OAMK OULUN AMMATTIKORKEAKOULU

Jami Salmela

INTENSIIVIMALLINEN TESTAUS MODERNIN OHJELMISTOKEHITYKSEN TUKENA

INTENSIIVIMALLINEN TESTAUS MODERNIN OHJELMISTOKEHITYKSEN TUKENA

Jami Salmela
Opinnäytetyö
Syksy 2023
Tietojenkäsittely
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittely

Tekijä: Jami Salmela
Opinnäytetyön nimi: Intensiivimallinen testaus modernin ohjelmistokehityksen tukena
Työn ohjaaja(t): Pekka Ojala
Työn valmistumislukukausi ja -vuosi: Syksy 2023 Sivumäärä: 22

Opinnäytetyön aiheena oli tutkia, miten Intensiivimallinen testaus voi tukea ohjelmistokehitystiimejä ja voiko Intensiivin avulla lyhentää kehityksen Time to Marketia. Työ tehtiin Intensiivimallin kehittäjäyrityksen Prove Expertise Oy:n kanssa.

Työssä käsiteltiin ohjelmistokehityksen eri käsitteitä ja miten Intensiivi toimii vuorovaikutuksessa niiden kanssa. Käytiin läpi myös, miten perinteinen ja Intensiivimallinen ohjelmistotestaus eroavat toisistaan, ja mihin niitä voi soveltaa parhaiten. Intensiivi käydään läpi sen oman uniikin toimitustapansa takia ja avataan sen eri vaiheet.

Opinnäytetyössä käytettiin useita eri internet lähteitä eri ohjelmistokehityksen käsitteistä. Intensiivistä ei ole, pintapuolisia läpileikkauksia lukuun ottamatta, julkisia lähteitä. Työn alussa käydään läpi eri käsitteet ja mitä ne tarkoittavat. Kolmannessa luvussa käydään läpi erilaisia ongelmia, mitä ohjelmistokehityksessä voi tulla vastaan ja neljännessä luvussa tutkitaan, miten Intensiivi voi auttaa ongelmissa.

Asiasanat: Ohjelmistotestaus, ohjelmistokehitys, vesiputous, V-malli, ketterä kehitys, Intensiivi

ABSTRACT

Oulu University of Applied Sciences
Information Technology

Author: Jami Salmela
Title of thesis: Intensive model testing as a support for modern software development
Supervisor(s): Pekka Ojala
Term and year when the thesis was submitted: Autumn 2023
Number of pages: 22

In this thesis the aim was to explore how Intensive model testing can aid software development teams. One of the central points was can the Time to Market be shortened if development teams took advantage of the intensive. The Intensive model testing is created and used by Prove Expertise Oy and this thesis was written to deepen the understanding of how Intensive can be used in software development.

Different topics of software development and business around it were inspected. One of the major points was the differences between traditional software testing and the Intensive model testing and how to get the most out of both. Intensive model testing is explained thoroughly because of its unique characteristics.

A plethora of different sources have been used when explaining different topics to get a diverse look at all of them. As for Intensive, there is no great source and most of the written text is derived from writers expertise on the matter.

Keywords: Software testing, Software development, Waterfall, V-Model, Agile development, Intensive testing

SISÄLLYS

1 JOHDANTO 6
2 OHJELMISTOKEHITYS JA OHJELMISTOTESTAUS 7
2.1 Testaaminen ohjelmistokehityksessä 7
2.2 Ohjelmistokehitysmallit 7
2.2.1 Vesiputous 8
2.2.2 V-Malli 9
2.2.3 Ketterä kehitys 10
2.3 Time to market 11
2.4 Intensiivi 11
3 OHJELMISTOKEHITYKSEN HAASTEITA 12
3.1 Kehitysmallien ongelmia 12
3.2 Tasapainottelu laadun ja ajan välillä 13
3.3 Intensiivi kehitystiimien apuna 14
4 JOHTOPÄÄTÖKSET JA TULOKSET 16
5 POHDINTA 19
LÄHTEET 21

1 JOHDANTO

Sovelluskehitys on nyky-yhteiskunnalle elintärkeää ja sen taitajat tekevät lähes taukoamatta työtä sen laadun parantamiseksi. Kuitenkin vielä eri ohjelmistoja käytettäessä käyttäjä kohtaa ongelmia, mitkä haittaavat käyttöä, estävät käytön tai muuten vain aiheuttavat mielipahaa erinäisissä mittakaavoissa. Tähän on käytetty jo jonkin aikaa vastalääkkeeksi ohjelmistotestausta, minkä pohjimmainen tehtävä on löytää käyttöä haittaavat ongelmat.

Sovelluskehityksen ja testaamisen haitaksi on kuitenkin kehkeytynyt liiketoiminta. Suurin haaste tässä on aika ja asiakkaiden odotukset. Nämä kaksi asiaa ja niiden ohi mitoittaminen tekevät kehittäjien ja testaajien työn lähes mahdottomaksi ja niihin keksitään alati uusia ratkaisuja. Tässä työssä tarkastellaan yhtä vaihtoehtoa: Intensiivimallista testausta.

Intensiivimallinen testaus on Prove Expertise Oy:n kehittämä testausmuoto, minkä tarkoituksena on löytää ohjelmistojen pahimmat kipupisteet ja nostaa laatua erilaisten löydösten avulla. Prove on ohjelmistotestauksen ja laadun asiantuntijayritys, eikä omia ohjelmistoja ole tai kehitystiimiä, eli siis puhtaasti muiden yritysten tuotteita tutkiva testaustalo.

Opinnäytetyön kirjoittaja on ohjelmistotestaajana Provella ja hänellä on laaja kokemus Intensiivien tekemisestä. Intensiivimallinen ohjelmistotestaus on uniikki testaustyyli, minkä tekemiseen työntekijät koulutetaan Proven toimesta. Tällä viikon mittaisella testausrupeamalla on oma kulma, miten ohjelmistojen laatua tutkitaan.

Tässä opinnäytetyössä tutkitaan ohjelmistokehitystä ja mitä haasteita ohjelmistotestauksessa voi olla. Jos yksi suurimmista haasteista ohjelmistojen liiketoiminnassa on aika, voidaanko Intensiivin avulla lyhentää valmiin tuotteen kehittämiseen tarvittavaa aikaa?

