Tampereen yliopisto

Lassi Kirvesmäki

TEKNISEN VELAN HALLINTA OHJELMISTOTUOTANNOSSA

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Huhtikuu 2025

TIIVISTELMÄ

Lassi Kirvesmäki: Teknisen velan hallinta ohjelmistotuotannossa
Kandidaattitutkielma
Tampereen yliopisto
Tieto- ja sähkötekniikan koulutus, Tietotekniikka
Huhtikuu 2025

Tämä kandidaatintutkielma kartoittaa teknisen velan hallintaa ohjelmistotuotannossa. Teknisen velan käsitettä, syntymekanismeja, vaikutuksia ja hallintakeinoja tarkastellaan. Teknisellä velalla tarkoitetaan ohjelmistoprojekteissa tehtäviä kompromisseja, jotka nopeuttavat kehitystä lyhyellä aikavälillä, mutta joihin liittyy pitkän aikavälin kustannuksia, kuten lisätyötä tai laadun heikkenemistä. Tutkielma perustuu kirjallisuuskatsaukseen, jossa analysoidaan alan keskeisiä julkaisuja, määritelmiä ja käytäntöjä.

Teknisen velan syyt jaotellaan teknisiin, epäteknisiin, organisaatiollisiin ja prosessiliitännäisiin tekijöihin. Sen vaikutukset ulottuvat ohjelmiston laatuun, kehitystiimin tuottavuuteen sekä projektin taloudelliseen ja strategiseen toimivuuteen. Velan luokittelussa erotellaan mm. suunnittelu-, testaus- ja dokumentointivelka, joilla kullakin on omat tunnusmerkkinsä ja hallintamenetelmänsä.

Hallinnan kannalta keskeisiä ovat velan tunnistaminen ja arviointi sekä strategiat kuten ehkäisy, priorisointi, säännöllinen seuranta ja systemaattinen takaisinmaksu. Tutkielma korostaa, että tekninen velka on väistämätön osa ohjelmistokehitystä, mutta sen hallinta vaatii jatkuvaa tietoisuutta, avointa kommunikaatiota sekä kontekstisidonnaisia ratkaisuja. Tehokas hallinta edellyttää resurssien kohdentamista ja prosessien integrointia kehityksen eri vaiheisiin, mikä parantaa ohjelmiston ylläpidettävyyttä ja vähentää pitkän aikavälin kustannuksia.

Tutkielmassa esitellään muutamia viitekehyksiä ja näkökulmia teknisen velan hallintaan, joita hyödyntämällä organisaatiot voivat parantaa tekniseen velkaan liittyvää riskienhallintaa ja prosessien tehokkuutta.

Avainsanat: tekninen velka, ohjelmistotuotanto, määritelmä, vaikutukset, hallinta

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

TEKOÄLYN KÄYTTÖ OPINNÄYTTEESSÄ

Ilmoitukseni mukaan olen käyttänyt opinnäytteessäni tutkielmaprosessin aikana seuraavia tekoälysovelluksia:

Scopus AI
Google Gemini

Tekoälysovellusten nimet ja versiot:

Scopus AI
Gemini 2.0 Flash

Käyttötarkoitus: Scopus AI lähteiden löytämiseen. Gemini kielen ja tekstin tarkastukseen. Lähteitä löytynyt Scopus AI:lla noin 10 kappaletta.

Osiot, joissa tekoälyä on käytetty:

2.2 Synty ja syyt
3.2 Mittarit ja tunnusluvut
4.2 Hallintastrategiat

Olen tietoinen siitä, että olen täysin vastuussa koko opinnäytteeni sisällöstä, mukaan lukien osat, joissa on hyödynnetty tekoälyä, ja hyväksyn vastuun mahdollisista eettisten ohjeiden rikkomuksista.

Sisällysluettelo

1 Johdanto 1

2 Tekninen velka käsitteenä 2

2.1 Määritelmä 2

2.2 Synty ja syyt 2

2.3 Vaikutukset ja seuraukset 3

3 Teknisen velan luokittelu 5

3.1 Teknisen velan tyypit 5

3.2 Mittarit ja tunnusluvut 6

4 Teknisen velan hallinta ja vähentäminen 8

4.1 Tunnistaminen ja arviointi 8

4.2 Hallintastrategiat 9

5 Yhteenveto 12

6 Viittaukset 13

1 Johdanto

Tekninen velka ohjelmistotuotannossa on ollut olemassa käsitteenä 90-luvulta lähtien, ja sen määritelmissä on ollut erilaisia vivahteita. Teknisen velan käsitteellä viitataan päätöksiin ja kompromisseihin, jotka tehdään ohjelmistokehittäjien toimesta ohjelmiston toimittamisen nopeuttamiseksi lyhyellä aikavälillä, mutta joilla voi olla haitallisia vaikutuksia pitkällä aikavälillä. Tämä käsite on saanut laajaa huomiota ohjelmistokehityksen yhteydessä, ja sen vaikutusta ohjelmistojen laatuun, ylläpidettävyyteen ja kehitystyön kustannuksiin on tutkittu laajasti. [1]

Tämän kandidaatintutkielman tarkoituksena on kartoittaa kirjallisuudessa esitettyjä teknisen velan määritelmiä, sekä tarkastella millaisia keinoja teknisen velan hallintaan ja vähentämiseen on olemassa. Näitä määritelmiä analysoidaan, jotta voidaan hahmottaa teknisen velan käsitteen moniulotteisuutta ja sen merkitystä ohjelmistokehityksessä.

