INSTITUTIONEN FÖR TILLÄMPAD IT

TECHNICAL DEBT OCH MINDRE MJUKVARUFÖRETAG

En fallstudie om hur technical debt kan appliceras på mindre mjukvaruföretag

Jonathan Philipp

Kristian Gocko

Kandidatuppsats:	15 hp
Ämne:	Informatik
År:	2018
Rapport nr:	2018:110

Sammanfattning

Technical debt presenterades av Cunningham 1992 och nyligen har ämnet blivit av större intresse inom forskning. Idag är världens IT-skuld över 500 miljarder dollar och det fortsätter att öka. Därför finns det ett intresse att få en förståelse för hur det används i organisationer. Technical debt är ett begrepp som behandlar hur företag arbetar när de utvecklar mjukvara. Framförallt hur företag idag tar genvägar i utvecklingen för att fokusera på andra processer där de kan generera vinster nu. Denna undersökning har syftet att bidra till forskningen kring technical debt genom att fokusera på hur technical debt kan användas för att beskriva mindre mjukvaruföretags beslutsfattande och resultatet därefter. Frågeställningen lyder "*Hur kan technical debt användas för att beskriva mindre mjukvaruföretags beslutsfattande och konsekvenserna därefter?".* För att besvara frågeställningen utfördes det en kvalitativ fallstudie hos ett mindre mjukvaruföretag. Det genomfördes fyra semistrukturerade intervjuer för att fördjupa kunskaperna i hur de arbetade. Därefter analyserades svaren med ett teoretiskt ramverk för att hitta stöd till varför de tog på sig technical debt, vilken form av technical debt och vad utfallet blev. Studiens resultat visar att mindre mjukvaruföretag arbetar mycket med prioritering där fokus ligger på processer som genererar ny funktionalitet till användaren. Där de skjuter upp problem som inte är synliga för användarna. Detta för att få in kapital till företag då de inte har samma resurser som större företag. Studien visar att det är fruktbart att applicera ett teoretiskt ramverk baserat på technical debt hos mindre mjukvaruföretag.

Nyckelord

Technical debt, utveckling, mjukvaruföretag, beslutsfattande, konsekvenser, avvägningar

Technical debt and smaller software development firms

A case study on how technical debt can be applied on smaller software development firms

Abstract

Cunningham presented his idea of technical debt in 1992. Recently the term has seen an increase in attention from both the industry and academia. Today it is estimated that the global IT debt is around 500 billion dollars and continuing to climb. Technical debt in its essence is the act of taking shortcuts in software development that allows you to focus your resources on the core processes. The purpose of this study is to contribute to the research surrounding technical debt by studying how technical debt can be applied to smaller software development firms' decision making and the results that come with it. This leads to the research question "*How can technical debt be used to describe smaller software development firms' decision making and the results that come with it?".* To answer the research question four interviews were held at a smaller software development firm in Sweden. The purpose of the interviews was to get an understanding for how they worked when developing their product. The answers from the interview were analyzed together with a theoretical framework on technical debt to gain knowledge on why companies took on technical debt, what form of technical debt, and what the result was. This study found that smaller companies have a strong use of prioritization where focus lies on processes that generate new functionality to the user. They postpone problems that the user can't see. Doing so to generate income to the business since they do not have the same capital larger firms do. The study shows that it is possible to apply a theoretical framework for technical debt on smaller development firms to describe their way of working.

Keywords

Technical debt, software development, software company, decision-making, consequences, trade-offs

Tack!

Vi vill tacka IT-företaget för deras medverkan i vår undersökning. Vi vill också tacka vår handledare Alan B Carlson för hans rådgivning under arbetet. Till slut vill vi även tacka Marie Eneman för all hjälp vid frågor.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

I detta avsnitt kommer det att presenteras en bakgrund till technical debt. Därefter presenteras studiens syfte och frågeställning. Avsnittet kommer att avslutas med en presentation av studien och uppsatsens upplägg för att få en förståelse över hur de är uppbyggda.

1.1 Bakgrund

Technical debt (TD) påbörjades som en metafor introducerat av Cunningham (1992) för att kunna förklara för otekniska intressenter vad vi förstår idag som *refactoring*. Refactoring, eller som Cunningham förklarade det – *consolidation*, vilket betyder att du som kodare bör ta reda på hur koden skulle ha varit, och du gör det till det (Cunningham 2009). Vad Cunningham menar är att utvecklare alltid ska försöka göra sin kod på bästa möjliga sätt som man förstår problemet vid just det tillfället. Men under projektets gång blir förståelsen mer djupgående och kan därefter gå tillbaka och skriva om koden så att den speglar den förståelsen. Cunningshams inspiration grundas i hur metaforer kan påverka hur vi människor tänker, vilket han läste i boken *Metaphors We Live By* av Lakoff och Johnson (1983). Det initiala fokuset av Cunningham var i en synvinkel av mjukvaruutveckling. Denna metafor har sedan byggts och expanderats av McConnell (2007) som anses vara en större ledare inom området. McConnell går in djupare gällande taxonomin i artikeln och kategoriserar och visar hur man hanterar TD. Efter McConnells inlägg fortsatte trenden gällande TD och Fowler (2009) gav större utbredning om metaforen med hans påbyggande teorier.

Det Cunningham och Fowler beskriver är att refactoring är som en väg att gå tillbaka och ordna gammal kod. Detta är en del av problematiken – att otekniska intressenter ej förstår att programmerare måste välja att skriva kod som reflekterar deras nuvarande förståelse för problemet även om förståelsen är ofullständig (Cunningham 2009; Fowler 2003). Vilket är varför de väljer att lyfta TD – att få fram budskapet av viktigheten att kunna beskriva hur programmerare måste gå till väga för att slutföra någonting i tid; att nå *Time to Market* (TTM). TTM som är ett strategiskt exempel för att skapa värde som kommer exempelvis betala tillbaka skulderna när man behöver gå tillbaka och ordna kod vid ett senare tillfälle.

För ett mindre mjukvaruföretag är det högt prioriterat att nå TTM för att skapa värde. Mindre mjukvaruföretag befinner sig i en arbetsmiljö med hög risk och snabbgående förändring (Giardino, Unterkalmsteiner, Paternoster, Gorschek & Abrahamsson 2014). Beslut, deadlines och att få saker gjorda sker på ett dagligt basis, till skillnad från större företag med ett mer disciplinerat arbetssätt. Det uppstår nya mindre mjukvaruföretag varje år där vissa lyckas och andra inte och därför är det av intresse att få en förståelse till hur de arbetar och hur deras arbetsprocesser påverkar organisationen och produkten.

En annan del av problemet är att det tekniska teamet tillsammans med affärsteamet har inte haft en enkel väg att prata om vad TD innebär. McConnell beskrev detta i en intervju (OnTechnicalDebt 2012) där han berättade att konceptet av TD ger en medvetenhet till alla dessa kopplade problem som ofta är gömda och svåra att prata om. Konceptet ger ett stöd till att göra processen till tekniska beslut enklare – och att bygga en brygga mellan det tekniska- och affärsteamet. Detta är för att kunna ha en öppen konversation som leder till det bästa möjliga affärsbeslut.

Enligt McConnell har TD blivit mer verksamt det senaste decenniet inom forskningen. McConnell påpekar att konceptet är en tredjedels hjälpfull som ett beslutsverktyg och två tredjedelar som ett kommunikationsverktyg mellan utvecklarna och affärsteamet (OnTechnicalDebt 2012).