2 OHJELMISTOKEHITYS JA OHJELMISTOTESTAUS

2.1 Testaaminen ohjelmistokehityksessä

Ohjelmistotestaus on tapa varmistaa, että tuote vastaa vaatimusmäärittelyjä ja on vapaa bugeista (ohjelmistovirheistä) (Hamilton 2023). Ohjelmistotestaus on ala IT-maailmassa siinä missä ohjelmointikin. Ohjelmistotestaajan tehtäviin kuuluu muun muassa löytää sovelluksesta sitä haittaavat ongelmat, logiikkavirheet ja bugit.

Ohjelmistotestaus ei mahdu yhteen malliin, eikä yhden testaajan tarvitse osata kaikkia testaustyylejä tai tapoja. Indeed on listannut 111 erilaista tapaa testata sovellusta (Indeed 2023), eikä määrä ole ainakaan vähenemässä tulevaisuudessa. Kaikista karkein tapa jakaa testaustavat on manuaalisiin sekä automatisoituihin testaustapoihin. Manuaalisessa testauksessa testaaja käyttää sovellusta itse ja pyrkii löytämään erilaiset ongelmat. Automatisoidussa testauksessa on kirjoitettu koodinpätkä, mikä suorittaa toimintoja sovelluksen syövereissä. Jos toiminto suoritetaan määrittelyjen mukaan, saadaan hyväksytty tulos, ja jos toiminto ei kykene menemään ongelmitta loppuun asti, testi antaa virheilmoituksen, jolloin bugi on löytynyt. Testausta voidaan jakaa myös esimerkiksi tietoturvaan, saavutettavuuteen tai käytettävyyteen. Jokaisella osa-alueella on omat työkalut, tiedot ja taidot.

Hyvin keskeinen tapa validoida ohjelmistoa on testitapaukset. Testitapaus on testaajien ohje, kuinka testata jokin tietty toiminnallisuus. Niissä on kerrottuna askelmerkit, tarvittava tieto ja oletettu lopputulos yksityiskohtaisesti (Bose 2023). Jos testitapaukset ovat hyvin laadittuja, voi sovellustestaaja näiden avulla löytää tuotteesta ristiriitaisuudet vaatimusmäärittelyiden kanssa.

2.2 Ohjelmistokehitysmallit

Ohjelmistokehitys on kasvanut alana todella nopeasti, ja tämä on johtanut prosessien ja metodologien hienosäätämiseen. Ohjelmistokehitysmallit ovat kehityksen tueksi rakennettuja valmiita viitekehyksiä.

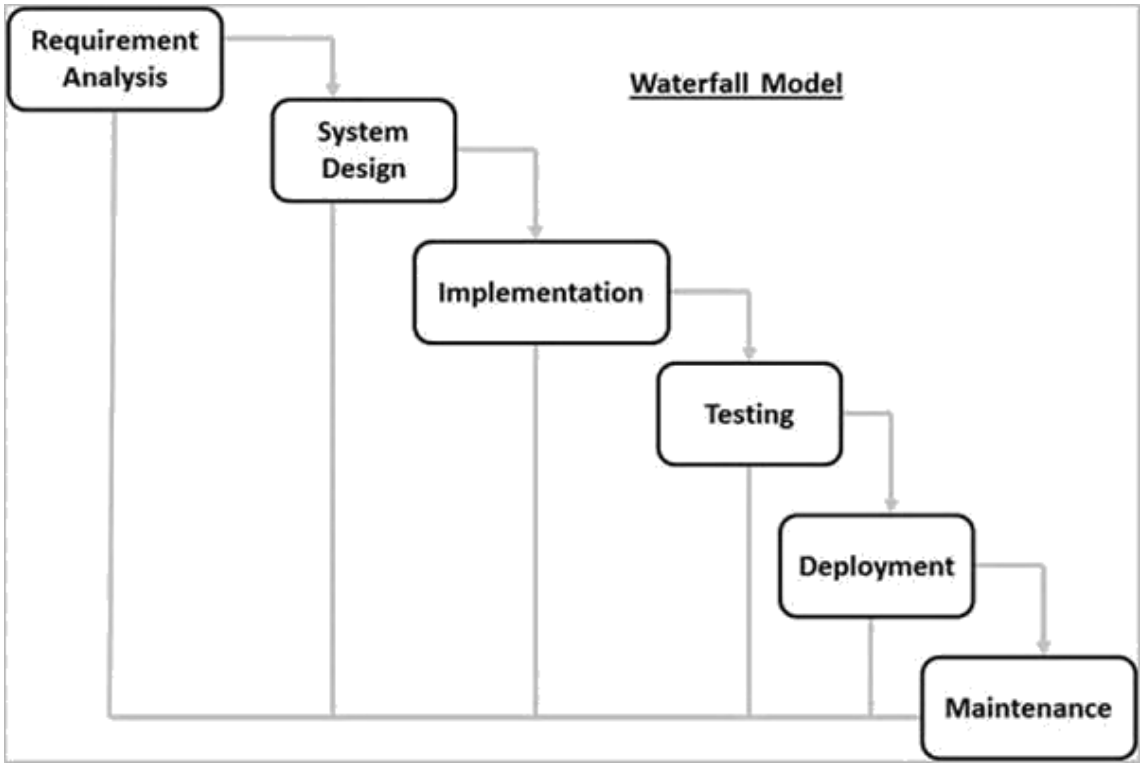
Ohjelmistokehitysmallit auttavat kehitystiimejä viitekehyksien avulla. Tiimien ei tarvitse miettiä, mitä tehdään ja missä järjestyksessä. Kehitystiimien täytyy vain valita omaan projektiin sopiva malli. Erilaiset mallit sopivat erilaisille projekteille koon tai kompleksisuuden perusteella.

Sovellustestaus on myös sisällytettynä näihin malleihin, yleensä projektin tai vaiheen loppupuolella. Tiukan aikataulun takia tämä saattaa johtaa hätiköityyn testaukseen tai testaussuunnitelman laiminlyömiseen.

2.2.1 Vesiputous

Vesiputousmallissa ohjelmistokehitys jaetaan ennalta määrättyihin vaiheisiin. Vaiheet suoritetaan toistensa jälkeen, kun edellinen vaihe on saatu valmiiksi. Vaiheita voivat olla esimerkiksi: suunnittelu, toteutus, testaus ja käyttöönotto.