Tutkielma lähestyy aihetta kirjallisuuskatsauksen avulla, jossa keskitytään alan merkittävimpiin julkaisuihin, jotka käsittelevät teknisen velan käsitettä. Tavoitteena on tunnistaa ja vertailla erilaisia määritelmiä sekä ymmärtää, miten tekninen velka liittyy laajempiin ohjelmistokehityksen haasteisiin ja käytäntöihin. Tämä tutkielma tarjoaa syvällisemmän katsauksen teknisen velan käsitteeseen ja sen merkitykseen ohjelmistokehityksessä.

Tutkielman tavoitteena on tarjota kattava kuvaus teknisen velan hallinnasta ohjelmistotuotannossa. Tutkielmassa esitellään erilaisia menetelmiä ja työkaluja teknisen velan hallintaan, ja tarjotaan käytännön neuvoja ohjelmistoprojektien vetäjille ja kehittäjille. Tutkimustuloksia voidaan hyödyntää ohjelmistoprojektien suunnittelussa ja toteutuksessa. Tutkimustulosten avulla voidaan parantaa ohjelmistotuotantoprosessien tehokkuutta ja laatua.

Seuraavaksi luvussa 2 käsitellään teknistä velkaa käsitteenä, alkaen sen määritelmästä sekä syistä, jotka johtavat sen syntyyn. Luvussa tarkastellaan myös teknisen velan erilaisia vaikutuksia ja seurauksia. Tämän jälkeen luvussa 3 syvennytään teknisen velan luokitteluun sekä siihen liittyviin mittareihin ja tunnuslukuihin. Luvussa 4 keskitytään teknisen velan hallintaan ja vähentämiseen, mukaan lukien sen tunnistaminen, arviointi sekä erilaiset hallintastrategiat. Lopuksi luvussa 5 esitellään työn yhteenveto.

2 Tekninen velka käsitteenä

Tässä luvussa käydään lävitse teknisen velan määritelmä, synty ja syyt sekä vaikutukset ja seuraukset.

2.1 Määritelmä

Tekniselle velalle on olemassa useita hieman erilaisia määritelmiä, mutta suurimmassa osassa niistä keskitytään taloudelliseen vertauskuvaan, sekä ”korkoon” jota tästä täytyy maksaa. Ensimmäisenä käsitettä käytti Ward Cunningham kuvatessaan uuden ohjelmistotuotteen toimittamista. Cunningham kuvasi teknisen velan alun perin analogiaksi rahoitusvelkaan. Hänen mukaansa tietyissä tilanteissa kehittäjät voivat ottaa teknistä velkaa tekemällä tietoisesti nopeita ja yksinkertaisia ratkaisuja saavuttaakseen lyhyen aikavälin tavoitteita, mutta nämä ratkaisut voivat aiheuttaa teknisiä ongelmia ja lisätyötä tulevaisuudessa. [2]

Teknisen velan korolla tarkoitetaan ylimääräistä työtä, joka aiheutuu puutteellisista ja vajavaisista ratkaisuista. Esimerkiksi voidaan ajatella, että uuden toiminnallisuuden toteutus veisi optimaalisessa tilanteessa 4 päivää. Järjestelmän arkkitehtuuri kuitenkin vaatii 2 ylimääräistä päivää erilaisine integraatioineen, joten toiminnallisuuden kehittämiseen kuluu 6 päivää.

Cunninghamin tavoitteena oli herättää tietoisuutta teknisen velan vaikutuksista ja kannustaa kehittäjiä käsittelemään sitä aktiivisesti. Hän korosti, että teknistä velkaa tulisi hoitaa ja maksaa takaisin samalla tavalla kuin taloudellista velkaa, jotta vältettäisiin pitkän aikavälin haitat ja kustannukset. [2]

2.2 Synty ja syyt

Useat tekijät vaikuttavat teknisen velan kertymiseen, ja nämä voidaan jaotella neljään kategoriaan: teknisiin, epäteknisiin, organisaatiollisiin ja prosessiliitännäisiin tekijöihin. Näistä epätekniset tekijät ovat suppein kategoria; epäteknisten tekijöiden kuten kulttuurin, kommunikoinnin ja organisaation prosessien laiminlyönti voi johtaa teknisen velan syntyyn. [3]

Teknisistä tekijöistä versionhallintaan tallennettujen muutosten taajuus sekä koodirivien määrä ovat vahvasti yhteydessä teknisen velan määrään. Nämä voivat johtaa ohjelmistovirheisiin ja ylläpidon haasteisiin; jokainen koodirivi on riski itsessään. Näiden lisäksi huonot suunnittelu- ja toteutusvalinnat, eli koodin hajut, lisäävät teknistä velkaa. Ne ovat myös merkkejä huonoista koodauskäytännöistä ja huonontavat koodipohjan laatua [4] [1]. Projektin koolla ja kompleksisuudella on myös merkitystä. Nämä vaikeuttavat projektin ylläpidettävyyttä ja lisäävät siten teknisen velan määrää. [5] [6]

Organisaatiollisista tekijöistä koodikannan kehitykseen osallistuvien henkilöiden lukumäärän suuruus voi johtaa teknisen velan lisääntymiseen. Kasvavan tiimin myötä koordinaatio on vaikeampaa ja erilaisten koodaustapojen lukumäärä lisääntyy [5]. Kehittäjien kokemattomuudella on myös oma vaikutuksensa teknisen velan määrään; oikoteitä otetaan helpommin eikä seurata hyviä koodauskäytänteitä. Liian tiukat määräajat tiketeillä omalta osaltaan lisäävät teknisen velan ottoa. [7] [8]