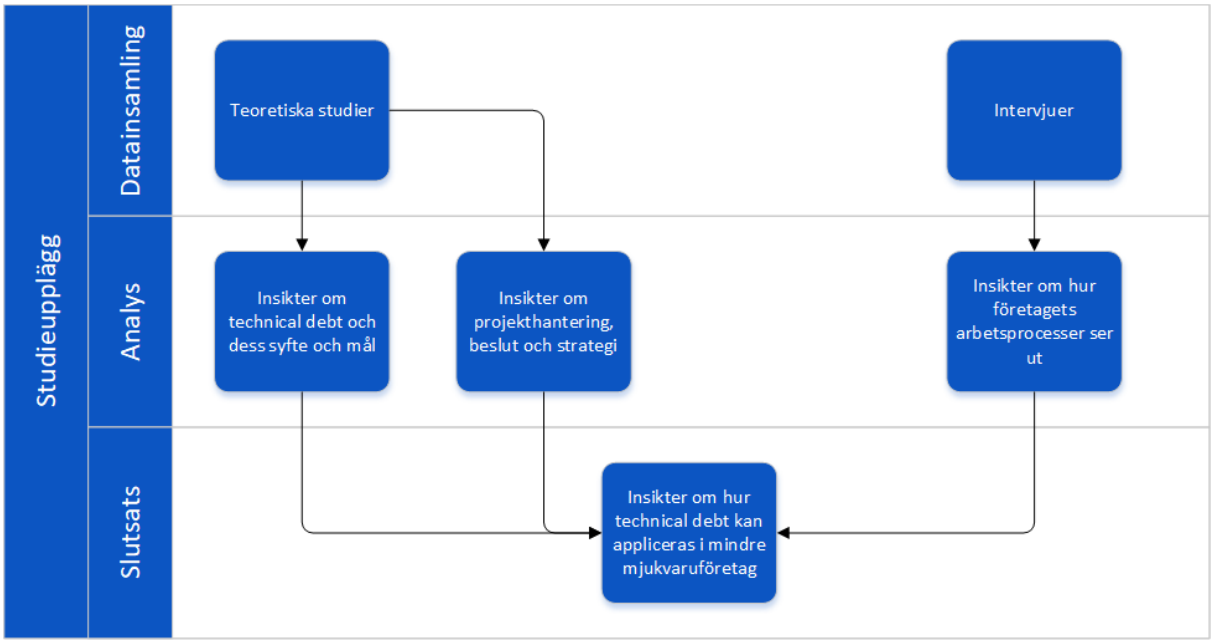
1.2 Syfte och frågeställning

Cunningham presenterade TD för första gången 1992 (Cunningham 1992) och sen dess har forskningen kring TD ökat markant. Gartner estimerade att världens IT-skuld vid 2010 var ungefär 500 miljarder dollar och att det fanns en risk att det skulle öka till en biljon dollar vid 2015 (Gartner 2010). Denna uppsats har syftet att bidra till forskningen till hur TD kan appliceras på arbetsprocesserna och besluten tagna i mjukvaruföretag. För att skilja denna undersökning från andra har vi valt att fokusera på mindre mjukvaruföretag. Detta leder till vår frågeställning:

Hur kan technical debt användas för att beskriva mindre mjukvaruföretags beslutsfattande och konsekvenserna därefter?

1.3 Studiens upplägg

För att beskriva studiens upplägg har vi tagit fram en modell som beskriver hur vi har gått tillväga (se figur 1). Modellen för studieupplägget har tre nivåer: *datainsamling*, *analys* och *slutsats*. Under datainsamlingen har vi teoretiska studier, som vi kopplar till vår analys som innehåller insikter om technical debt, samt projekthantering, beslut och strategi. Med detta gjort, ihop med intervjuer som gett oss insikt om hur företagets arbetsprocesser ser ut, kan vi se hur technical debt kan appliceras i ett mindre företag.



Figur 1: Studieupplägg

1.4 Uppsatsens upplägg

I nästkommande avsnitt presenteras definitioner av TD följt av teori om mindre mjukvaruföretag och därefter presenteras det teoretiska ramverket vi har valt använda i vår undersökning. I avsnitt tre behandlas hur studien har gått tillväga samt motiveringar till varför vi valt att använda dem. Avsnitt fyra innehåller resultatet från vår studie hos ett mindre mjukvaruföretag. Avsnitt fem är en diskussion där resultatet jämförs mot det teoretiska ramverket. Uppsatsen avslutas med en slutsats i avsnitt sex.

2 Teori och tidigare forskning

Detta avsnitt har målet att presentera teorin som ligger till grund för denna undersökningen. Först kommer det att presenteras ett kort avsnitt där definitionen för TD och generella drag för mindre mjukvaruföretag presenteras. Därefter kommer det att presenteras teorier om TD i form av ett ramverk. Teoriavsnittet avslutas med ett mindre avsnitt om organisationers åsikter på TD.

I detta avsnitt kommer det att presenteras teorier från Cunningham (1992), McConnell (2007) och Fowler (2003). Anledningen till varför dessa källor lyfts här är för att motivera varför vi har valt källor som inte är vetenskapliga texter. Under litteraturundersökningen såg vi att många av texterna refererade till dessa tre individer. Dessa tre var en gemensam faktor i nästan alla vetenskapliga artiklar och därför anses de vara legitima att använda som källor i studien.

2.1 Definitioner

I detta avsnitt kommer det att presenteras vad TD är och hur mindre mjukvaruföretag definieras.

2.1.1 Technical debt

Technical debt (TD) är ett koncept främst inom mjukvaruutveckling som iakttar kostnader som uppstår när man väljer att ta en snabbare och enklare väg inom utveckling, som vid senare tillfälle måste åtgärdas (refactoring) (Cunningham 1992; Fowler 2003). Detta kan vara både positivt och negativt; en positiv aspekt exempelvis är att man väljer den enklare vägen att utveckla för att få en funktionalitet ut i tid för att inte missa möjlighetsfönstret på marknaden (Time to Market). Med detta gjort får man tillbaka värde från att nå marknaden i tid som kommer att betala tillbaka skulderna för att ordna koden i efterhand.

2.1.2 Startups och mindre mjukvaruföretag

Från början är ett nytt företag en startup, ett nyskapat företag som utforskar nya affärsmöjligheter, eller arbetar att lösa problem som inte är välkända, där marknaden är under konstant förändring (Giardino et al. 2014). Ett nyskapat företag behöver dock inte betyda att det är en startup – det finns två egenskaper som kännetecknar en startup, vilket är hög osäkerhet och snabb utveckling (Giardino et al. 2014). Dessa två egenskaper är vad differentierar startups från väletablerade företag.

Det som definierar en startup är hur miljön är där man arbetar, vilket ofta kan vara oförutsägbara, dynamiska och ibland kaotiska – vilket tvingar entreprenörerna att fatta beslut snabbt

och lära sig från sina misstag (Giardino et al. 2014). Detta kan vara exempelvis att hitta en nischad marknad för att kunna upprätthålla kapital för företagets överlevnad. Data visar att 60% av startups överlever inte de första fem åren, medan 75% av riskkapitalsstartade startups misslyckas (Giardino et al. 2014). Detta är på grund av den höga risken av startups, där missade marknadsöppningar är en stor orsak (Giardino et al. 2014). Detta är något Fowler och McConnell lyfter; att ta strategiska beslut genom att ta genvägar för att lansera något inom en specifik tidsram för att undvika missade marknadsöppningar är essentiellt för värdeskapande (Fowler 2003; Giardino et al. 2014; McConnell 2007).

Det finns vissa återkommande teman som karaktäriserar mjukvarustartups, vilket är brist på resurser. Mjukvarustartups är typiskt väldigt flexibla och snabba att agera till omställningar i marknaden eller nya teknologier till skillnad från mer etablerade företag. Innovation är även något startups fokuserar på; att utforska innovativa segment av marknaden (Giardino et al. 2014).

På grund av brist på resurser är tredjepartsbibliotek något startups tenderar att använda och förlita sig på för att utveckla sin produkt. Detta kan vara externa API:er, outsourcing av viss utveckling och öppen källkod (Giardino et al. 2014).

2.2 Teoretiskt ramverk för technical debt

Nedan presenteras ramverket som kommer att ligga till grund för analysavsnittet. Först kommer det att presenteras klassificeringar och typer av TD. Därefter kommer aspekter av TD och varför organisationer tar på sig TD. För att sedan avsluta med utfallet av TD. Ramverket visualiseras (se figur 2, tabell 1) i avsnitt 2.2.6 för att få en bättre inblick i hur de olika delarna relaterar till varandra. I avsnitt 2.2.7 presenteras det ett exempel på hur ramverket kan appliceras på ett scenario.

2.2.1 Klassificeringar av technical debt

Tom, Aurum och Vidgen (2013) bygger på McConnells (2007) beskrivning av TD när de presenterar deras klassificeringar av TD. De presenterar att det finns fyra olika klassificeringar av TD strategisk, taktisk, inkrementell och oavsiktlig TD där de tre första kategoriseras i McConnells (2007) typ två klassificering. Det görs ett aktivt val att ta på en skuld. *Strategisk TD* är en form av långsiktig TD (Tom, Aurum & Vidgen 2013) där en organisation gör ett aktivt val att prioritera processer som gynnar organisationen nu i stället för att få nytta i framtiden (McConnell 2007). Denna form av TD karaktäriseras med en mer distinkt och synlig skuld där återbetalningsplanen är långsiktig (Tom, Aurum & Vidgen 2013). Strategisk TD kan exempelvis användas genom att en organisation väljer att öka hastigheten på deras utvecklare på bekostnad av produktens kvalitet. I det här scenariot är organisationen medveten om risken med långsiktiga kvalitetsproblemen kopplat till detta val men väljer att ta på sig en skuld för att exempelvis potentiellt vara först på marknaden (Tom, Aurum & Vidgen 2013).

