityistaitoja ja jatkuvaa koulutusta pysyäkseen mukana teknologisessa kehityksessä.

3. Monimutkaisuus ja virheenkorjaus: Automaattisten testien luominen ja korjaaminen voi olla monimutkaista ja aikaa vievää. Virheitä on vaikea tunnistaa ja korjata.

4. Rajoitettu testikattavuus: Tietyt testit ovat vaikeita automatisoida, mikä voi synnyttää aukkoja kattavuuteen.

5. Sopeutuminen muutoksiin: Automaattisten skriptien päivittäminen ohjelmistomuutoksiin voi olla vaikeaa, mikä vaikuttaa testien luotettavuuteen.

6. Epärealistiset odotukset: Liian korkeat odotukset voivat johtaa pettymykseen ja aiheuttaa panostuksen aliarviointia.

7. Vastustus ja päätöksenteko: Jotkut organisaatiot vastustavat automaatiota koettujen kustannusten ja epävarmuuksien vuoksi.

8. Manuaalisen testauksen puutteellinen poistaminen: Tietyt testit vaativat edelleen manuaalista työtä. Haasteeksi muodostuu automatisoidun ja manuaalisen testauksen tasapainottaminen.

4.4 Erityyppiset automaatiotestaukset

Automaatio testauksia on useita sillä niillä tarkkaillaan eri asioita. Seuraavassa on esitelty erilaisia testaus tapoja [20]:

1. Avainsanaohjattu testaus (keyword-driven testing): Tämä testaus on toiminnallisen automaatiotestauksen lähestymistapa, joka hyödyntää tietotaulukoita testitoimintojen määrittelyssä. Tätä menetelmää varten käytetään inhimillisesti luettavaa kieltä kuvaamaan sovelluksen prosesseja, mikä tekee siitä saavutettavan eri alojen tiiminjäsenille ilman syvällistä teknistä asiantuntemusta. Testaajat voivat rakentaa skriptejä käyttämällä tarkkoja avainsanoja, mikä helpottaa luettavien testitapausten luomista, päivittämistä ja uudelleenkäyttöä ilman huolta alustariippuvuuksista tai ohjelmointikielistä.

2. Integrointitestaus (integration testing): Testaus varmistaa sen, että sovelluksen erilliset moduulit toimivat oikein koodimuutosten jälkeen. Se havaitsee viestintäongelmat sovelluskomponenttien välillä, vaikka ne olisivat jo osoittautuneet toimiviksi itsenäisinä yksikköinä.

3. Yksikkötestaus (unit testing): Testaus pyrkii varmistamaan jokaisen koodiobjektin toimimiseen itsenäisesti muiden toiminnallisten

moduulien tai ulkoisen koodin tuella. Automatisointi lisää testattavien komponenttien määrää ja suo kehittäjille muutos mahdollisuuden vaikuttamatta kokonaisvaltaiseen toiminnallisuuteen.

4. Savutestaus (smoke test): Testaus on toiminnallisen testauksen varhainen vaihe ja vahvistaa sovelluksen perustoiminnot. Se tunnistaa ohjelmassa olevat merkittävät toimintovirheet ennen julkaisua ja tarjoaa kustannustehokkaan tavan havaita ongelmia.

5. Regressiontestaus (regression testing): Regressiontestaus tarkastaa sovelluksen toiminnalliset ja ei-toiminnalliset näkökohdat kehittäjien tekemien koodimuutosten jälkeen ja varmistaa suorituskyvyn vakauden.

6. Suorituskykytestaus (performance testing): Suorituskykytestaus arvioi sovelluksen nopeutta, viivettä ja vakautta erilaisissa olosuhteissa. Se pyrkii jäljittelemään todellisia käyttötilanteita analysoimalla tuloksia ja varmistamalla sen, että määritettyjä suorituskykymetriikoita noudatetaan.

7. Tietoturvatestaus (security testing): Turvallisuustestaus havaitsee mahdolliset riskit järjestelmässä ennen kuin haitalliset hakkerit saavat mahdollisuuden hyödyntää niitä. Se auttaa kehittäjiä tarkistamaan turvallisuusriskit kussakin kehitysvaiheessa.

8. Aineisto-ohjattu testaus (data-driven testing): Testaus käyttää ehtotaulukoita tarjoamaan testisyötteitä ja tallentamaan tulokset. Tämän jälkeen kehittäjät käyttävät taulukkoa luodakseen koodin, joka käyttää taulukon tietoja muuttujina.

5 Testausoperaatioiden optimointi: roolit, tehokkuus ja kustannukset

5.1 Testaajan rooli

Ohjelmistotestauksessa erilaisilla rooleilla on ratkaiseva merkitys kun halutaan varmistaa ohjelmistosovelluksen laatu ja tehokkuus. Näiden roolien ymmärtäminen on olennaista tehokkaalle tiimityölle ja projektin onnistumiselle [21].