Prosessiliitännäisistä tekijöistä kehityssyklien nopeus ja uusien versioiden julkaisunopeus ovat myös yhteydessä teknisen velan syntyyn [5] [9]. Testaukselle ja laadunvalvonnalle ei tällöin varata tarpeeksi aikaa, ja ohjelmiston laatu huononee. Riittämätön suunnittelu ja kehitysajan allokointi lisäävät myös teknisen velan määrää; jo menneiden syklien seurauksena syntyneet ohjelmistovirheet vievät enemmän aikaa ja uusien toiminnallisuuksien kehittämiseen käytetty aika on pienempi [10]. Tällaisessa ympäristössä vanhan koodin uudelleenkäyttö on erityisen riskialtista, varsinkin kun uusia ohjelmistonosia kopioidaan vanhasta ja muokataan vähän uuteen käyttötarkoitukseen. Bugit kertaantuvat ja tekninen velka kasvaa. [8]

2.3 Vaikutukset ja seuraukset

Teknisellä velalla on useita vaikutuksia ohjelmistoprojekteissa; vaikutukset voivat olla teknisiä, kohdistua ohjelmistokehitysprosesseihin ja kustannuksiin, tai ohjelmiston laatuun sekä ylläpidettävyyteen. Näiden lisäksi vaikutukset voivat kohdistua projektin taloudellisiin ja strategisiin lähtökohtiin.

Tekninen velka voi johtaa ei-optimaalisiin ratkaisuihin ohjelmistoa kehittäessä säästöjen takia, ja tällöin ohjelmiston jatkuva kehitys kärsii [11]. Tämä voi johtaa teknisen velan oton kierteeseen, kun lisääntyneet taloudelliset paineet pakottavat kehittäjät jatkamaan teknisen velan ottoa. Ohjelmiston laatustandardista poikkeaminen lasketaan myös tekniseksi velaksi, ja tällä voi olla samanlainen vaikutus jatkokehitystä ajatellen. [5]

Ohjelmistokehitysprosessin näkökulmasta tekninen velka vähentää kehittäjien tuottavuutta, kun teknisestä velasta aiheutuva lisätyö vie enemmän aikaa. Tällöin myös tiimin resurssien allokointi vaikeutuu, kun teknisestä velasta aiheutuvaa lisätyötä on vaikea arvioida [11]. Tämä lisätyö voi olla koodikannan uudelleen kirjoitusta, bugien korjailua tai vanhojen huonolaatuisten integraatioiden ylläpitoa [12]. Teknisellä velalla on myös alentava vaikutus ohjelmistokehitystiimin mielialaan ja suorituskykyyn. [13]

Ohjelmistoprojektin taloudellisesta ja strategisesta näkökulmasta teknisen velan laiminlyönnillä voi olla merkittäviä vaikutuksia, koska teknisen velan hallintaan vaadittava työmäärä kasvaa ajan myötä [11]. Ohjelmistoprojektissa joudutaan mahdollisesti priorisoimaan teknisen

velan hallintaa uusien toiminnallisuuksien ylitse, ja tätä on usein vaikea perusteella asiakkaalle tai projektin tilaajalle; parhaimmillaan teknistä velkaa voidaan hallita uusien toiminnallisuuksien kehityksen ohella. [14]

3 Teknisen velan luokittelu

Teknistä velkaa luokitellaan usein eri ulottuvuuksien ja velan tyypin perusteella. Esimerkiksi yhdessä tutkimuksessa ehdotetaan kolmea ulottuvuutta teknisen velan luokittelulle: asiakastyytyväisyysvaatimukset, luotettavuusvaatimukset, sekä teknologisen häiriön todennäköisyys [15].

3.1 Teknisen velan tyypit

Caglayan ja Özcan-Top esittelevät tutkielmassaan 10 eri teknisen velan tyyppiä, sekä 120 indikaattoria näille [16]. Nämä eri tyypit ovat suunnittelu-, testaus-, dokumentointi-, arkkitehtuuri-, infrastruktuuri-, käännös-, prosessi-, vika- sekä henkilöstövelka. Suunnitteluvelkaa syntyy suunnitteluvaiheen aikana, ja se ilmenee esimerkiksi puutteellisina vaatimuksina tai vajavaisesti määriteltyinä rajapintoina, jotka vaikuttavat ohjelmiston jatkokehitykseen. Testausvelka taas kertyy riittämättömistä tai puutteellisista testeistä, mikä kasvattaa virheiden riskiä ja hankaloittaa uudelleenkirjoitusta. Dokumentointivelka taas ilmenee puutteellisena tai vanhentuneena dokumentaationa, jolloin ohjelmiston ymmärtäminen ja ylläpito vaikeutuvat esimerkiksi puuttuvien koodikommenttien tai vanhentuneiden käyttöohjeiden vuoksi.

Ohjelmiston perusarkkitehtuurissa piilevät ongelmat muodostavat arkkitehtuurivelkaa, joka voi ilmetä rajoituksina skaalautuvuudessa tai suorituskyvyssä esimerkiksi monoliittisen rakenteen tai sekavien riippuvuuksien takia. Myös infrastruktuurivelka voi hidastaa kehitystä ja lisätä kustannuksia vanhentuneiden palvelimien tai puutteellisten verkkoyhteyksien vuoksi. Käännösvelka liittyy kääntämisprosessin ongelmiin, jotka voivat hidastaa kehitystä ja lisätä virheiden todennäköisyyttä.