Taktisk TD lik strategisk är en form av TD där en organisation aktivt väljer att sätta sig i ett skuldsatt läge. Skillnaden är att taktisk TD är kortsiktigt i stället för långsiktigt samt att i stället för att vara planerat som strategisk är taktisk en form av reaktions TD (Tom, Aurum & Vidgen 2013). Ett exempel på kortsiktig taktisk TD är att om en organisation är i slutstadiet av ett projekt och inte har tid att implementera en ny funktion på det rätta sättet kan de i stället välja att ta genvägar nu för att få ut det på marknaden och sen lösa det efter att det har släppts (McConnell 2007).

Enligt McConnell (2007) går det att jämföra *inkrementell TD* med kreditkort eftersom det är väldigt enkelt att samla på sig men är svår att spåra och hantera. McConnell (2007) beskriver inkrementell TD med många små genvägar. Det kan vara generiska variabelnamn i koden, bristande kommenterar i kod eller att en utvecklare väljer att inte följa konventioner inom programmering. Inkrementell TD är en av de största orsakerna till varför underhållskostnader för äldre system ökar (Tom, Aurum & Vidgen 2013). Det är upp till cheferna att ta beslut för att gå tillbaka och åtgärda genvägarna för att förhindra att problemen bygger på varandra och riskera hamna i en snöbollseffekt (Tom, Aurum & Vidgen 2013).

Oavsiktlig TD är enligt McConnell (2007) och Tom, Aurum och Vidgen (2013) en form av TD där det inte görs ett medvetet val att ta på sig TD utan som namnet tyder samlas på oavsiktligt. Även om det inte görs ett medvetet val att ta på sig en skuld uppstår det fortfarande en kostnad som måste betalas tillbaka (Tom, Aurum & Vidgen 2013). McConnell (2007) presenterar exempelvis att en junior programmerare kan orsaka oavsiktlig TD – eftersom de skriver kod av bristande karaktär. Ett annat exempel är att en programmerare tar ett beslut utan att veta konsekvenserna – eftersom det gjordes utan att vet vad andra gör (Tom, Aurum & Vidgen 2013). Tom, Aurum och Vidgen (2013) säger att det är när besluten skapar ökade kostnader för framtiden att en oavsiktlig TD har skapats.

2.2.2 Typer av technical debt

Innan det presenteras typer av TD är det viktigt att få en förståelse för skillnaden mellan klassificeringar och typer. Klassificeringar av TD behandlar den strategiska anledningen till varför TD togs på av företaget. Till skillnad beskriver typer av TD vilken form TD har manifesterats i organisationen. Exempelvis om TD har uppstått i koden, hos testning eller hos dokumentationen (Tom, Aurum & Vidgen 2013).

Det har presenterats många olika typer av TD och för att säkerställa att de flesta representeras i denna studie kommer det att presenteras en sammanställning av typer från Li, Avgeriou och Liang (2015) med stöd från Tom, Aurum och Vidgen (2013). Den första typen Li, Avgeriou och Liang (2015) presenterar är *krav TD*. Krav TD enligt Ernst (2012) är en form av TD som uppstår när den optimala kravspecifikationen inte är den som blev implementerad. Mängden TD beror på hur stor skillnaden är mellan den optimala och den faktiska kravspecifikationen (Ernst 2012). Krav TD förekommer när en organisation väljer att prioritera krav som inte gynnar användaren eller egentligen inte behövs. När en organisation använder en bristande kravspecifikation sätter den sig i en problematisk situation då krav kan ändras med tiden och med

det krävs uppdateringar till systemet. Men detta extra arbete hade potentiellt inte uppstått om en korrekt analys på krav hade gjorts från början (Ernst 2012).

När en organisation tar ett beslut angående en produkts arkitektur där utfallet får en negativ påverkan på produktens kvalitet blir det en form av *arkitektur TD* (Li, Avgeriou & Liang 2015). Tom, Aurum och Vidgen (2013) definierar arkitektur TD med resultatet av suboptimala lösningar eller att en lösning klassificeras utgående på grund av att teknologin har utvecklats tillräckligt mycket att den nuvarande lösningen inte är aktuell längre. Enligt Zazworka, Shaw, Shull och Seaman (2011) går arkitektur TD och *design TD* in i varandra. Design TD behandlar genvägar tagna vid designen av produkten vilket har lett till brister i den slutgiltiga produkten. Det kan exempelvis vara ett bristande fokus på produktens underhållning (Li, Avgeriou & Liang 2015; Tom, Aurum & Vidgen 2013). Design TD förekommer när den befintliga designen inte längre är bäst lämpad för lösningen. Det kan exempelvis vara implementeringen av nya funktioner på en bristande arkitektur (Zazworka, Shaw, Shull & Seaman 2011).

Kod TD behandlar kod av en sämre karaktär. När utvecklare går emot etablerade kodstandarder eller varje gång en utvecklare tar en omväg byggs det upp TD i projektet i form av kod TD (Li, Avgeriou & Liang 2015; Tom, Aurum & Vidgen 2013). Några utav de vanligaste orsakerna till kod TD är komplex kod – vilket gör det svårt att läsa och förstå koden samt att den bakomliggande designen till koden inte är optimalt utformad. Detta bidrar till en ökad risk att produkten får problem i framtiden (Tom, Aurum & Vidgen 2013). Det finns en enorm risk att om det väljs att låta bli att åtgärda kod TD att system får långsiktiga problem där det blir svårt att underhålla och bygga på systemet (Bohnet & Döllner 2011).

För att förhindra att användare får en produkt med buggar utförs testning på produkten innan den släpps på marknaden. När en organisation beslutar att ta genvägar under detta stadie eller om det finns brister i deras metodik uppstår det *test TD* (Li, Avgeriou & Liang 2015; Tom, Aurum & Vidgen 2013). Det kan exempelvis vara brist på antal tester, att de saknar specifika tester (Li, Avgeriou & Liang 2015) eller att det saknas automatiserade tester vilket gör att allt måste testas manuellt (Tom, Aurum & Vidgen 2013). Effekterna av att ett system inte testats tillräckligt är att dels får användarna en produkt med buggar vilket får en negativ påverkan på varumärket. Samtidigt att det krävs tid och pengar för att felsöka och åtgärda problem som kunde upptäckas innan produkten släpps hade testningen varit godtycklig (Tom, Aurum & Vidgen 2013).

Dokumentations TD eller enligt Tom, Aurum och Vidgen (2013) brist på kunskapsfördelning behandlar problematiken med bristande dokumentation under utvecklingen av mjukvara (Li, Avgeriou & Liang 2015). Det kan exempelvis vara att dokumentationen inte är uppdaterad, att det inte har färdigställts eller att det inte finns någon dokumentering exempelvis kommentarer i koden. Välskriven dokumentering gör det enklare att sprida kunskap inom organisationen och genom det minskar chansen att en organisations TD ökar (Tom, Aurum & Vidgen 2013).

Infrastruktur TD bildas när en organisation har suboptimala processer vid utvecklingen av deras produkt (Li, Avgeriou & Liang 2015). Det kan exempelvis vara användandet av teknologier inte anpassade åt jobbet, suboptimala supportverktyg (Li, Avgeriou & Liang 2015) eller manuella processer där de kunde använda automatiserade i stället (Tom, Aurum & Vidgen 2013). Effekterna av detta är en negativ påverkan hos ett teams förmåga att producera en produkt med bra kvalitet (Li, Avgeriou & Liang 2015) vilket har potentialen att påverka varumärkets rykte negativt (Tom, Aurum & Vidgen 2013).

De sista typerna av TD från Li, Avgeriou och Liangs (2015) är *bygg TD*, *defekt TD* och *versionshanterings TD*. Dessa behandlas i samma stycke då de bedöms vara mindre betydande än de ovanstående i denna undersökning. Bygg TD behandlar brister i processen när en mjukvara ska byggas efter att utvecklingen är klar. Det kan exempelvis vara manuell byggprocess (Li, Avgeriou & Liang 2015). Defekt TD handlar om mängden buggar och defekter i en mjukvara. Till sist finns det versionshanterings TD vilket är när det uppstår problem med versionshanteringen i ett projekt. Det kan exempelvis vara att det finns för många olika förgreningar eller versioner av ett projekt vilket gör det komplicerat att hantera projektet (Li, Avgeriou & Liang 2015).