Testaus on vesiputousmallissa yksi vaihe kehityskaaren lopussa. Jos aiemmissa vaiheissa tapahtuu myöhästymisiä, voi testaamiselle allokoitu aika murentua pois pikkuhiljaa. Tämä voi johtaa ongelmilla täytettyyn sovellukseen.



KUVIO 1. Vesiputousmalli (Tutorialspoint 2023a)

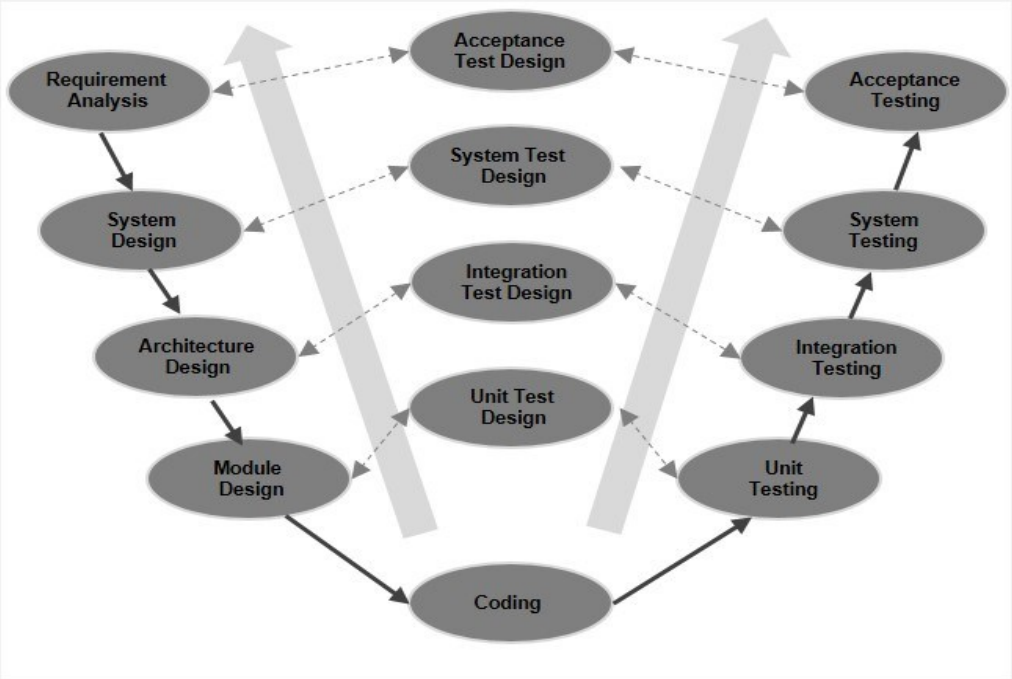
Kuviossa 1 on esitettynä vesiputousmallin kulku. Kuviosta nähdään, että testausvaiheen jälkeen on tuotteen käyttöönotto. Ennen käyttöönottoa kuitenkin testausvaiheessa löydetyt bugit on korjattava, ettei viallinen tuote pääse jakeluun. Aivan viimeisenä vaiheena on kuvattu ylläpito. Se pitää sisällään tuotteen päivittämisen, korjaamisen ja muokkaamisen tuotteen loppukäyttäjien tarpeiden mukaiseksi.

Vesiputous voi yksinkertaisuudessaan olla loistava työkalu pienille projekteille, missä on kevyt suunnittelutaakka. Ennalta päätettyjen vaiheiden takia isot projektit voivat olla haastavia pitää kasassa, jos vaatimukset vaihtuvat tai jotain ei ole otettu huomioon suunnitteluvaiheessa. On myös mahdollista, että suunnittelutyön alussa asiakas ei tiedä, mitä kaikkea tuotteeseen halutaan. Tällaiset suuremmat muutokset voivat olla hitaita implementoida, ja tämän vuoksi myös todella kalliita (Adobe 2022).

2.2.2 V-Malli

V-mallissa on tietyt kiveen hakatut vaiheet, kuten vesiputousmallissa. V-malli eroaa kuitenkin siinä, että jokaisen suunnitteluvaiheen aikana suunnitellaan myös vaihetta korreloiva testaus. Tämä edesauttaa ajan allokoinnissa myöhemmälle testaukselle, kun testien suunnittelutyö on jo tehty.

Kuviossa 2 on esiteltynä V-mallin kulku. Kuviosta voi nopealla vilkaisulla huomata monta yhteneväisyyttä vesiputousmallin kanssa. Malliin on lisätty paljon testaukselle allokoitua kohtaa ja kuvio muistuttaa kovasti V-kirjainta, mistä juontaa mallin kekseliäs nimi. Kehittämisen jälkeen mallissa alkaa testaaminen. Se alkaa yksikkötesteillä, eli testataan mahdollisimman pieniä testattavia osia tuotteesta, kuten esimerkiksi vaikka onnistunut kirjautuminen tuotteeseen. Tällaisella testaamisella voidaan tunnistaa bugit ja ongelmat jo aikaisessa vaiheessa. (Tutorialspoint 2023b). Yksikkötestejä seuraa integraatiotestaus, missä yksiköitä testataan yhdessä toisten yksiköiden kanssa kommunikaation ja yhteisymmärryksen takaamiseksi. Kolmantena vaiheena on järjestelmätestaus, missä testataan jo koko tuotetta. Ja neljäntenä, eli viimeisenä, vaiheena on hyväksymistestaus. Tässä vaiheessa tuotetta testataan määrittelyjä vastaan ja selvitetään, tekeekö tuote kaiken juuri niin kuin se on suunniteltu tuotekehityksen elinkaaren alussa. (sama).



KUVIO 2. V-Malli (Tutorialspoint 2023b)

2.2.3 Ketterä kehitys

Ketterä metodologia on projektin hallinnan lähestymistapa, missä projekti pilkotaan pienempiin osiin, ja painotetaan jatkuvaa kehitystä ja parantamista (Atlassian 2023). Ketterää metodologiaa käytetään useilla eri aloilla, kuten esimerkiksi henkilöstöhallinnossa ja informaatioteknologian tehtävissä. Ketterän kehityksen sisälle on kehitetty useita erilaisia malleja ja runkoja, joiden mukaan projektien tulisi edetä. Tunnetuimmat näistä ovat SCRUM ja Kanban (Santo 2023).