Testausjohtaja tai päällikkö:

- Suunnittelee ja valvoo testaustoimintoja.
- Varmistaa, että tiimillä on tarvittavat resurssit.
- Koordinoi testausta ohjelmistokehityksen kanssa.
- Valmistelee tilanneraportteja ja pitää yhteyttä asiakkaisiin.

Testausinsinöörit (QA&QC testers):

- Lukevat ja ymmärtävät testivaatimukset.
- Päättävät, miten testaus tehdään kerätyn tiedon perusteella.
- Ilmoittavat testausjohtajalle tarvittavista resursseista.
- Luovat testitapauksia, priorisoivat toimintoja ja suorittavat testit.
- Raportoivat puutteista ja tekevät regressiotestauksen koodimuutosten jälkeen.

Ohjelmistotestaaja:

- Suunnittelee testitilanteita käytettävyystesteihin.
- Suorittaa testauksen, analysoi tulokset ja raportoi havainnoista.
- Vaihtaa tietoa asiakkaiden kanssa tuotetarpeista tai muutoksista.
- Luo testidokumentaatiota ja osallistuu testauksen läpikäynteihin.

Ohjelmistotestauspäällikkö:

- Edustaa tiimiä eri osastojen kokouksissa.
- Vaihtaa tietoa asiakkaiden kanssa ja rekrytoi testaushenkilöstöä.
- Suunnittelee testaustoimintoja, laatii budjetteja ja arvioi resurssitarvetta.
- Valitsee sopivat testausvälineet ja parantaa jatkuvasti testiprosesseja.
- Tarkistaa vaatimusten laatua ja ylläpitää testauksen jäljitettävyyttä.

Ohjelmistotestiautomaattori:

- Suunnittelee testiprosesseja ja -tapauksia automaattista testausta varten.
- Kehittää uudelleenkäytettäviä ja standardoituja automaattisia testiskriptejä.
- Varmistaa yrityksen määrittelemiin automaatiostandardeihin noudattamisen.
- Päivittää taitojaan sopeutuakseen automaation muuttuviin trendeihin.

5.2 Tehokkuus

Tehokkuus testauksessa on elintärkeää, jotta testaustoimet kyetään suorittamaan nopeasti ja tarkasti. Tehokkaassa testausprosessissa aikaa ja resursseja käytetään optimaalisesti, mikä johtaa parempaan testikattavuuteen ja virheiden tehokkaampaan tunnistamiseen. Tehokkuus tarkoittaa samalla sitä, että tiimin jäsenten välillä on selkeää ja avointa viestintää, jotta yhteistyö on sujuvaa ja projekti etenee ilman turhia viivästyksiä. Tehokkuus ei ole pelkästään ajan säästämistä, vaan myös oikeiden prioriteettien asettamista ja tiimityön sujuvuutta. Näin varmistetaan, että testausprosessi on sekä nopea että laadukas [22].

5.3 Manuaalisen ja automaattisen testauksen kustannukset

Testaus on olennainen vaihe ohjelmistokehityksessä, ja sen tavoitteena on varmistaa lopputuotteen laatu ja luotettavuus. Tästä syntyy erilaisia kustannuksia kuten henkilöstö, työkalut, infrastruktuuri ja aika. Testauskustannusten tehokas hallinta on ratkaisevaa laadun ja budjettirajoitteiden tasapainottamiseksi [23].

5.3.1 Manuaalisen testauksen kustannukset

Manuaalisen testauksen kustannuksista suurimman menoerään tuottavat henkilöstökustannukset. Testauksessa tarvitaan inhimillisiä resursseja mikä lisää kustannuksia suuremmissa projekteissa. Testaaminen voi viedä aikaa. Erityisesti toistuvat tehtävät tai laajat projektit vaativat paljon aikaa. myös inhimilliset virheet ovat todennäköisempiä mikä johtaa lisääkustannuksiinvirheiden korjaamiseen ja uudelleen testaamiseen [24].

5.3.2 Automaattisen testauksen kustannukset

Automaattinen testaus aiheuttaa alkuinvestointeja työkaluihin ja kehyksiin. kustannuksia syntyy myös skriptien päivittämisestä ja ylläpidosta sovelluksen kehittyessä. Testaus on nopea ja säästää aikaa pitkällä tähtäimellä [24].

5.3.3 Kustannusten Vertailu

Näitä kahta testaustapaa vertailtaessa voidaan todeta, että automaattinen testaus vaati suuremman alkupääomaan, mutta tuo pitkällä aikavälillä säästöjä erityisesti suurissa projekteissa ja toistuvissa tehtävissä. Se vähentää manuaalista työtä ja siihen liittyviä kustannuksia.

Manuaalinen testaus taas voi olla kustannustehokasta pienissä projekteissa tai tutkimus testeissä. Testaus tapa lisää henkilökustannuksia, koska testaus voi vaatia suuremman tiimin. Automaattinen testaus mahdollista tiimin keskittymiseen strategisempiin näkökohtiin eikä vaati yhtä suurta henkilöstöä. Lisäksi automaattisen testauksen etu manuaalisen testauksen nähden on nopeus, tehokkuus ja ajan säästö. Valinta näiden kahden testauksen välillä tulisi tehdä huomioiden projektin koko monimutkaisuus ja tasapaino välittömien kustannusten ja pitkään aikavälin etujen välillä [25].