2.2.3 Aspekter av technical debt

För att komplettera ovanstående klassificeringar och typer av TD har det tagits fram olika aspekter för att beskriva TD. Det har visats att TD har direkta kopplingar till ökade monetära kostnader inom organisationen. Tidigare har det nämnts att TD kan få en negativ effekt på produktens kvalitet och tiden det tar för utvecklarna att åtgärda de problemen kostar organisationen pengar. Det är tid och pengar organisationen kunde använt för ny funktionalitet men i stället behöver användas till att åtgärda tidigare problem (Tom, Aurum & Vidgen 2013). När man går tillbaka och åtgärdar ett tidigare problem – exempelvis implementera en funktion på det korrekta sättet i stället för den suboptimala lösningen kallas det för *paying back the principal* (Brown, Cai, Guo, Kazman, Kim, Kruchten, Lim, MacCormack, Nord, Ozkaya, Sangwan, Seaman, Sullivan & Zazworka 2010). Men det är inte den enda kostnaden utan det bildas även *ränta* på TD och det är kopplad till påverkan av att ha implementerat en suboptimal lösning (Brown et al 2010). Detta kan exempelvis vara personalens moral, produktiviteten i organisation eller produktens kvalitet (Tom, Aurum & Vidgen 2013).

Likt hur ett finansiellt lån kan användas för att få ut en form av avkastning kan TD användas i organisationer för att få kortsiktiga vinster (Tom, Aurum & Vidgen 2013). Detta benämns till *leverage* och med det kan en organisation välja att ta genvägar i sitt arbete för att till exempel få ökad produktivitet. McConnells (2007) liknelser med kreditkort beskriver hur enkelt det är att samla på sig TD men trycker på hur oerhört viktigt det är att det betalas tillbaka i tid. Eftersom TD kan vara svårt att spåra blir det ännu svårare för organisationer att betala tillbaka sin skuld (Tom, Aurum & Vidgen 2013). En utav orsakerna till varför utvecklare inte går tillbaka för att lösa deras TD är för det är inte kul. De lägger hellre mer tid på att utveckla nya funktioner i stället för att åtgärda gamla.

Om en organisation kommer till ett läge där mängden TD är för mycket att hantera och det krävs att man skriver om projektet från början benämns det att projektet har gått i *konkurs*. Mängden skuld har blivit för mycket att hantera (Tom, Aurum & Vidgen 2013). Det finns en situation där en organisation inte behöver betala tillbaka sin skuld utan kan skriva av det. Denna situation har namnet *debt amnesty*. Det har beslutats att en produkt eller funktion har nått slutet på sin livscykel och då finns det en möjlighet att skriva av den produkt eller funktionens TD (Tom, Aurum & Vidgen 2013).

2.2.4 Varför organisationer tar på sig technical debt

Det finns flera olika anledningar till varför organisationer tar på sig TD. Två utav anledningar med starka kopplingar till varandra är *prioritering* och *pragmatism* (Tom, Aurum & Vidgen 2013). Prioritering handlar i sin grund att ett team väljer att prioritera vissa processer högre än andra. Det kan exempelvis vara att fokusera på att implementera kritiska funktioner över kvalitet för att säkerställa att man blir först på marknaden (McConnell 2007). Pragmatism lik prioritering handlar om att bedöma de viktigaste uppgifterna men till skillnad från den mer strukturella processen med prioritering där det finns regler för vad ska prioriteras över andra bygger pragmatism på att hantera problem och beslut utifrån parametrar man har fått i stället för att följa etablerade regler och teorier (Cambridge Dictionary 2018; Tom, Aurum & Vidgen 2013). Exempel på en sådan situation är när det tas beslut om att släppa en produkt där man är medveten att det kommer behövas underhåll men valet görs för att få in inkomst till företaget (Tom, Aurum & Vidgen 2013).

Etablerade *processer* i en organisation och hur väl de följs av medarbetarna är ännu en orsak till varför TD skapas i en organisation (Tom, Aurum & Vidgen 2013). Två nyckelprocesser för att hålla nere TD är kommunikation och kollaboration. Finns det inga rutiner för de här processerna får utvecklare en sämre förståelse för vad andra gör. Vilket gör det enklare för dem att ta beslut om genvägar i sin kod vilket ökar TD i projektet. *Attityden* och *kunskapen hos medarbetarna* är de sista orsakerna Tom, Aurum och Vidgen (2013) presenterar till varför organisationer tar på sig TD. Olika personer och olika team har olika åsikter och attityder mot TD där vissa är mer för det och andra emot det. Detta gör att mängden TD i en organisation beror till viss del på vem det är som arbetar på projektet. Det beskrivs en situation där medarbetarna besitter på kunskapen om vad TD är men inte kunskapen om hur de undviker det. Detta benämns till okunnighet. En av de mer riskfyllda typerna av utvecklare är de med kunskapen – men väljer att inte använda det eftersom de inte tänker på vad det har för påverkan i framtiden (Tom, Aurum & Vidgen 2013).

2.2.5 Resultatet av technical debt

Det finns fyra huvudområden i en organisation där TD får en påverkan. Tom, Aurum och Vidgen (2013) identifierar påverkan på moral, produktivitet, kvalitet och risk. Kortsiktigt anser de att det finns goda chanser för en positiv påverkan på alla områden utom kvalitet – men om TD inte tas hand om och återbetalas får det alltid en negativ påverkan långsiktigt. Tittar man djupare på hur *moralen* påverkas kan det både påverkas positivt och negativt kortsiktigt.

Generellt går det att säga att moralen ökas för utvecklare eftersom de får arbeta med nya funktioner men det finns en risk att påverkan blir negativ för de som är stolta över sitt arbete och inte vill ta genvägar i sitt arbete. Låter man TD ligga för länge kommer det att enbart vara negativa konsekvenser långsiktigt på grund av att vid någon tidpunkt måste det betalas tillbaka.

Kortsiktigt möjliggör TD ökad produktivitet i organisationer men på längden kommer den effekten att försvagas eller resultera att produktiviteten blir noll (Tom, Aurum & Vidgen 2013). Med hjälp av TD ges det möjlighet för utvecklare att fokusera sin tid på uppgifter där det genereras värde nu i stället för de med en mer långsiktig syn. Men dessa genvägar får en negativ påverkan på längden av olika anledningar. Eftersom det tas genvägar i utvecklingen blir det svårare att ändra och bygga på koden, mer tid behöver läggas ner för att felsöka och åtgärda tidigare problem i stället för att utveckla nya funktioner. När det tas genvägar i exempelvis dokumentering där resultatet blir bristande dokumentation för projektet måste utvecklarna spendera mer tid på att förstå både projektet i sin helhet och koden de ska arbeta på (Tom, Aurum & Vidgen 2013).

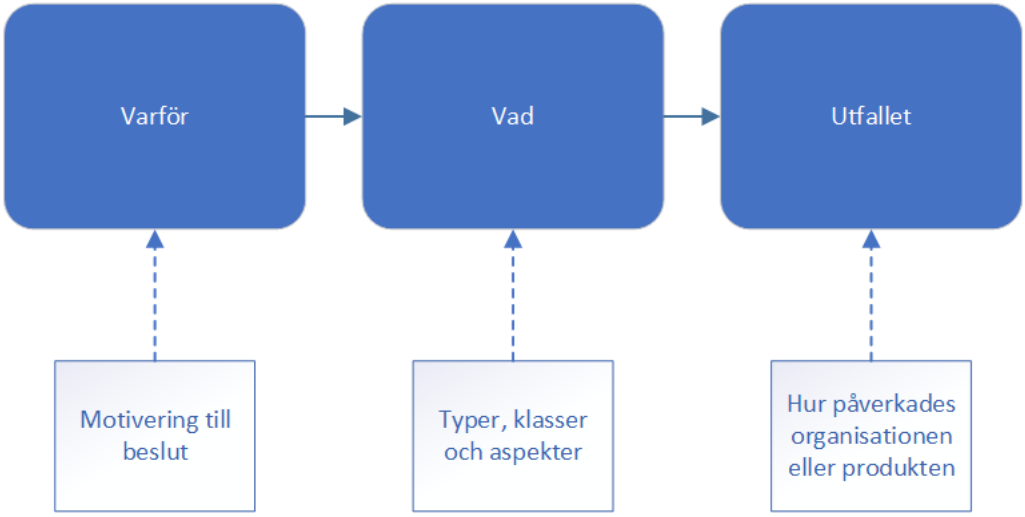
Risk behandlar projektets risker och med TD finns det potential att minska riskerna kortsiktigt men finns alltid en risk att det ökar i stället. Genom att ta genvägar och bygga upp TD blir det enklare att nå deadlines men om inte projektets TD är synlig blir det svårare att nå deadlines (Tom, Aurum & Vidgen 2013). Problematiken ökar långsiktigt om TD inte återbetalas där det byggs TD på varandra och till slut krävs det en hel ombyggnation av produkten.