Ketterä kehitys eroaa vesiputous- ja V-mallista sen osiin jakamisella. Nämä osat, eli sprintit, yleensä keskittyvät yhteen pienempään kokonaisuuteen ja sen kehittämiseen sen sijaan, että tiimi tekee laajasti koko projektia eteenpäin kerralla. Ketterän kehityksen vetävänä ideana on sen itse määrätyn mittaisen jakson joustavuus. Kehittäjät kykenevät kesken kehityksen ottamaan huomioon asiakkaiden palautteet ja ottamaan seuraavalle projektin osalle palautteet kehitykseen. Myös testauksen pystyy ajoittamaan näille kehityksen sykleille, ja testauksesta nousseet ongelmat voidaan käsitellä uuden kierroksen suunnittelussa. Ketterä kehitys tuo joustavuutta myös testaukselle, eikä se välttämättä ole aina viimeisenä, vähän unohdettuna asiana.

2.3 Time to market

Time to market on aika, mikä alkaa tuotteen konseptoinnista ja loppuu tuotteen julkaisuun. Toisen määrittelyn mukaan Time to market alkaa silloin, kun tiimi alkaa työskentelemään ja loppuu kun ensimmäinen tuote on myyty (Carter 2023).

Time to market on yksi sovelluskehityksen monista suorituskykymittareista. Näiden mittareiden avulla pyritään asettamaan projekteille takarajat. Nämä takarajat asetetaan mahdollisimman lyhyeksi, jotta saataisiin mahdollisimman isot hyödyt uudesta tuotteesta. Nopea Time to market saa tuotteen markkinoille ennen kilpailijoita, ja jos yritys on markkinoilla ennen kilpailijoita, myynti alkaa ensimmäisenä, tuotot ovat isompia ja yritys saa isomman markkinaosuuden (Kleshchuck 2023).

2.4 Intensiivi

Intensiivimallinen testaus on Prove Expertise Oy:n käyttämä ja jalostama testausmalli. Intensiivi kestää viikon, minkä aikana asiakkaan sovellusta testataan halutulla tavalla. Testaus voi tapahtua esimerkiksi loppukäyttäjän näkökulmasta tutkivana testauksena, saavutettavuustestauksena tai tietoturvatestauksena.

Intensiivi on kulminoitunut kolmen vaiheen korkeapainetestausprojektiksi. Aluksi suoritetaan Intake-vaihe. Tässä vaiheessa hankitaan tarvittavat tiedot tuotteesta, rajataan testausalue sopivaksi ja käydään läpi käytännön asiat, kuten aikataulut ja ympäristöt. Toinen vaihe on testausvaihe, eli Intensiivi. Toisessa vaiheessa otetaan sovellus viikoksi Proven testaajien käsittelyyn. Viikon aikana keskitytään johonkin testauksen osa-alueeseen, kuten esimerkiksi käytettävyyteen, tietoturvaan tai suorituskykytestaukseen. Testausviikon jälkeen on kolmas vaihe, Followup. Intensiivin asiat käydään vielä läpi ja tehdään jatkosuunnitelmia testauksen suhteen.

Testaus tapahtuu usein joko juuri ennen kuin tuote on jakelussa niin sanotusti release-testauksena tai tuotannossa, kun sovellus on jo julkaistu. Ulkopuolinen testaus voi antaa tuotteen omistajille mielenrauhaa, kun iso osa bugeista on löydetty. Intensiivin isoin valtti on testaajan ulkopuolinen näkökulma, mikä on parhaiten hyödynnettynä kehityksen lopussa. Tämän takia kehityksen keskelle ei tehdä kovin usein Intensiivejä.

3 OHJELMISTOKEHITYKSEN HAASTEITA

3.1 Kehitysmallien ongelmia

Vaikka ohjelmistokehityksen erilaiset mallit ovatkin jo pitkällä, niiden mukana ovat jalostuneet erilaiset ongelmat. Nämä vaativat nykyisellään vielä ajattelua fiksummilta ihmisiltä, että saataisiin kansalaisille parempilaatuisia sovelluksia.

Sogetin blogissa (Sogeti 2019) on listattuna ajankohdat, milloin testaus tulee lopettaa. Ensimmäisenä asiana on ajan loppuminen. Ajan loppuminen voi tulla kehitystiimeille nopeasti vastaan, jos kehitys tai tuotteen suunnittelu venyy pidemmälle kuin suunniteltu. Toisena listassa mainitaan testaajien testauskohteen vaihtuminen. Tämä ongelma tulee vastaan, jos sovelluskehitystiimillä ei ole omaa testaajaa, vaan testaajat ovat yksi tiimi yrityksen sisällä. Tällöin testaajat ovat allokoituna tietyllä viikolla tiettyyn projektiin, ja jos viikko loppuu liian pian ensimmäisen projektin kohdalla, on testaajien siirryttävä kuitenkin seuraavaan projektiin. Testaukselle allokoitu aika voi olla myös vähissä, jos jokin aikaisempi vaihe venyy odotettua pitemmäksi. Tämä haaste tulee tiimeille vastaan varsinkin, jos käytössä on jokin kehitysmalli, jonka vaiheet ovat niin sanotusti kiveen hakattuja kuten esimerkiksi vesiputousmalli.

Loppuvaiheessa tapahtuva testaus voi johtaa myös isoihin aikaongelmiin. Jos tuotetta ei ole testattu muuten kuin kehittäjien pikatesteillä tuotetta koodatessa, voi tuotteesta löytyä yllättäväkin määrä bugeja testauksen alkaessa. Testauksen isoin bugihaavi on erilaisten käyttöskenaarioiden testaaminen. Kehityksessä usein testataan vain, että tuote ”toimii” eikä miten sitä voi käyttää eri lailla. Erilaiset käyttötavat nostavat bugien määrää huomattavasti, ja kaikki käyttöä haittaavat tai jopa käyttöä estävät bugit täytyy korjata ennen tuotteen julkaisua. Yllättävä iso bugimäärä tuottaa paljon korjaustöitä kehittäjille, ja jos julkaisupäivämäärästä ei voi tinkiä, on tiimin uhrattava aikaansa kehittämiselle tai tuote julkaistaan keskeneräisenä.