Itse kehitysprosessissa ilmenevät tehottomuudet ja riskit muodostavat prosessivelkaa, joka voi johtua esimerkiksi huonosta kommunikaatiosta tai liian jäykistä työtavoista. Vikavelka on suora seuraus korjaamattomista ohjelmistovirheistä, jotka heikentävät ohjelmiston laatua ja käyttäjäkokemusta. Henkilöstövelka viittaa osaamisen, kokemuksen tai saatavuuden puutteisiin tiimissä, mikä voi vaikeuttaa ohjelmiston kehittämistä ja ylläpitoa. Kaikki nämä velkatyypit voivat yhdessä muodostaa merkittävän haasteen ohjelmistoprojektien onnistumiselle. Näitä teknisen velan eri tyyppejä voidaan hyödyntää teknisen velan tunnistamisessa, korjaamisessa, sekä riskien ja kustannusten optimoinnissa, ja ne on tiivistetty taulukossa 1 kuvauksineen ja esimerkkeineen.

Taulukko 1: Teknisen velan 10 eri tyyppiä [15].

Tyyppi	Kuvaus	Esimerkki
Suunnitteluvelka	Puutteet ohjelmiston suunnittelussa, jotka vaikeuttavat myöhempiä muutoksia ja ylläpitoa.	Epäselvät vaatimukset, huonosti määritellyt rajapinnat, väärät suunnitteluratkaisut.
Testausvelka	Riittämättömät tai puutteelliset testit, jotka lisäävät virheiden riskiä ja vaikeuttavat muutosten tekemistä.	Automatisoitujen testien puute, liian vähän testattuja polkuja, testien vanhentuminen.
Dokumentointivelka	Puutteellinen tai vanhentunut dokumentaatio, joka vaikeuttaa ohjelmiston ymmärtämistä ja ylläpitoa.	Puuttuvat kommentit koodissa, vanhentuneet käyttöohjeet, puutteellinen arkkitehtuurikuvaus.
Arkkitehtuurivelka	Ongelmat ohjelmiston arkkitehtuurissa, jotka rajoittavat sen skaalautuvuutta, suorituskykyä tai ylläpidettävyyttä.	Monoliittinen rakenne, riippuvuuksien sekamelska, huonosti määritellyt moduulit.
Infrastruktuurivelka	Ongelmat ohjelmiston käyttämässä infrastruktuurissa, jotka aiheuttavat suorituskykyongelmia, rajoittavat skaalautuvuutta tai lisäävät ylläpitokustannuksia.	Vanhentuneet palvelimet, puutteellinen verkkoyhteys, sopimattomat työkalut.
Käännösvelka	Ongelmat ohjelmiston kääntämisprosessissa, jotka hidastavat kehitystä ja lisäävät virheiden riskiä.	Monimutkainen käännösjärjestelmä, puutteellinen automaatio, riippuvuuksien hallinnan ongelmat.
Prosessivelka	Ongelmat ohjelmistokehitysprosessissa, jotka heikentävät tehokkuutta ja lisäävät virheiden riskiä.	Huono kommunikaatio, puutteellinen versionhallinta, liian jäykät prosessit.
Vikavelka	Korjaamattomat bugit ja virheet ohjelmistossa, jotka heikentävät sen laatua ja käyttäjäkokemusta.	Avoimet bugit, joita ei ole korjattu ajoissa.
Henkilöstövelka	Ongelmat henkilöstön osaamisessa, kokemuksessa tai saatavuudessa, jotka vaikeuttavat ohjelmiston kehittämistä ja ylläpitoa.	Liian vähän kokeneita kehittäjiä, osaamisvaje tiettyjen teknologioiden osalta, korkea henkilöstön vaihtuvuus.

3.2 Mittarit ja tunnusluvut

Teknisen velan mittaaminen ja dokumentointi on olennaista, jotta ohjelmistokehitystiimin resurssit voidaan kohdentaa tehokkaasti koodin uudelleenkirjoitukseen ja velan takaisinmaksuun. Prosessimittarit, kuten kehitysvauhti, bugien määrä ja vakavuus, sekä korjausaika voivat

olla merkkejä teknisestä velasta [3]. Dokumentoinnissa ja teknisen velan tunnistamisessa voi hyödyntää SonarQuben kaltaisia työkaluja. Nämä työkalut analysoivat koodia ja tunnistavat mahdollisia ongelmia, kuten koodin hajuja, monimutkaisuutta, monistumista sekä dokumentaation laatua. [16] [17]

Ohjelmiston ajonaikaiset indikaattorit, kuten suorituskyky ja resurssien käyttö, ovat myös tärkeitä teknisen velan vaikutusten ymmärtämisessä [18]. Esimerkiksi hidas latausaika tai toistuvat kaatumiset voivat olla merkkejä siitä, että ohjelmiston arkkitehtuurissa tai infrastruktuurissa on ongelmia. Näiden indikaattorien seuraaminen voi kuitenkin olla haastavaa, koska ne riippuvat vahvasti ohjelmiston suoritusympäristöstä. Lisäksi on tärkeää huomioida, että nämä ovat epäsuoria indikaattoreita, jotka voivat johtua muistakin tekijöistä kuin teknisestä velasta.

Teknisen velan tiheyden kehityksen seuraaminen, eli teknisen velan määrän muutoksen tarkastelu peräkkäisissä koodiversioissa, antaa kuvan siitä, miten velka kasautuu tai vähenee ajan myötä [19]. Tämä voi auttaa tunnistamaan trendejä ja tekemään päätöksiä uudelleenkirjoituksesta.

Lisäksi teknisen velan koronmaksujen seuraaminen, esimerkiksi teknistä velkaa koskeviin tiketteihin käytettyjä työtunteja seuraamalla, on hyödyllinen tunnusluku [20]. Tämä tosin vaatii manuaalista työtä, mutta onnistuu työtuntiseurannan ohessa. Tämä mittari kertoo suoraan, kuinka paljon resursseja käytetään teknisen velan maksamiseen. Vaikka tämä vaatii manuaalista työtä, se on arvokasta tietoa, joka auttaa priorisoimaan uudelleenkirjoitustoimenpiteitä.