Den sista påverkan är *kvalitet* och denna del får alltid en negativ påverkan oavsett om det är kortsiktigt eller långsiktigt (Tom, Aurum & Vidgen 2013). Hela konceptet med TD bygger på att ta genvägar i utvecklingsprocessen och med genvägarna kommer det att förekomma defekter och buggar. Men av strategiska anledningar väljs det exempelvis att prioritera snabbhet över kvalitet. Det viktiga med kvalitet är att all TD bidrar till att skapa en negativ påverkan, det finns ingen typ eller klassificering där resultatet blir positivt. En utav anledningarna till varför TD får en negativ påverkan på kvalitet är att TD ligger till grund för varför defekter skapas men är också anledningen till varför de inte hittas. Några utav de vanligaste områden TD påverkar inom kvalitet är skalbarhet, underhållning, prestanda, användbarhet, testbarhet och säkerhet (Tom, Aurum & Vidgen 2013).

2.2.6 Visualisering av ramverket

Figur 2 visar hur det teoretiska ramverket från avsnitt 2.2 kan visualiseras. För att komplettera bilden har det även tagits fram en tabell (se tabell 1) för att på ett enkelt sätt visa upp de olika delarna i ramverket. Figuren bygger på ramverket beskrivet av Tom, Aurum och Vidgen (2013) där de presenterar en liknande bild. Men för att göra den mer lättförståelig har vi skalat ner den till de viktigaste elementen. Ramverket börjar med att få en förståelse till varför det togs på en TD (avsnitt 2.2.4) och vilka motiveringar det fanns bakom beslutet. Därefter beskrivs det vilken klassificering (avsnitt 2.2.1) och typer (avsnitt 2.2.2) av TD det är samt vilka

aspekter (avsnitt 2.2.3) den har. Till sist vill vi veta hur detta beslut med TD påverkade (avsnitt 2.2.5) både organisationen och produkten. Både kortsiktigt och långsiktigt.



Figur 2: Visualisering av ramverket

Tabell 1: Överblick på ramverket

Varför	**Klassificeringar**	**Typer**	**Aspekter**	**Utfall**
Prioritering Pragmatism Processer Attityd Kunskap	Strategisk Taktisk Inkrementell Oavsiktlig	Krav Arkitektur Design Kod Test Dokumentation Infrastruktur Bygg Defekt Versionshantering	Monetär kostnad Ränta och principal Leverage Amnesty Konkurs Utdrag och återbetalning	Moral Produktivitet Kvalite Risk

2.2.7 Exemplifiering av ramverket

För att exemplifiera hur ramverket kan användas ges följande enklare scenario. Ett företag är i slutstadiet av ett projekt och behöver nu bara testa produkten innan den släpps på marknaden. Innan testningen påbörjas får de information om att deras konkurrent planerar att släppa en liknande produkt om en vecka. Chefen bestämmer då att bara göra en grundlig testning i stället för den normala djupgående testningen för att hinna före konkurrenten. Produkten släpps på marknaden och defekterna som inte fångades av testningen åtgärdas nu efteråt.

Det vi kan se här är att chefen tog beslutet utifrån magkänsla i stället för regler vilket motsvarar *pragmatism*. Det kan finnas flera anledningar varför men i detta exempel använder vi bara en. Det var en klassificering av *taktisk TD* eftersom det var kortsiktigt och reagerade mot vad konkurrenterna planerade. TD manifesterade i form av *test TD* och *defekt TD* där det fanns brister hos testningen och defekter uppstod hos produkten. Den hade aspekter i form av *monetär kostnad* för att laga defekterna därefter. *Ränta och principal* i form av återbetalning av skulden och andra påverkningar den har haft. *Leverage* går att identifiera eftersom chefen gjorde ett beslut för att få kortsiktiga vinster. Beslutet fick en ökning i *produktivitet* kortsiktigt – där de kunde få ut produkten snabbare på bekostnad av att produktiviteten kommer sänkas senare för att åtgärda problemen. Samtidigt som produktens *kvalitet* påverkades negativt då defekter gick igenom testningen.

Detta illustrerar hur ramverket kan användas för att beskriva varför TD togs på, vilken form den tog och vad utfallet blev därefter. En viktig detalj är att varje beslut kan ha flera anledningar till varför TD togs på, det kan ha flera typer och aspekter och få en påverkan på flera olika sätt. Dock är det bara en form av klassificering per beslut.

2.3 Organisationers syn på technical debt

McConnell (2007) drar paralleller med ett banklån till hur organisationer ser på TD. Där vissa är emot det och inte vill att det ska finnas någon TD alls i organisationen medan andra vill veta hur man använder det korrekt. Enligt Tom, Aurum och Vidgen (2013) borde inte organisationer arbeta mot att betala tillbaka all sin TD eller aktivt undvika det utan lik McConnell (2007) borde de fokusera på att hantera det på bättre sätt. McConnell (2007) beskriver vidare att alla är överens om att tanken är bra men att utförandet är problemet. Tom, Aurum och Vidgen (2013) fortsätter med att TD är oundviklig, prioriteringar måste göras i en organisation och med det kommer TD. De fortsätter med att förklara att du kan göra allt du kan för att få bort det men det är nästintill omöjligt idag. En utav anledningarna till varför det har fått en negativ ton är på grund av att eftersom det är mycket mer osynligt till skillnad från vanliga monetära skulder blir det svårt att hantera det. När det kontrolleras korrekt är det ett oerhört viktigt verktyg med potentialen att hjälpa organisationer leverera en bättre produkt (McConnell 2007; Tom, Aurum & Vidgen 2013).

3 Metod

För att besvara vår frågeställning har vi valt att genomföra en kvalitativ fallstudie på ett mindre bolag, som vi benämner i studien med IT-företaget. IT-företaget levererar en produkt inriktad mot byggbranschen. Datainsamlingsmetoden är semistrukturerade intervjuer, som genomförts på flera anställda inom företaget vilket ger ett perspektiv från flera synvinklar.

I detta avsnitt kommer vi först introducera metodologin och motivering till varför den valdes. Därefter beskrivs det kort om fallstudieobjektet IT-företaget. Sedan framläggs studiens litteraturundersökning. Därefter beskrivs den valda datainsamlingsmetoden och hur den genomfördes. Det avslutas med att beskriva hur analysen gick till väga.

3.1 Kvalitativ fallstudie

Frågeställningen har anlag för att vara en undersökande typ, vilket innebär att djupgående och flexibla metoder behövdes för att nå ett svar. Detta beskriver Patel och Davidson (2011) – att kvalitativa metoder lämpar sig väl för denna typ av undersökning.

3.1.1 Fallstudieobjekt

För att besvara frågeställningen har vi valt att studera ett mindre mjukvaruföretag, IT-företaget. Mer specifikt de anställda inom företaget och hur de arbetar. IT-företaget har fyra anställda på kontoret och en person som arbetar på en annan plats. Gruppen vi valt att undersöka har arbetat tillsammans i cirka två år, med en nyanställd som varit med IT-företaget lite längre än ett kvartal. Därav får vi ett helhetsperspektiv av hur IT-företaget arbetar och täcker information från flera synvinklar. Vi valde att utföra studien på IT-företaget eftersom de matchade kriterierna till mindre mjukvaruföretag och var villiga att ställa upp för intervjuer.

3.2 Litteraturundersökning

Litteraturundersökningen skedde ständigt under projektets gång. Vi hade en forskningsperiod där vi enbart bearbetade litteratur. Då fokuserades det främst på översiktslitteratur och de mest citerade inom ämnet. Under denna tid bedömdes vilken relevant litteratur vi kommer att använda oss av, där vi fördjupade oss i de väsentliga områdena för syftet av undersökningen. Under denna period hade vi som avsikt att förhålla oss enbart till vetenskaplig litteratur – men upptäckte att TD-området har byggts på av industriexperter i stället för akademin. Det vi har sett under litteraturundersökning är att de mest citerade och relevanta artiklarna refererat till Cunningham, Fowler och McConnell. Vår bedömning kring val av litteratur bedömdes av antal citeringar, publiceringsdatum och vilken tidskrift den är publicerad i. Sökningsmetodiken

var till stor del via Google Scholar och dess filtreringsfunktionalitet med termer för att begränsa oss inom specifika områden i avsikt till vår undersökning, som Patel och Davidson (2011) nämner är väsentligt under sökningsperioden. Vi tog även vara på behändig hjälp från dels forskare och universitetsadjunkter inom IT-institutionen för rekommendationer av litteratur för vår specifika undersökning.

3.3 Datainsamlingsmetod

Den valda datainsamlingsmetoden för vår undersökning bestod av semistrukturerade intervjuer på samtliga anställda på det lokala kontoret hos IT-företaget. Detta ger ett helhetsperspektiv då vi får se organisationens arbetsprocesser från alla synvinklar (från VD till huvudutvecklare till juniorutvecklare till kundtjänst). Det ansågs inte vara nödvändigt att informanterna behövde komma förberedda då frågorna var främst fokuserade på deras arbetsprocesser.