Tuotteen julkaisupäivämäärän lähestyessä, on tuotteen oltava julkaistavassa kunnossa. Kehittäjät ovat kehittäneet tuotteen hyväksyttävään kuntoon, eikä isompia bugeja enää löydy ja tuote on pyörinyt kehitystiimin kierteissä jo kauan ja se on koko tiimille tuttu. Tuotteen tunteminen voi kuitenkin

johtaa tietynlaiseen sokeuteen sen heikkoja kohtia kohtaan. Kehityskaaren lopussa ongelmat keskittyvät yleensä käyttäjäkokemukseen, jos tuote on monimutkainen ja monen asian osaaminen on ”tietäjät tietävät” -akselilla. Kehitysmalleihin ei aina ole sisällytettynä loppukäyttäjän näkökulman hankkimista, joten voi olla vaikea tietää ennen julkaisua, kuinka haastavaa tuotetta on oikeasti käyttää.

Bugien korjaaminen on kalliimpaa, mitä myöhemmin se löydetään kehityskaaren aikana (Sanket 2019). Jos ongelma löytyy jo kehitysvaiheessa, on koodi vielä tuoreena mielessä ja bugin korjaamiseen ei kulu niin paljoa aikaa. Testausvaiheessa löytyvät bugit täytyy kirjata ylös ja sen jälkeen ne täytyy käsitellä ja ohjata oikealle kehittäjälle. Vielä tämän jälkeen kehittäjän täytyy toistaa bugi omassa ympäristössään. Kaikista vaikeinta on, jos bugi löytyy vasta tuotannossa. Samat askelmerkit kuin testausvaiheessa löytyneiden ongelmien lisäksi täytyy pitää mielessä se, että muutokset tuotteeseen voi luoda lisää bugeja paikkoihin, mihin niitä ei odottaisi. Sen lisäksi että ongelmat ovat kalliita korjata, tuotannosta löytyvät bugit alentavat tuotteen arvon tuntua. Jos käyttäjä löytää monta bugia käyttösession aikana, voi tämä alkaa ärsyttämään, ja käyttäjä saattaa lopettaa tuotteen käytön, jos vastaava tuote löytyy kilpailijalta.

Projektin alussa kannattaa suunnitella huolellisesti ja jättää varaa virheille, koska niitä tapahtuu (Bąk 2023). Voikin olla hyvä jättää kehityskaaren suunnitelmaan vähän tyhjää tilaa ja olettaa, että jossakin vaiheessa jokin menee vikaan. Ohjelmistokehitys vaatii todella huolellista suunnittelua, että tuote saadaan kunnialla maaliin, eikä kenelläkään epävarmuuden tunteita ohjelmiston laadusta.

3.2 Tasapainottelu laadun ja ajan välillä

Time to marketia ajaa nopeammaksi muun muassa asiakkaiden halu saada uusia ominaisuuksia tai ylemmän hallinnon painostus saada tuote julkaisuun (Carter 2023). Liian nopeasti lähestyvä julkaisupäivämäärä voi johtaa tuotteen laajuuden uudelleen määrittämiseen. Tämä on välttämätöntä, jos nopea julkaisu on tuotteen menestykselle pakollista. Tällöin joudutaan tekemään valintoja ajan ja laadun välillä.

Laadusta tinkiminen voi johtaa tyytymättömään asiakaskuntaan. Asiakkailta voi pyydellä anteeksi loputtomiin, mutta se ei auta, kun ensimmäinen käyttökerta jää viimeiseksi. Asiaa voi paikata laastareilla, joita kutsumme päivityksiksi. Internet on mahdollistanut tuotteiden ja sovellusten päivittämisen julkaisupäivämäärän jälkeen. Jos laatua voi parantaa jälkikäteen, niin luonnollisesti Time to marketia voi lyhentää, ja voittoja hilata ylöspäin saavuttamalla markkinoille tuonnin ykkössijan.

Jos tuote on laaja ja siinä on paljon toimintoja, mutta ne eivät toimi kunnolla, ei välttämättä ole kannattavaa pyrkiä olemaan ensimmäinen markkinoilla. Joskus voi olla hyödyllistä tehdä tuotteesta MVP (Minimum Viable Product) eli tuote, missä on perustoiminnallisuudet, sitä voi käyttää, mutta siinä ei ole paljoa toiminnallisuuksia. MVP:n avulla tuotteen voi saada markkinoille ensimmäisenä ilman, että se on bugien ja ongelmien riivaama. Tuotteeseen pystyy lisäämään toimintoja myöhemmin päivitysten avulla. Tämä voi olla yksi ratkaisu Time to Marketin nopeuttamiseksi ilman, että tarvitsee tinkiä laadusta.

3.3 Intensiivi kehitystiimien apuna

Jos testauksen suurin haaste on aika, auttaako testaajien lisääminen? Intensiivin avulla sovelluskehitystiimit saavat lisää testausresursseja käyttöönsä. Jos tiimeillä on tietty määrä testitapauksia, joiden tulee olla hyväksyttyjä ennen kuin sovelluksen voi julkaista, niin silloin testaajien lisääminen auttaa. Jos sovellus voidaan julkaista ilman mitään isompia määrittelyjä, niin silloin välttämättä useampi testaaja ei nopeuta julkaisua. Jos kehitystiimeillä on halu saada mielenrauha bugien puolesta tai jos johtoporrasta askarruttaa sovelluksen tietoturva, voi viikon mittainen tutkiva testaus olla paikallaan.

Intensiivin heikot paikat ovat ehdottomasti isot ja monimutkaiset sovellukset, koska niiden sisäistämiseen menee todella paljon aikaa. Viikon testauksessa ei ehdi tehdä syväluotaavaa tutkimusta sovelluksen mielenmaisemasta, mutta kevyemmästä sovelluksesta saa todella hyvän vaikutelman siitä, mitä käyttäjät kokevat ensimmäisellä kerralla käyttäessään tuotetta ja jopa minkälaisia ongelmia pitkäaikaiset käyttäjät voivat kohdata. Intensiivi on myös hyvin pitkälti tutkivaa testausta, joten todistusaineisto voi olla vähäistä. Sovellukset, jotka vaativat suurta todistustaakkaa, kuten terveysteknologian tuotteet, voivat olla myös haastavia Intensiiville. On ymmärrettävää, että jos sovelluksella voi olla terveydelle haitallisia vaikutuksia, on oltava paljon aineistoa tuotteen toiminnan toimivuuden puolesta.