4 Teknisen velan hallinta ja vähentäminen

Hallitsematon tekninen velka voi johtaa korkeampiin kehityskustannuksiin ja huonompaan ohjelmistolaatuun. Teknisen velan hallinta ja vähentäminen ovat siis oleellinen osa ohjelmiston laadun ja jatkokehitettävyyden ylläpitoa. Tämä voidaan jaotella teknisen velan tunnistamiseen ja arviointiin sekä hallintastrategioihin. Näiden lisäksi tarkastelemme myös teknisen velan vähentämisen hyötyjä ja haasteita.

Teknisen velan hallinnassa käytettävien mittarien ja tunnuslukujen tulisi olla kontekstisidonnaisia [15]. Usein eri mittareiden yhdistäminen on paras tapa saada kattava kuva teknisestä velasta. On myös tärkeää muistaa, että mittarit ovat vain yksi osa teknisen velan hallintaa.

4.1 Tunnistaminen ja arviointi

Teknisen velan tunnistaminen ja arviointi on monisyinen haaste, johon ei ole yhtä yksiselitteistä ratkaisua. Usein tarvitaan yhdistelmää erilaisista menetelmistä ja työkaluista. Kehittäjät ovat avainasemassa teknisen velan tunnistamisessa, sillä heillä on paras tuntemus koodista ja sen historiasta. Organisaation tulisi panostaa kehittäjien tietoisuuteen eri velkatyypeistä koulutuksen ja tiedon jakamisen avulla. Avoin kommunikaatio ja luottamus tiimissä ovat myös tärkeitä, jotta kehittäjät voivat vapaasti raportoida havaitsemistaan ongelmista. [21]

Koneoppiminen tarjoaa lupaavia mahdollisuuksia teknisen velan arvioinnin tarkentamiseen [22]. Koneoppimismalleille voidaan syöttää tietoja esimerkiksi koodin laadusta, muutoshistoriasta ja projektin ominaisuuksista. Näiden tietojen perusteella mallit voivat tunnistaa ja ennustaa teknistä velkaa [23]. On kuitenkin tärkeää muistaa, että koneoppiminen on työkalu, ei korvike asiantuntijoiden arvioille.

Arkkitehtuurinen tekninen velka on erityinen haaste tunnistamisen ja arvioinnin kannalta. Sen tunnistaminen voi olla vaikeaa, koska se vaatii kokonaisvaltaista näkemystä ohjelmistosta. Koodianalyysin, arkkitehtuurianalyysin, asiantuntija-arvioiden ja historiallisen analyysin yhdistäminen on usein paras tapa tunnistaa arkkitehtuurista velkaa. Hyvä dokumentaatio on myös tärkeä osa tätä prosessia. [24]

Teknisen velan tunnistamisessa ja arvioinnissa on edelleen haasteita, koska teknisen velan määrän ja vaikutusten mittaaminen on itsessään haastavaa ja subjektiivista. Lisäksi eri konteksteissa kehitetyt mallit eivät välttämättä ole yleistettävissä muihin konteksteihin, mikä vaikeuttaa riippumatonta todentamista. Aversano et al. [25] korostavat, että lisää tutkimusta tarvitaan, jotta voidaan kehittää luotettavampia ja yleistettävämpiä malleja teknisen velan kehityksen ennustamiseen.

4.2 Hallintastrategiat

Teknisen velan hallintaan on esitelty useita strategioita, jotka voidaan jaotella muutamiin lähestymistapoihin. Näitä ovat ennaltaehkäisy-, priorisointi-, seuranta-, ja takaisinmaksustrategiat.

Teknisen velan ehkäisy on kustannustehokkaampaa kuin sen takaisinmaksu jälkikäteen. Tällöin ohjelmiston kehitystiimin arkkitehdeistä lähtien täytyy olla tietoisia teknisen velan syistä ja hyvistä suunnittelu- sekä koodauskäytännöistä [26]. Teknisen velan tunnistaminen kehityksen aikaisessa vaiheessa voi auttaa sen tehokkaassa hallinnassa [27]. Tässä käytettäviä työkaluja voivat olla staattiset analyysityökalut kuten edellä mainittu SonarQube tai teknisen velan kehitystä ennustavat koneoppimismallit [28]. Teknisen velan hallinnan integrointi ohjelmistokehitysprosessiin voi auttaa hallitsemaan teknista velkaa mahdollisimman aikaisessa vaiheessa. Projektinhallinnan näkökulmasta teknisen velan juurisyitä voidaan myös ehkäistä; uusille toiminnallisuuksille ja koodimuutoksille tulee varata riittävästi aikaa jokaiseen vaiheeseen, eikä esimerkiksi katselmointeja tai testausta tule unohtaa. [29]