6 Yhteenveto

Opinnäytetyössäni tutkin manuaalisen ja automatisoidun testauksen eroja dynaamisten testausympäristöjen näkökulmasta. Tavoitteenani oli syventyä kummankin testausmenetelmän haasteisiin ja mahdollisuuksiin tarjota samalla arvokkaita näkökulmia kehittäjille ja testaajille. Työ kattaa laajasti testauksen perusteet, prosessit sekä erilaiset testityypit, erityisesti painottaen toiminnallista ja ei-toiminnallista testausta.

Manuaalista testausta tarkastellaan perinteisenä ja kärsivällisyyttä vaativana lähestymistapana. Toisaalta automatisoitu testaus esitellään tehokkaana, tietokoneavusteisena menetelmänä, jonka päätavoitteina ovat kustannusten alentaminen ja testaustaajuuden parantaminen. Vaikka automaation hyödyt sisältävät nopeamman suorituksen ja regressiotestauksen, sen haasteisiin kuuluvat alkukustannukset ja ylläpitokustannukset sekä erityisosaamisen tarve.

Opinnäytetyöni päättyy konkreettisiin käytännönohjeisiin jotka ovat tukena päätöksenteossa ohjelmistokehityksen testauskontekstissa. Tarkastelen erityisesti sitä, kuinka valita oikea testausmenetelmä projektin tarpeisiin, kun kriteereinä ovat budjetti, aikataulu ja projektin laajuus. Työni pyrkii valottamaan testauksen monimuotoisuutta ja auttamaan alan ammattilaisia tekemään informoituja päätöksiä testausstrategioiden suhteen.

Lähteet

1 ISTQB, Foundation Level Syllabus. v3.1.1. < https://www.istqb.org/certifications/certified-tester-foundation-level-v3-1>.

2 Techtarget, Software testing. < https://www.techtarget.com/whatis/definition/software-testing>.

3 TOOLSQA, Error, Defect, and Failure <https://www.toolsqa.com/software-testing/istqb/error-defect-failure/ >.

4 Foundations of Software Testing ISTQB Certification, 4th edition page10 [book]. < https://www.adlibris.com/fi/kirja/foundations-of-software-testing-9781473764798?gad_source=1 >.

5 ITPEDIA blogi, ohjelmiston testausperiaatteet < https://fi.itpedia.nl/2017/09/11/7-software-test-principes/ >.

6 IEEE Standard for Software and System Test Documentation. < https://ieeexplore.ieee.org/document/4578383 >.

7 geeksforgeeks, Software Testing< https://www.geeksforgeeks.org/unit-testing-software-testing/>.

8 SoftwareTestingHelp, Types Of Software Testing: Different Testing Types With Details, <https://www.softwaretestinghelp.com/unit-testing/>.

9 Atlassian, The different types of software testing. < https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing>

10 Software Test Types, tutorialRide.com. <https://www.tutorialride.com/software-testing/software-test-types.htm>

11 ISTQB, Foundation Level Syllabus. v3.1.1 (sivu 39). < https://www.istqb.org/certifications/certified-tester-foundation-level-v3-1>.

12 Guru99, Non-Functional Testing. <https://www.guru99.com/non-functional-testing.html>

13 Educba, White Box Testing. < https://www.educba.com >

14 ProgramsBuzz, change-related Testing. < https://www.programs-buzz.com/article/change-related-testing >

15 Book. Test Automation Fundamentals, Advance level specialist < https://www.amazon.com/Test-Automation-Fundamentals-Certified-Specialist-ebook/dp/B0BG28NWRV>

16 Book. Test Automation Fundamentals, Advance level specialist (sivu9).

17 Book. Test Automation Fundamentals, Advance level specialist (sivu10).

18 BrowserStack, Benefits of Automation Testing < https://www.browserstack.com/guide/benefits-of-automation-testing>

19 Javatpoint, Disadvantages of Automated Testing. < https://www.javatpoint.com/advantages-and-disadvantages-of-automated-testing>

20 Ranorex, Types of Automated Testing . < https://www.ranorex.com/blog>

21 Book. Test Automation Fundamentals, Advance level specialist (sivu40).

22 geeksforgeeks < https://www.geeksforgeeks.org/software-testing-techniques/> .

23 Scalac, Manual Testing Vs Automation Testing: How Much Does It Really Cost. < https://scalac.io/blog/manual-testing-vs-automation-testing/>.

24 QAcraft, QA Cost Comparison: Manual vs. Automated < https://qacraft.com/efficient-qa-cost-manual-vs-automated-testing/>

25 devtechnosys, Manual Testing vs. Automated Testing.< https://devtechnosys.com/insights/tech-comparison/manual-testing-vs-automated-testing/>