Intensiivin vahvoihin puoliin kuuluu ammattimaiset tutkivat testaajat. Hyvä tutkiva testaaja osaa omaksua monenlaisten käyttäjien käytöstavat ja tehdä käyttäjien kohtaamista ongelmista kattavat raportit. Tämä on valtavan arvokasta kehitystiimeille, varsinkin jos sovellusta ei ole vielä julkaistu. Näiden raporttien perusteella tiimit kykenevät kehittämään sovellustaan parempaan suuntaan. Intensiivi on loistava väline release-testaukseen, mutta se toimii oivasti myös jo julkaistun sovelluksen testaamiseen. Tällöin intensiivi toimii niin sanotusti ”valorakettina” sovellustiimille. Jos sovellus on ollut olemassa jo pitkään, ja sitä kehitetään jatkuvasti, voi tuotteessa olla pesiytyneenä jo useita tuntemattomia bugeja. Intensiivin avulla kehitystiimit saavat tuotteesta löytyvät ongelmat listattuna ja samalla heille valaistaan sovelluksen nykytilanne.

Erilaiset testauksen tyylit tarvitsevat laajaa osaamista. Jos perinteinen testaaja tekee esimerkiksi kerran vuodessa jotain tietyntyyppistä testaamista, ei siitä kerry osaamista pitkälle aikaa. Intensiivin erilaiset testauslajit voivat auttaa kehitystiimejä tässä. Laitekattavuustestaus tai tietoturvatestaus voivat olla hitaita testata, jos niitä ei tee usein. Hitauden lisäksi voi jäädä huomaamatta asioita, mitkä kokeneelta testaajalta tulee tarkistettua lähestulkoon automaattisesti. Intensiivimallinen testaus voi tukea kehitystiimejä tällaisissa asioissa. Kehitystiimi voi päättää, miten ohjelmistoa testataan ja saa siihen aina asian osaavan testaajan töihin.

4 JOHTOPÄÄTÖKSET JA TULOKSET

Intensiivin pääasiallinen tarkoitus on löytää testattavan kohteen laadun kipupisteet, ei niinkään luoda kattavaa testauksen todistusaineistoa. Jos tuote vaatii kattavan todistusaineiston toimivuudesta, on kehitystiimin kasattava se itse. Todistusaineiston kasaaminen vaatii paljon työtä ja aikaa, joten tässä asiassa ei ole suurta vaikutusta TTM:n kanssa. Bugien löytäminen auttaa kuitenkin itse testaustyön kanssa. Kun kehitystiimi saa löydöksistä laadukkaat raportit, voidaan korjaaminen aloittaa pikaisesti ja kehitystiimin omat testaajat voivat keskittyä testitapausten luomiseen. Testitapauksilla saadaan aikaan todistusaineistoa. Jos testitapaukset saadaan hyväksytysti läpi, saadaan raportille merkintä toimivasta toiminnallisuudesta. Tapaukset voidaan tehdä joko manuaalisesti, tai kehityksen rauhoittuessa voidaan tehdä niille automaatio, mikä suorittaa tapaukset automaattisesti ja niistä voidaan tehdä raportit.

Tällaisia toimivuuden takaavia raportteja ei intensiivistä saa, mutta laadun kipupisteiden löytyminen ja niiden huoltaminen nostaa tuotteen laadun tuntua huomattavasti. Intensiivistä saadut raportit voivat kuitenkin toimia nykytilanteen valottajina. Raporteissa käydään läpi tuotteen hyvät ja huonot kohdat, sekä miten löydökset vaikuttavat niihin. Nämä raportit toimivat hyvin tilanteessa, kun tuotteen asiakkaille täytyy selventää nykytilanne. Raporteista on helposti nähtävissä, miten laatua saadaan parannettua ja tätä kautta tuotteesta käyttäjäystävällisempi. Palautteet intensiivistä ovat selkeämpiä kuin yksityisiltä henkilöiltä, eli tuotteen käyttäjiltä. Käyttäjien palautteet voivat olla turhautuneisuuden värittämiä ja pahimmillaan olla kahden sanan kommentteja ilman isompaa tarkoitusta. Tarkat selostukset ohjelmiston heikoista kohdista ovat olennaisia kehityksen kannalta.

Jos tuotteen toimiminen on kehitystiimin vastuulla, ei intensiivi välttämättä lyhennä tuotteen Time to Marketia. Intensiivi tuottaa jopa lisää tekemistä kehitystiimille. Kun löydökset raportoidaan, ne aiheuttavat toimia koko tiimille. Löydöksien korjaaminen ja korjausten todentaminen vie useita tunteja ja samalla pidentää Time to Marketia. Tuntien käyttäminen on myös kallista ja voi olla kannattavaa pohtia vaihtoehtoa Minimum Viable Productista. Jos tuotteesta viedään markkinoille MVP, voidaan tätä parantaa päivitysten kautta ja samalla tasapainotella laadun ja ajan kanssa. Tämäkin on hyvä pitää kohtuullisena, koska on hyvin turhauttavaa, jos joka viikko täytyy ladata ja asentaa uusi päivitys. Useat päivitykset myös tuovat riskiä uusiin bugeihin, mitkä eivät jää kiinni testauksen seulassa.

Toinen lähestymistapa Intensiivin kanssa voisi olla se, että jos kehitystiimissä on rajallisesti testaajia eikä käsiparit riitä tarpeeksi nopeaan testaukseen, voi Intensiivin avulla saada testauksen maaliin määräaikaan mennessä. Tai jos kehitystiimillä ei ole omia testaajia laisinkaan, on Intensiivi hyvä työkalu tuotteen laadun varmistamiseen sekä bugien löytämiseen.