Efter intervjuerna var klara användes det inspelade materialet för att transkriberas enligt Bailey (2008). Det Bailey menar är att transkribering verkar som en typisk uppgift, men i själva verket involverar flera olika detaljer, vilket man får bedöma vilka är lämpliga för sin situation (t.ex. att utelämna interaktioner som man vanligtvis inte ser i textform). För oss var noggrannheten ett huvudsakligt fokus där vi inte utelämnade någon information och beskrev ordagrant, samt beskrivningar av vissa betoningar på ord för att undvika misstolkningar (Bailey 2008).

3.3.1 Intervjuer

För att svara på vår frågeställning intervjuade vi anställda på IT-företaget. Intervjuerna var semistrukturerade med satta teman kopplade till det teoretiska ramverket (se figur 2). Dessa teman togs fram via diskussioner och med hjälp av att avbilda det teoretiska ramverket visuellt som i sin tur ledde till valda frågor inom de olika temana. Se bilaga 1 för intervjumall. Dessa frågor var standardiserade, men vi hade ett öppet förhållningssätt till att ställa frågorna, som Patel och Davidson (2011) påpekar att vissa intervjuare väljer att ställa frågorna som faller bäst i det enskilda fallet. I vårt fall följde vi intervjumallen och valde att ställa följdfrågor för att fördjupa kunskaperna kring ett visst tema som vi ansåg givande för undersökningen.

3.3.2 Urval

Det beslutades att vi skulle intervjua fyra informanter eftersom studiens tidsram och resurser var begränsade. Samtliga fyra anställda på kontoret var tillgängliga att intervjuas – vilket gav oss ett helhetsperspektiv från olika synvinklar, som är en grundläggande stapel för vår typ av undersökning. Detta bekräftas av Bell (2014) att det är en bra metodik för att studera ett specifikt problem mer ingående med de begränsningar vi har haft.

Våra informanter var:

> **Informant 1:** Grundaren/utvecklare och chef över teknik, ansvarar för beslutsfattande gällande utveckling.

Informant 2: Juniorutvecklare, arbetar främst med front-end-utveckling. Relativt ny på företaget.

Informant 3: Support, tar emot kundsamtal och dess feedback och rapporterar vidare. Håller även på med testning av produkten.

Informant 4: VD, håller främst på med det ekonomiska, marknadsföring och ledning – samt även sälj, support och testning.

3.3.3 Bortfall

IT-företaget är av det mindre slaget, där alla anställda på det lokala kontoret intervjuades, då de har en person som är anställd och arbetar utanför kontoret. Vi upplevde att detta ej var ett problem eftersom en av de anställda på IT-företaget hade liknande arbetsuppgifter, där dess primära uppgifter är kundsupport och sälj.

3.3.4 Genomförande

Intervjuerna bokades genom e-post där vi hade kontakt med företagets grundare. Vi valde att strukturera upp intervjuerna över två dagar. Två intervjuer första dagen och två den andra. Detta gjordes av anledningen att vi ville få tid att gå tillbaka och analysera hur det har gått under första dagen för att hitta aspekter vi kan förbättra på. Vi ändrade inga frågor utan en av aspekterna vi kritiserade var dynamiken mellan forskarna för att göra intervjuerna smidigare vid ställningen av följdfrågor.

Innan intervjuerna hölls förberedde forskarna sig en kortare tidsperiod på 15 minuter i konferensrummet. Intervjuerna hade varierande längder. Första intervjun var ca. 50 minuter, andra ca. 35 minuter, tredje ca. 60 minuter och den sista ca. 60 minuter. Valet av att intervjua dem på informanternas arbetsplats var för att få informanterna att inte känna sig utsatta vilket hade potentiellt påverkat deras svar (Patel & Davidson 2011). En eventuell förklaring till varierande längd på intervjuerna kan bero på personligheten av informanten och val av följdfrågor. Följdfrågsvalen kan ha påverkats då vi är oerfarna och detta är ett nytt område för oss.

När informanten kom till konferensrummet presenterades vilka vi var och vart vi kommer ifrån. Därefter började vi förklara syftet med intervjun och förklara varför de valdes som informant och hur deras svar kommer att användas. Detta gjordes för att göra informanten mer bekväm genom att inte dölja våra avsikter (Patel & Davidson 2011). Dock undvek vi att nämna begreppet TD utan berättade att vi vill veta mer om hur de arbetar på IT-företaget. Detta gjordes för att inte riskera att svaren blir påverkade. I slutet av intervjun togs det upp TD med en förklaring om vad det innebär för att öppna en diskussion med informanten om dess åsikter.

Innan intervjun påbörjades frågade en av forskarna informanten om dess inspelningsmedgivande som samtliga informanter skrev på. Se bilaga 2 för inspelningsmedgivande. Informanterna informerades om att deras svar kommer att behandlas anonymt i studien och att de enda som kommer ha tillgång till inspelningen är huvudforskarna. Anledningen till detta var att

följa god forskningsetik (Bell 2014). Intervjun påbörjades med generaliserade frågor med fokus på bakgrundsvariabler som namn och position, där friheten låg hos informanten hur väl djupgående informanten ville gå. Därefter ställdes frågor om deras sätt att arbeta. Då informanterna var väldigt öppna och delade med sig sina erfarenheter ställdes det många följdfrågor, vilket hade sina fördelar för undersökningen, men i efterhand ansågs det att vissa av följdfrågorna ej var relevanta för frågeställningen.

3.3.5 Dokumentation

Under genomförande av intervjuerna spelades de in med två olika inspelningsenheter. Detta för att säkerställa en redundans. Om en av enheterna inte spelade in fanns det en reserv att falla tillbaka på. Inspelning valdes av två anledningar. Dels kunde forskarna fokusera på att förstå informantens svar och ställa lämpliga följdfrågor i stället för att dela fokus på både ta anteckningar och frågor (Bell 2014). För det andra blir analysen lättare då det fanns möjlighet att gå tillbaka och lyssna på svaren igen vilket säkerställer att inget har glömts och säkerställer ett helhetsperspektiv för IT-företaget.

När intervjuerna var klara behövdes det råa materialet bearbetas, vilket genomfördes via transkribering som senare analyserades, kategoriserades och därefter gjordes det ett urval av vilka citat vi tog med i resultatet. Transkriberingen skedde på ett noggrant sätt beskrivet av Bailey (2008). Vi valde att skriva av intervjuerna ord för ord för att säkerställa att ingen information försvann från intervjuerna. Varje intervju fick sitt eget dokument för att förenkla hanteringen och sökningen av informationen vid analys.

Metodiken kring rapporteringen av undersökningen var främst baserat på Patel och Davidson (2011) och Bell (2014) som underlag och vägledning. Vi diskuterade också med vår handledare om möjliga upplägg.

3.4 Analys

När alla intervjuer var klara analyserades datan från inspelningen och transkriberingen. Transkriberingen av materialet gav nya insikter och en förbättrad förståelse för svaren. Bailey (2008) säger att transkribering ger en noggrann inblick som kan ge oförutsedda upptäckter. Eftersom frågorna var utformade efter det teoretiska ramverket och följdfrågorna lite mer öppna behövde vi kategorisera svaren enligt ramverket. Analysen skedde via ramverkets typer, aspekter och klassificeringar samt motiveringar till varför de tog ett visst beslut och resultatet därefter där vi diskuterade hur svaren passade in på ramverket. Baserat på svaren och dess kategorisering såg vi trender och samband mellan beskrivningarna informanterna gav – vilket styrker evidensen av vad som sker inom företaget.

Det teoretiska ramverket användes som ett utvärderingsverktyg för att kunna klassificera de olika svaren från informanterna. Alla typer, aspekter och klassificeringar var listade på en tavla med svaren åtkomliga, vilket gav ett tydligt förhållningssätt för att göra en godtycklig

bedömning. Därefter analyserades transkriberingarna för att hitta orsaken till varför de valde att göra på det sättet samt vad utfallet blev därefter.

4 Resultat

I detta avsnitt kommer resultatet från undersökningen att presenteras. För att besvara frågeställningen gjordes semistrukturerade intervjuer där frågorna var utformade att svaren kunde analyseras med hjälp av ramverket presenterat i avsnitt 2.2. Frågorna var inriktade på att svara på hur företaget arbetar när de utvecklar deras produkt. Detta format speglas i hur vi presenterar resultatet. Strukturen på upplägget är att presentera hur de på IT-företaget arbetar vid utvecklandet av deras produkt. För att göra det mer lättförståeligt har det delats upp i tre stadier planering av arbete, utveckling och testning och dokumentering. Vid slutet av intervjuerna presenterades begreppet TD för informanterna och förklarade vad det innebar. Detta avsnitt avslutas med att presentera deras tankar kring TD.