Teknisen velan hallinta priorisointistrategian avulla on tehokasta, mutta tätä varten pitää olla tarkasti määritellyt kriteerit [30]. Kriteerejä voivat olla mm. vaikutukset ylläpitokustannuksiin tai ohjelmiston suorittamiseen. Näitä voidaan valvoa esimerkiksi ohjelmiston suorituskyvyn säännöllisellä testaamisella sekä ohjelmiston suorituksen vaatimia resursseja seuraamalla [31]. Reboucas De Almeida et al. [32] esittelevät tutkielmassaan liiketoimintalähtöisen Tracy-viitekehyksen teknisen velan hallinnalle, ja siinä teknisen velan priorisointi on tärkeässä osassa. Tracy koostuu viidestä vaiheesta: tunnistaminen, arviointi, priorisointi, korjaussuunnitelma, seuranta ja arviointi. Priorisointi tehdään sekä teknisestä että liiketoiminnallisesta näkökulmasta. Tracyssa korostetaan sitä, ettei teknisen velan korjaus ole vain tekninen harjoitus, vaan sillä tulisi aina olla liiketoiminnallinen peruste. Plösch et al. [33] sen sijaan esittelevät portfoliomatriisin teknisen velan priorisoinnin tueksi. Heidän menetelmässään hyödynnetään suunnittelun parhaita käytäntöjä ja niiden mittaamista, jotta korjaustoimet voidaan järjestää tärkeysjärjestykseen sekä viestiä selkeästi sidosryhmille. Menetelmässä mittauksia verrataan vertailuarvoihin, ja saadut tiedot visualisoidaan kaksiulotteisessa matriisissa, jossa yhdistetään suunnittelukäytäntöjen tärkeys ja niiden toteutumisen taso. Tämä auttaa tunnistamaan ne alueet, joissa korjaukset tuottavat eniten hyötyä.

Seurantastrategiat ovat myös keskeinen osa velan hallintaa. Järjestelmällisellä usean mittarin ja tunnusluvun seurannalla voidaan päättää paras mahdollinen teknisen velan takaisinmaksuaika [30]. Useita viitekehyksiä ja työkaluja on kehitetty teknisen velan monitorointiin. Ciolkowski et al. [34] esittelivät ProDebt-prototyypin, jossa koneoppimismallia koulutettiin 5–

6 vuoden aikana kerätyllä laatu- ja työmäärätiedolla arvioimaan tiedostokohtaista teknistä velkaa annettuna ajanhetkenä.

Jotkin strategiat painottuvat takaisinmaksuun. Esimerkiksi Scrum-ohjelmistokehitysprosessissa teknistä velkaa seurataan kehitysjonossa työtehtävinä muiden toiminnallisuuksien ja ohjelmistovirheiden seassa ja maksetaan takaisin kirjoittamalla koodia uudelleen [35]. Teknistä velkaa siis työstetään samalla tavalla sprinttien aikana kuten mitä tahansa muita työtehtäviä. Teknisen velan takaisinmaksuun keskittyvät viitekehykset, kuten TAP [29], sisällyttävät teknisen velan takaisinmaksun projektinhallintaan.

Kyseinen viitekehys kehitettiin kolmen tavoitteen pohjalta: Ensimmäiseksi tekninen velka tulisi estää, jos se onnistuu aikataulun puitteissa. Toiseksi kun asetettu aikaraja johtaa tekniseen velkaan, tulee se maksaa takaisin pikimmiten. Kolmanneksi projektipäälliköiden tulisi pystyä seuraamaan teknistä velkaa tuotteissa. TAP-viitekehyksessä teknistä velkaa koskevat työtehtävät on luokiteltu 11 työtehtävätyyppiin ja nämä edelleen 4 kategoriaan; kaikki muut tiketit ovat toiminnallisia vaatimuksia (TV) ja kirjataan sellaisina. Määritellyt 4 kategoriaa ovat ylläpitotiketit (YPT), ylläpitoprojekti (YPP), teknisen velan tiketit (TVT), sekä purkamistiketit (PT). 11 työtehtävätyyppiä ovat ohjelmistovirhekorjaus, koodin uudelleenkirjoitus, kiertotie, dokumentaatio/testi, vajavainen toteutus, väliaikainen kytkin, arkkitehtuurimuutos, infrastruktuuri, konseptiotodistus, monitorointi ja suorituskyky. Näille kategorioille on erikseen määritelty vastuuhenkilöt, kirjaajat, prosessointiaika sekä riippuvuudet. Taulukossa 2 on tiivistetty TAP-viitekehyksen tehtävätyypit kategorioittain laajuuden mukaan.

Osa näistä tehtävätyypeistä voidaan lajitella vielä laajuuden mukaan. Bugikorjaukset ja suorituskykytiketit voi olla käyttäjän näkökulmasta näkyvissä, ja tällöin luokitellaan TV-tiketiksi; näkymättömät bugikorjaukset luokitellaan YPT-luokkaan. Koodin uudelleenkirjoitustiketit, arkkitehtuurimuutokset ja konseptitodistukset voidaan luokitella pieniksi tai suuriksi laajuuden perusteella; yksittäisen komponentin käsittely voi olla pienehkö ja menee YPT-luokkaan, kun taas kokonaisen ohjelmanosan käsittely kirjataan YPP-luokkaan.

Taulukko 2: TAP-viitekehyksen tehtävätyypit kategorioittain. [29]

Tehtävätyyppi	**Laajuus**	**Kategoria**
Bugikorjaus	Ei näkyvissä	YPT
	Näkyvissä	TV
Koodin uudelleenkirjoitus	Pieni	YPT
	Suuri	YPP
Kiertotie		TVT
Dokumentaatio/testi		TVT
Vajavainen toteutus		TVT
Väliaikainen kytkin		PT
Arkkitehtuurimuutos	Pieni	YPT
	Suuri	YPP
Infrastruktuuri		YPP
Konseptitodistus	Pieni	YPT
	Suuri	YPP
Monitorointi		YPT
Suorituskyky	Ei näkyvissä	YPT
	Näkyvissä	TV

Voidaan siis todeta, että teknisen velan hallinta on moniulotteinen prosessi, joka vaatii organisaatiolta proaktiivista otetta. Onnistunut hallinta edellyttää teknisen velan tunnistamista, sen vaikutusten arviointia, priorisointia liiketoimintalähtöisesti sekä systemaattista seurantaa ja takaisinmaksua osana normaalia ohjelmistokehitystyötä. Eri strategioiden ja viitekehysten hyödyntäminen voi auttaa organisaatioita löytämään juuri heille sopivat menetelmät teknisen velan tehokkaaseen hallintaan ja siten varmistamaan ohjelmistojen pitkäaikaisen laadun ja ylläpidettävyyden.