Intensiivin testaaminen on hyvin pitkälti tutkivaa testaamista. Tutkiva testaus löytää ominaisuuksien katvepaikkoja todella hyvin. Testitapauksiin pohjautuva testaaminen varmistaa tuotteen toimivuuden, mutta rajatapaukset jäävät usein varjoon. Nämä varjoon jääneet ongelmat tulevat usein esille vasta pitkän ajan käytön jälkeen ja voivat johtaa käyttäjän turhautumiseen. Pienet rajatapaukset eivät kuulosta isolta ongelmalta, mutta kun niitä tulee vastaan useampia lyhyen ajan sisällä, tuntuu tuote hyvin vajavaiselta ja keskeneräiseltä. Intensiivissä tulee myös käytettyä tuotetta laaja-alaisesti, eli myös perustoiminnallisuuden ongelmat tulevat varmasti esille. Kehitystiimin omat testaajat tuntevat tuotteen läpikotaisin. Testitapauksilla testattaessa he pystyvät huomaamaan pienenkin ongelman toimivuudessa ja kykenevät tutkimaan testitapausten ympäriltä tehokkaasti. Jos tuntee tuotteen todella hyvin, voi olla vaikeaa asettua käyttäjän asemaan. Käyttäjä ei tunne tuotetta ollenkaan ja saattaa painella vääriä nappeja ja tehdä asioita epäloogisessa järjestyksessä. Epäloogisuus aiheuttaa rajatapausten ongelmia, jolloin päästään takaisin henkilön turhautuneisuuteen. Intensiivin tutkiva testaus voi kitkeä juuri tällaisia tilanteita, kun tiimin ulkopuolisella testaajalla ei ole samanlaista tuntemusta tuotteesta ja hän saattaa tehdä asiat epäloogisessa järjestyksessä.

Tuotekehityksen yksi haastavimmista vaiheista on määrittely. Se on dokumentti, missä on kaikki tieto tuotteesta, mitä se tekee ja miten sen oletetaan käyttäytyvän (Romani 2022). Määrittelyn on siis tiedettävä, miten tuotteen käyttäjä tulee käyttämään sitä. Asioiden näkeminen käyttäjän näkökulmasta on todella vaikeaa, koska käyttäjien skaala on usein laaja. Tämä aiheuttaa sen, että kun määrittely tehdään tietynlaiselle käyttäjälle, osa on kuitenkin jää tyytymättömäksi. Tuote hyötyy tällöin ulkopuolisen näkökulmasta ja Intensiivi voi sen tarjota. Ero käyttäjätestaukseen on se, että Intensiivin tekijä osaa asettua erilaisiin käyttäjärooleihin ja löytää erilaiset ongelmat tehokkaasti. Käyttäjätestauksessa saa oikeaa käyttäjädataa, mutta vain tietyntyyppiseltä käyttäjältä. Käyttäjän näkökulman hankkiminen on hidasta, eikä auta tuotteen markkinoille pääsyn kanssa, mutta voi antaa kehitystiimille uutta näkökulmaa siitä, miten käyttäjät tuotetta käyttävät. Ymmärrys käyttäjien ajattelutavoista on arvokasta tietoa tuotteen laatua ajatellen.

Ohjelmiston tekeminen määrittelyjen pohjalta voi olla haastavaa myös yhteisymmärryksen puolesta. Asiakkaan toiveista on vielä pitkä matka valmiiseen tuotteeseen ja ajatusten täytyy kulkea

”rikkinäisen puhelimen” läpi kehitystiimin eri henkilöiden kautta. Se mitä asiakas tarkoittaa ei ole aina sama, mitä tuotteen omistaja ymmärtää tai mitä kehittäjä tekee. Tämän takia testaajan on ymmärrettävä, mitä asiakkaalle on luvattu. Ulkopuolisen on helpompi ottaa kantaa erilaisiin käyttäjäkokemusten pulmiin, jos tietää, mitä tuotteen tulisi käyttäjän mielestä tehdä.

Määrittely laaditaan kehityksen alkutaipaleella, ja jos kehitys kestää yli vuoden, voivat ne olla jo ”vanhentuneita”, sillä teknologia kehittyy valtavalla vauhdilla. Eri asiat ovat trendikkäitä eri aikoina ja vuoden kehityksen jälkeen, ne eivät ehkä ole aivan yhtä hehkeitä. Intensiivien tekijät näkevät erilaisia sovelluksia ja ohjelmistoja laajalla skaalalla eri käyttötarkoituksista ja aikakausista. Laajan näkemyksen ansiosta on myös mahdollista antaa palautetta tuotteen tilasta nykypäivään peilattuna.

Intensiivi ei siis lyhennä tuotteen Time to Marketia, jos ohjelmisto ei vaadi korkeaa laatua. Intensiiviä voi käyttää työkaluna laadun nostamiseen, jos hinnan saa mahdutettua tuotekehityksen budjettiin. Laadun, ajan ja rahan välillä taisteleminen on iso urakka, mikä kalvaa nykypäivän ohjelmistokehitystä. Joidenkin mielestä on hyväksyttävää päästää bugeja ihmisten tuotteisiin, ja toiset puolestaan käyttävät mielellään rahaa laadun nostamiseksi. Molemmissa näkemyksissä taistellaan aikaa vastaan, ja ohessa vedetään köyttä rahan ja laadun välillä käyttäen saatuja hyötyjä ja omia arvoja vetiminä.

5 POHDINTA

Aiheina laadunvarmistus ja ohjelmistotestaaminen ovat lähellä sydäntäni. Laadukas sovellus tai nettisivu tuntuu hyvältä käyttää, mutta niitä ei välttämättä osata arvostaa. Jos taas ohjelmisto on vaikeasti ymmärrettävä ja bugien riivaama, ei valituksista ja mielensä pahoittamisesta tule loppua. Ongelmat ovat usein lopputulos riittämättömästä testaamisesta, ja tuntui mielekkäältä valita aihe, missä saa tutkia syitä ja ratkaisuja näihin. Testaushan ei ole ikinä valmis eikä kaikkia ongelmia välttämättä edes saada kiinni ennen tuotantoon vientiä. Olisi kuitenkin todella hienoa, jos ohjelmistokehityksen prosessi saataisiin hiottua niin hyväksi, että kaikki ”turhat” ongelmat saataisiin aina pois tuotteesta.