Eftersom intervjun från informant 2 gjordes på engelska kommer alla citat tagna från den intervjun att vara på engelska. Detta görs för att vi såg att när vi översatte hens citat till svenska tappades känslorna och budskapet i citaten. För att bibehålla informantens ursprungliga svar och inte riskera att förvränga resultatet valdes det att inte översätta.

4.1 Planering av arbete

IT-företaget har två huvudsakliga källor till information om vad de borde arbeta på att implementera i systemet. Informant 3 arbetar mycket med kunderna och tar emot önskemål och buggrapporter. Den här informationen blir därefter sparad i form av tickets i deras interna system där de kan därefter ges en prioritet. Informant 4 arbetar med att samla in information om hur marknaden ser ut och hur deras produkt jämförs med andra på marknaden.

För att planera vilka buggar de ska åtgärda eller vilka nya funktioner de ska implementera finns det två olika möten. Det första mötet sker på en mer daglig basis där informant 3 och kollega pratar med informant 1 om vad kunderna har sagt och bestämmer därefter vad de kommer att fokusera på. De beskriver att om en bugg är återkommande eller om den är synlig för användaren att de lägger fokus på att lösa den men om en bugg är inte synlig för användaren eller inte uppstår ofta att de skjuter på att åtgärda det.

> *"[...] är det en bugg som användarna märker, eller är det en bugg som användarna inte märker? Och ja, buggar som användarna märker, de tar vi direkt."* (Informant 1)

> *"Är det något som är uppenbart en bugg som gör att systemet inte fungerar som det skall så åtgärdas den alltid innan vi släpper."* (Informant 4)

> *"Om en bugg är återkommande, om en bugg inte är synlig, men återkommande, då gör vi någonting åt det. Om det är en enstaka och inte av något större problem så gör vi egentligen ingenting åt det förrän det kanske kommer gång nummer två. Eller liksom tre."* (Informant 1)

> *"Då blir det mer att man prioriterar bara det som är prio."* (Informant 3)

> *"[...] sitter jag med tio osynliga buggar, då bryr jag ju inte mig om dem, då sitter jag och utvecklar vår nya fakturamodul i stället — för att få ut den så snabbt som möjligt, och sen när den är ute, ska den ligga och skvalpa ett tag och då kan jag fixa de här tio buggarna i stället."* (Informant 1)

Det andra mötet är ett produktionsmöte där hela företaget samlas för att diskutera vad som skall göras, vad som borde prioriteras och andra synpunkter på produkten och utvecklingen.

> *"[...] på fredagar så har vi försökt att ha produktionsmöten för att få en dialog på vad som utvecklas, vad som kommer att utvecklas, osv."* (Informant 3)

> *"[...] vi har produktmöten där informant 1 och informant 2 sitter med från tekniksidan sen sitter även kollega med från säljsidan och informant 3 från support och så jag."* (Informant 4)

Men de berättar att de gemensamma mötena inte alltid sker och att de senaste tre månaderna har det inte varit något möte alls.

> *"Det kommer alltid finnas undantag som gör att en dag så är inte informant 1 på jobbet eller en dag har vi väldigt mycket att göra eller vad det kan vara. Så det händer inte alltid."* (Informant 4)

> *"[...] it was not like everybody meeting. [...] I would have loved to like have like monday meeting where informant 3 is there and this is the problem and this is the highest one how long do we think we can get it done [...]"* (Informant 2)

Informant 3 beskriver att det har funnits en tidsbrist under senare tiden vilket har påverkat deras arbete. Det leder till att strukturen inte följts till sin helhet. Samtidigt som det blir frustrerande för informanten.

> *"[...] väldigt mycket tidsbrist bara att det har varit många olika typer av saker."* (Informant 3)

> *"[...] just nu har vi direkt ingen bra struktur så därför blir det att när det blir tidsbrist att man slarvar med de strukturerna man har, så då blir inte lika dokumenterat av det man gör eller om man ska lösa ticketsarna, men om man inte gör det så ligger ticketsarna och bara samlar damm om man säger så. Och då tar det lång tid innan man lösmarkerar ticketen."* (Informant 3)

"Det är både kul men ibland också frustrerande." (Informant 3)

Efter att all planering har skett tas besluten om vad det är som ska utvecklas av informant 1.

> *"[...] sen så är det upp till informant 1 att säga ja eller nej om det här behöver göras eller det här får vänta, det här gör vi nu det här gör vi sen."* (Informant 4)

4.2 Utveckling och testning

I det här avsnittet kommer det att presenteras resultatet tillhörande IT-företagets utvecklingsprocesser och testning. Utvecklingen av produkten görs utav informant 1 och 2 vilket är orsaken till varför de två används oftast som källa till citat. Det är de med kunskapen inom området. Men vid testning är det fler aktörer involverade.

4.2.1 Utveckling av produkten

Vid frågan om hur det arbetar vid utveckling av deras produkt beskrev två informanter att deras fokus är att få det att funka först och göra det korrekt senare. Att det är viktigare att funktionen gör det den ska i stället för att koden är perfekt från start. Detta görs för att få snabbare utvecklingstider.

> *"Om man bygger jätteful kod, jag säger till och med till informant 2 om det… bygg nu bara, make it work. Så att du får punkt a till ö. För det största misstaget är när utvecklare sitter och tror att de ska skriva den perfekta koden."*
> (Informant 1)

Vid frågan om vad de tyckte om detta arbetssätt beskrev informant 2 blandade känslor.

> *"I didn't really want to do that but unfortunately I had to do it because being new in a company you want to do your best [...]"* (Informant 2)

> *"[...] just do it this way, ok if you say so. Even though I know like it's not the best way [...]"* (Informant 2)

De arbetar dock aktivt med att gå tillbaka till koden och förfina det så att det kan bygga nya funktioner med den koden. Informant 1 beskriver om det går att bygga ny kod på den koden har du skrivit bra kod. Men att de inte städar all kod utan bara det som behövs.

> *"Sen när du bygger kod ovan på den koden, då måste du fixa denna så den är flawless. Och saken är den, den definitionen är också, om du har byggt kod, som någon kan stå på tre gånger om, då har du byggt väldigt bra kod."*
> (Informant 1)

> *"[...] jag sätter mig inte och bygger, om jag har fyra sådana här i botten, och det bara de här två vi använder hela tiden, då sätter jag ju inte mig och städar de andra… total waste of time."* (Informant 1)

IT-företaget är i en process där de håller på att bygga om deras front-end. I detta projekt använder de många tredjepartslösningar när de implementerar nya funktioner. Informant 1 gav följande svar till varför de inte väljer att utveckla komponenterna själva.

> *"[...] det är ett hål i huvudet. För det är så mycket… Om några sitter och bygger en tabell, det är det enda de gör. Vad är det som säger att vi skulle bygga det bättre? Det är ju det som är det hela med Git, själva crowdidén."*
> (Informant 1)

Hen förklarar även att de bedömer risken att vara lägre genom att använda dem kontra bygga eget eftersom tredjeparten har med stor sannolikhet mer kunskap i hur komponenten borde byggas upp. Informant 1 är medveten om riskerna att förlita sig på tredjepart men anser dem relativt låga.

> *"Visst kan du få problem med dem också, men risken bedömer jag som lägre."*
> (Informant 1)

Denna syn delas dock inte i sin helhet med andra utvecklaren informant 2 då hen hade föredragit att skapa mycket själv samtidigt som hen tycker att tredjepart medför en ökad risk för man låser in sig.

> "*I would love to more of the design but with the third libraries nowadays I haven't been so much into design. We have what we wanna do and we use them."*
> (Informant 2)

> *"There is always a risk in relying on third parties because you get locked in unfortunately."* (Informant 2)

Men hen håller med informant 1 om att det är ett väldigt effektivt sätt att få produkten fungerande för att få ut den på marknaden. Men håller fast vid att därefter borde det läggas tid på att utveckla det själv.

> *"It's true, it's an easy way of like getting MVP out (Minimal Viable Product). But once you feel like I'm a company that is established I think it's better to always think like it's good but we have to create some things ourselves."*
> (Informant 2)

En till anledning till varför Informant 2 har en negativ inställning till tredjepartsbibliotek är deras dokumentering.