5 Yhteenveto

Tekninen velka viittaa ohjelmistokehityksen kompromisseihin, joilla pyritään nopeuttamaan kehitystä lyhyellä aikavälillä, mutta jotka voivat aiheuttaa pitkäaikaisia haittoja. Tutkielma esittelee erilaisia määritelmiä, joissa korostuu taloudellinen vertauskuva ja "koron" käsite, eli lisätyö, joka aiheutuu puutteellisista ratkaisuista.

Tekninen velka syntyy monista tekijöistä, jotka voidaan luokitella teknisiin, epäteknisiin, organisaatiollisiin ja prosessiliitännäisiin tekijöihin. Esimerkiksi huonot suunnitteluvalinnat, tiukat aikataulut ja riittämätön testaus lisäävät velan määrää. Vaikutukset voivat olla teknisiä, prosessuaalisia, taloudellisia ja strategisia, kuten alentunut tuottavuus ja kasvavat ylläpitokustannukset.

Tutkielmassa esitellään teknisen velan luokittelua eri ulottuvuuksien ja tyyppien mukaan, kuten suunnittelu-, testaus- ja dokumentointivelka. Lisäksi käsitellään mittareita ja tunnuslukuja, kuten koodianalyysityökaluja ja ajonaikaisia indikaattoreita, jotka auttavat velan tunnistamisessa ja arvioinnissa.

Teknisen velan hallinta ja vähentäminen on olennaista ohjelmiston laadun ja jatkokehityksen kannalta. Teknisen velan hallinta on myös tiukasti kontekstisidonnaista, ja hallintastrategiaa suunnitellessa tulee valita tai kehittää parhaiten eri sidosryhmiä palveleva kokonaisuus. Tutkielma esittelee erilaisia hallintastrategioita, kuten ehkäisyä, priorisointia monitorointia ja takaisinmaksua. Kehittäjien tietoisuus, avoin kommunikaatio ja koneoppimisen hyödyntäminen ovat tärkeitä osia velan hallintaa. Tutkielma korostaa, että teknisen velan hallinta on jatkuva prosessi, joka vaatii systemaattista lähestymistapaa ja resurssien kohdentamista.

6 Viittaukset

[1] M. Tufano, F. Palomba, G. Bavota, R. Olivetox, M. Di Penta, A. De Lucia ja D. Poshyvanyk, ”When and why your code starts to smell bad,” In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering,* Vol. 1, 403-414. IEEE.

[2] W. Cunningham, ”The WyCash portfolio management system.,” *ACM SIGPLAN OOPS Messenger 4.2 29-30*, (1992).

[3] H. Saeeda, M. O. Ahamd ja T. Gustavsson, ”A Multivocal Literature Review on Non-Technical Debt in Software Development: An Insight into Process, Social, People, Organizational, and Culture Debt.,” *e-Informatica Software Engineering Journal, 18(1), 240101., (*2024).

[4] J. Bedi ja K. Kaur, ”Understanding factors affecting technical debt,” *International Journal of Information Technology (Singapore),* 14.2 (2022): 1051-1060.

[5] J. S. De Jesus ja A. C. V. De Melo, ”Technical debt and the software project characteristics. A repository-based exploratory analysis,” *In 2017 IEEE 19th Conference on Business Informatics (CBI),* Vol. 1, 444-453. IEEE.

[6] D. Mancl ja S. Fraser, ”TD 2014: Workshop on technical debt in a world of big data and big teams,” *In Proceedings of the companion publication of the 2014 ACM SIGPLAN conference on Systems, Programming, and Applications: Software for Humanity,* 99-100.

[7] A. Aldaeej ja M. Alshayeb, ”Familiarity, Common Causes and Effects of Technical Debt: A Replicated Study in the Saudi Software Industry,” *Arabian Journal for Science and Engineering,* 49.3 (2024): 4459-4477.

[8] F. Bi, B. Vogel-Heuser, Z. Huang ja F. Ocker, ”Characteristics, causes, and consequences of technical debt in the automation domain,” *Journal of Systems and Software, Volume 204,* (2023): 111725.

[9] L. Buchmann ja K. Haki, ”Digital nudging for technical debt management: Insights from a technology-driven organization,” *Proceedings of the Annual Hawaii International Conference on System Sciences*, Online, (2020).

[10] B. Perez, C. Castellanos ja D. Correal, ”Developing a theory based on the causes of technical debt injection into software projects in Colombia,” *Journal of Physics: Conference Series*, San Jose de Cucuta, Vol. 1587. No. 1. IOP Publishing, (2020).

[11] Z. S. H. Abad, R. Karimpour, J. Ho, S. Didar-Al-Alam, G. Ruhe, E. Tse, K. Barabash ja I. Hargreaves, ”Understanding the impact of technical debt in coding and testing: An exploratory case study,” *In Proceedings of the 3rd International Workshop on Software Engineering Research and Industrial Practice*, 25-31.

[12] T. Besker, A. Martini ja J. Bosch, ”Impact of architectural technical debt on daily software development work - A survey of software practitioners,” *In 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA),* 278-287. IEEE.