Laadukkaat tuotteet korreloivat myös suoraan todella moneen asiaan. Jos ohjelmisto on selkeä, ei tarvita kalliita korjauksia tai raskasta tukikoneistoa, ja lapsilla ei välttämättä menisi niin usein hermot vanhempiensa opastamisessa teknologian syövereissä. Ohjelmistojen ongelmat ovat kalliita ja voivat turhauttamisellaan vaikuttaa suoraan ihmisen mielialaan.

Opinnäytetyötä kirjoittaessa tuli koko ajan vastaan uusia käsitteitä ja oli haastavaa valita, mistä aiheesta kirjoittaa ja mistä ei. Moni aihe oli itselle aivan uusi, ja tuntui, että jatkuvasti oli uudessa tiedon kaninkolossa tutkimassa vähän aiheen vierestä. Tämä kuitenkin kehitti minua itseäni, kun opin uutta, mutta opinnäytetyön prosessia se ei ainakaan nopeuttanut. Intensiiviä täytyi myös tarkastella ihan uudelta kantilta. Ennen tätä työtä ei tullut ajateltua sen syvällisemmin, miten Intensiivi voisi sopia jonkin ohjelmistokehitystiimin agendaan. Tiesin jo ennen opinnäytetyön aloittamista, että ohjelmistokehityksessä on paljon asioita, mistä en ole ikinä edes kuullutkaan. Mutta tehdessä se tieto realisoitui, ja nyt on olo, että eihän tästä voi ikinä tietää kaikkea. Varsinkaan nyt, kun koko ajan keksitään jotain uutta ja parempaa.

Time to Market oli työn yksi keskeisimmistä käsitteistä. Tämä tuli vähän puun takaa aiheena, koska en ollut oikeastaan koskaan aikaisemmin kuullut käsitteestä. Aiheesta kun etsii tietoa, tulee lähinnä vain artikkeleita, mitkä on suunnattu managereille tai muille hierarkian yläpäässä oleville rooleille. Koska aika on tärkeä osa kehitystä, olisi ehkä hyvä vähän teroittaa asian tärkeyttä myös tekijöille. Koodaajien ja testaajien ei välttämättä tarvitsisi omata syvällistä ymmärrystä aiheesta, mutta selitys

ainaiselle kiireelle työssä olisi kiva olla jokin selitys. Jos tämän työn aihetta haluaisi jatkaa eteenpäin tai tehdä uuden työn aiheeseen liittyen, yksi lähestymisen kulma voisi olla Time to Marketin avaaminen aiheena kaikille kehitystiimin jäsenille.

Opinnäytetyön kirjoittaminen oli pidempi prosessi, kuin kehtaan myöntää. Työn tekeminen alkoi isolla määrällä opiskelua. Vaikka väitänkin olevani testaamisen ammattilainen, ei osaamista ohjelmistobisneksen puolelta tai perinteisestä testaamisesta ollut oikeastaan ollenkaan. Asiaa täytyi lukea paljon. Uusia oppeja ja oivalluksia tuli viikoittain, mikä johti myös oman osaamisen kyseenalaistamiseen ja samaan aikaan myös koin olevani päivä päivältä pätevämpi.

Kirjoittaminen tuntui todella työläältä alussa ja se johti vaikeuksiin huomion pitämisessä työssä. Paikoitellen täytyi pakottaa itseä kirjoittamaan edes yksi kappale kirjoitussessiota kohden. Työstämisen viimeisenä kuukautena kirjoittaminen oli jo oikeastaan ihan mukavaa. Tekemiseen löytyi kiva rytmi, ja havahduin yksi päivä töissä odottavani, että pääsen kirjoittamaan. Nyt kun työtä lukee, tekstin sujuvuuden voi havaita uudemmasta tekstistä.

Tunne työn lopputulosta kohti on kovin myönteinen. Vaikka tekstin tuottaminen olikin vaikeaa, eikä se paikoitellen ole parasta minua, ohjelmistokehityksestä opitut asiat edistivät osaamistani enemmän kuin osasin odottaa.

LÄHTEET

Adobe 2022. Waterfall Methodology: A Complete Guide. Hakupäivä 12.9.2023. https://business.adobe.com/blog/basics/waterfall.

Atlassian, 2023. What is the agile methodology? Hakupäivä 3.9.2023. https://www.atlassian.com/agile.

Bąk, Tomasz 2023. Software Development Planning - Perfect Project Plan in 10 Steps. Hakupäivä 28.9.2023. https://www.softkraft.co/software-development-planning/.

Bose, Shreya 2023. How to write test cases in Software Testing? Hakupäivä 19.9.2023. https://www.browserstack.com/guide/how-to-write-test-cases.

Carter, John 2023. Time To Market (TTM) Why it's important – 5 Ways to Reduce it. Hakupäivä 20.5.2023. https://www.tcgen.com/time-to-market/.

Hamilton, Thomas 2023. What is Software Testing? Definition. Hakupäivä 15.9.2023 https://www.guru99.com/software-testing-introduction-importance.html.

Indeed, 2023. 111 Types of Testing in Software. Hakupäivä 19.9.2023. https://www.indeed.com/career-advice/career-development/types-of-testing.

Kleshchuk, Sophie 2023. Time to Market (TTM) – What is it and why does it matter for my business? Hakupäivä 8.10.2023. https://enkonix.com/blog/time-to-market/.

Romani, Edoardo 2022. What is a Software Requirement Specification? Hakupäivä 30.10.2023. https://builtin.com/software-engineering-perspectives/software-requirement-specification.

Santo, Diogo Espírito 2023. Top 5 main agile methodologies: advantages and disadvantages. Hakupäivä 3.9.2023. https://www.xpand-it.com/blog/top-5-agile-methodologies/.

Sanket 2019. The exponential cost of fixing bugs. Hakupäivä 28.9.2023. https://deepsource.com/blog/exponential-cost-of-fixing-bugs.

Sogeti 2019. When should you stop testing? Hakupäivä 5.9.2023. https://www.uk.sogeti.com/content-hub/blog/when-should-you-stop-testing/.

Tutorialspoint 2023a. SDLC – Waterfall Model. Hakupäivä 5.9.2023. https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.html.

Tutorialspoint 2023b. SDLC – V-Model. Hakupäivä 5.9.2023. https://www.tutorialspoint.com/sdlc/sdlc_v_model.html.