[13] L. Torres, E. Fabricio, R. Cadena Martinez ja M. A. Pico Pico, ”Technical debt management in academic environments: perspectives and challenges of a software development team,” Data and Metadata, (2024); 3:.233

[14] M. Ciolkowski, L. Guzmán, A. Trendowicz ja F. Salfner, ”Lessons learned from the ProDebt research project on planning technical debt strategically,” *In Product-Focused Software Process Improvement: 18th International Conference, PROFES 2017, Innsbruck, Austria, November 29–December 1, (2017), Proceedings 18,* 523-534. Springer International Publishing.

[15] D. Caglayan ja Ö. Özcan-Top, ”Revisiting Technical Debt Types and Indicators for Software Systems,” *In Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing,* 834-841.

[16] M. I. Murillo ja M. Jenkins, ”Technical Debt Measurement during Software Development using Sonarqube: Literature Review and a Case Study,” *In 2021 IEEE V Jornadas Costarricenses de Investigacion en Computación e Informática (JoCICI),* 1-6. IEEE.

[17] M. A. Al Mamun, C. Berger ja H. Jörgen, ”Correlations of software code metrics: An empirical study,” *In Proceedings of the 27th international workshop on software measurement and 12th international conference on software process and product measurement*, 255-266. (2017).

[18] M. Ciolkowski, L. Guzmán, A. Trendowicz ja A. M. Vollmer, ”Challenges in assessing technical debt based on dynamic runtime data,” *Proceedings - 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2018*, Prague, (2018).

[19] M. A. A. Mamun, A. Martini, M. Staron, C. Berger ja J. Hansson, ”Evolution of technical debt: An exploratory study,” *In 2019 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, IWSM-Mensura 2019, Haarlem, The Netherlands, October 7-9, (2019),* 2476, 87-102. CEUR-WS.

[20] V. Sing, W. Snipes ja N. A. Kraft, ”A framework for estimating interest on technical debt by monitoring developer activity related to code comprehension,” *In 2014 Sixth international workshop on managing technical debt*, 27-30. IEEE.

[21] L. Pavlič, T. Hliš, M. Heričko ja T. Beranič, ”The Gap between the Admitted and the Measured Technical Debt: An Empirical Study,” *Applied Sciences 12.15 (2022): 7482.*

[22] V. Leonarduzzi, A. Martini, D. Taibi ja D. A. Tamburri, ”Towards surgically-precise technical debt estimation: Early results and research roadmap,” *In Proceedings of the*

3rd ACM SIGSOFT International workshop on machine learning techniques for software quality evaluation, 37-42.

[23] S. Akbarinasaji ja A. Bener, ”Adjusting the Balance Sheet by Appending Technical Debt,” *In 2016 IEEE 8th International Workshop on Managing Technical Debt (MTD)*, 36-39. IEEE.

[24] Y. Macit, G. Giray ja E. Tüzün, ”Methods for Identifying Architectural Debt: A Systematic Mapping Study,” *In 2020 Turkish National Software Engineering Symposium (UYMS)*, 1-6. IEEE.

[25] L. Aversano, M. L. Bernardi, M. Cimitile, M. Iammarino ja D. Montano, ”Forecasting technical debt evolution in software systems: an empirical study,” *Frontiers of Computer Science 17.3 (2023): 173210.*

[26] N. Rios, R. O. Spínola, M. G. De Mendonça Neto ja C. Seaman, ”A study of factors that lead development teams to incur technical debt in software projects,” *In 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 429-436. IEEE.

[27] C. Izuerita, G. Rojas ja I. Griffith, ”Preemptive management of model driven technical debt for improving software quality,” *In Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, 31-36.

[28] M. Mathioudaki, D. Tsoukalas, M. Siavvas ja D. Kehagias, ”Technical Debt Forecasting Based on Deep Learning Techniques,” *In Computational Science and Its Applications–ICCSA 2021: 21st International Conference, Cagliari, Italy, September 13–16, (2021), Proceedings,* Part VII 21, 306-322. Springer International Publishing.

[29] M. Wiese, P. Rachow, M. Riebisch ja J. Schwarze, ”Preventing technical debt with the TAP framework for Technical Debt Aware Management,” *Information and Software Technology 148 (2022): 106926.*

[30] L. F. Riberio, M. A. F. De Farias, M. Mendonça ja R. O. Spínola, ”Decision criteria for the payment of technical debt in software projects: A systematic mapping study,” *ICEIS 2016 - Proceedings of the 18th International Conference on Enterprise Information Systems*, 572-579, Rome, 2016.

[31] L. F. Ribeiro, N. S. R. Alves ja M. G. De Mendonca Neto, ”A strategy based on multiple decision criteria to support technical debt management,” *In 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA),* 334-341. IEEE.

[32] R. Reboucas De Almeida, C. Treude ja U. Kulesza, ”Tracy: A Business-Driven Technical Debt Prioritization Framework,” *In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 181-185. IEEE.

[33] R. Plösch, J. Bräuer, M. Saft ja C. Körner, ”Design debt prioritization: A design best practice-based approach,” *In Proceedings of the 2018 International Conference on Technical Debt*, 95-104.

[34] M. Ciolkowski, L. Guzmán, A. Trendowicz ja F. Salfner, ”Lessons learned from the ProDebt research project on planning technical debt strategically,” *In Product-Focused Software Process Improvement: 18th International Conference, PROFES 2017, Innsbruck, Austria, November 29–December 1, (2017), Proceedings 18,* 523-534. Springer International Publishing.

[35] G. de Sousa Leite, R. S. P. Maciel, R. E. P. Vieira, F. Sávio, L. Cerqueira ja M. Mendonça, ”Technical Debt Management in Agile Software Development: A Systematic Mapping Study,” *In Proceedings of the XXIII Brazilian Symposium on Software Quality*, 309-320.