> *"[...] I did a lot with React... Facebook's React and the point is they both have a very good documentation but once you start with production there are a lot of front-end frameworks Material, Angular, Gant, that you would think that they have good documentation like Google has and Facebook has. It's terrible."*
> (Informant 2)

"Those who are very good documentation you don't have a dick. It's just seamless. Then you doing something seamless and you so happy and then you meet something and the documentation is total crap. You go online, the help that you get is not so good so." (Informant 2)

4.2.2 Testning av produkten

IT-företaget använder sig idag av manuella tester för deras webbapplikation och mobila applikation. De är medvetna om riskerna med att använda manuella över automatiserade men det finns inte resurser att automatisera dem. Det beskrivs också att det är väldigt lite test av webbapplikationen att det i stället läggs mer fokus på att testa mobilapplikationen. Men beroende på storleken av en uppdatering kan det göras djupare testning. Dock fortfarande manuell.

"[...] generellt är det väldigt lite test på webben." (Informant 3)

"Right now everything that we do is manual testing [...] unfortunately it's still manual testing." (Informant 2)

"Vi kör manuellt." (Informant 1)

"[...] from everything that I've learnt it's not efficient because there are bugs that definitely informant 3 would not see until it gets to real production. And I'm sure that this could cause recurring problems like we fix something we think it's done we send it out, it opens another problem and because we haven't really tested very well so yeah." (Informant 2)

"[...] it's gonna be like a lot of hours to write a test and then write code for that and start testing. So I know like, it's not efficient." (Informant 2)

"Bättre är de inte. Eller man borde ha båda. Både automatiserade och manuella tester men det är en fråga om resurser." (Informant 4)

Vid fråga om det finns någon struktur till hur testningen skall gå till gavs det blandade svar. Innan var det väldigt ostrukturerat över hur testningen borde ske men med tiden har det blivit mer strukturerat.

"[...] vi hade inga tester helt enkelt, vi körde bara ut det. Alltså, utvecklaren testar basic och sen så kör man ut det." (Informant 1)

"Det är ganska icke-existerande kring någon form av struktur på det kan man säga. [...] När det kommer vi har inga klara processer kring det." (Informant 4)

"Nu har man lärt sig att man liksom måste man verkligen ha ett schema när man ska testa." (Informant 3)

Det gavs tre exempel på problem där det potentiellt inte hade varit ett problem hade testningen varit mer genomgående. Det första problemet var att auto-fill i webbläsaren överskred all information användaren hade.

> *"Han hade auto-fill på hela formuläret, på hela fakturaformuläret, så när han öppnade fakturakomponenten, och valde sitt projekt, då "overridade" autofill all form av data i hela formuläret. Sen säger användaren, jag har ju valt projektet, men… allting som visas här är något helt annat?"* (Informant 1)

Det andra problemet var att de hade fått en ny version av applikationen från deras konsulter till Apples mjukvara och efter några få tester verkade den fungera som den ska. IT-företaget skickade in uppdateringen till Apple och när användarna fick den märkte de att på en viss hårdvara med en viss mjukvara fungerade inte applikationen. Den startade och sen kraschade.

> *"Vi hade testat appen innan och hade funkat jättebra. Jag fick bara tillsagt till mig att vi har inte gjort någonting som kommer förändra någonting. Så jag sa okej vi releasar den. Så vi skickade in den till Apple och Apple tar mellan två och fem dagar på att få ut en appversion. När den kom ut vilket var en fredag eftermiddag naturligtvis så funkade inte appen."* (Informant 4)

När vi frågade informanten om hens åsikter om situationen var det inte positivt.

> *"Jag var mest irriterad på mig själv för att inte hade testat appen."*
> (Informant 4)

Det tredje problemet var att offertmodulen inte fungerade enligt önskemål. Detta resulterade till mycket städning av koden och ett minskat förtroende från en av deras kunder där de arbetar mycket med offerter.

> *"[...] de litar inte på det nu som de gjorde tidigare."* (Informant 3)

Efter att företaget har testat produkten inför release och den bedöms klar skickas den till en eller flera testkunder där den verifieras under en kortare period. Därefter bedöms den redo för användarna och blir tillgänglig för alla användare.

4.3 Dokumentering

I detta avsnitt presenteras informationen och svaren informanterna gav angående hur de arbetar med dokumentering i organisationen. Det är uppdelat i dokumentering under utveckling och under test.

4.3.1 Dokumentering under utveckling

Detta avsnitt kommer att behandla svar från informanterna angående hur de dokumenterar i deras utvecklingsprocesser. När det kommer till rapportering och dokumentering av kundernas buggar och vilka det är som ska åtgärdas först hanteras genom deras interna system med tickets.

> *"[...] buggar rapporteras förhoppningsvis i form av tickets [...]"* (Informant 4)

När det kommer till dokumentering i utvecklingsprocessen är det blandat. Vid frågor om de skriver dokumentation för koden fick vi två olika svar från informanterna.

> *"[...] vi har inte fått någon best practice [...] så det är svårt att göra någon form av dokumentation. Och när det gäller dokumentation, så är det nog highly överskattat, skulle jag säga."* (Informant 1)

> *"Yeah, I write a lot of comments. It's a practice that I've gotten used to since I was trying to create my own app. Because I know that even if it's just you sitting in the code you could come next week and be like did I write this? And you wrote it. Nobody else it's just me in the code. So I try as much as I can to write comments [...]"* (Informant 2)

Efter några följdfrågor kom det fram att även informant 1 skrev kommentarer i koden men att hen verkade ha en negativ inställning till kommentarer eftersom de blir utdaterade väldigt snabbt eftersom koden är i konstant förändring.

> *"[...] ja det har vi och de blir också utdaterade. Det är ju det som är problemet. Så när du skriver en kommentar den kan ju ändras." (Informant 1)*

Men till skillnad från informant 1 har informant 2 en väldigt positiv inställning till kommentarer i kod och beskrev till oss hur det kom att vara så; att det finns skillnad på en bra kommentar och en dålig kommentar.

> *"[...] I learned from Coursera and [...] the man explained so hard this is the importance of comment. And that is what I realize like even the comments that I see now it's like you come to a page and you see like comment about the whole page. In my world that is information that is not comment, it's information about the whole page. Then the comment should be like maybe on this function, what the function is doing, what is related to."* (Informant 2)

Mycket av kommunikationen mellan utvecklarna (informant 1 och 2) sker verbalt då de sitter bredvid varandra. Det finns mindre dokumentation om hur specifika funktioner skall vara designade eller hur de borde kodas. Det gör att informant 2 behöver fråga informant 1 om vad det är hen ska göra och hur hen ska göra det. Informant 2 hade föredragit en mer djupgående kravspecifikation på funktioner så hen behöver inte störa informant 1.

> *"You know it's easy when it's listed so while it's busy I don't need to disturb him I can jump and do something else while he's busy and when he comes to me and asks if I need help I can tell him yeah I have a problem. So it makes development faster [...]"* (Informant 2)

Något vi märkte under intervjuerna var att mycket av information i företaget är hos informant 1. Det finns väldigt lite nedskrivet om hur produkten är uppbyggd och hur man som ny skall komma in i projektet. En informant nämnde att det finns dokumentering för hur man startar projektet men att dokumenteringen för hur man arbetar i projektet är bristande. Informant 2 beskriver att skulle någon behöva sätta sig in i projektet för att få samma förståelse som informant 1 hade det tagit väldigt lång tid.

> *"[...] a lot of time, not even hours, maybe days [...]"* (Informant 2)

Informanten fortsätter med att säga att dokumenteringen hen fick när hen var ny gjorde hen nästan rädd för projektet. Men att under tiden började hen förstå koden och därefter var det inte lika läskigt utan det var bara kod. Dock höll hen med när det ställdes en fråga om dokumentationen hade varit bättre hade det varit enklare att komma in i projektet.

> *"[...] when I came I got scared of his code. I looked at his code and I literally got scared [...]"* (Informant 2)

Fortsättningsvis beskrev hen att när hen började med koden blev hen förvånad över hur vissa delar av koden såg ut. Det fanns delar som var lättförståeliga men att det fanns mycket som var svår att förstå.

> *"And going into his code it was not easy to understand unfortunately. Apart from easy to understand there were things that the way he looped through things looked like so så gammalt to me like are you really still looping this way."* (Informant 2)

> *"Sometimes I see very good code and sometimes I see very bad code."* (Informant 2)

4.3.2 Dokumentering under test

Under testning av den mobila applikationen ansågs det göras mycket dokumentering under processen men i verklighet togs det genvägar i arbetet. Dock är det bara under testning av mindre releaser dokumenteringen inte gjordes.

> *"[...] när det är större releaser då gör vi testningar och så dokumenteras det i dokument som vi skickar till dem som de får sen arbeta utifrån att prioritera vad det är som skall fixas om det är nu så kritiskt att det måste fixas innan vi gör release eller om det kan vänta till nästa release[...]" (Informant 4)*

> *"[...] när jag hittar buggar kommer oftast i någon form av Slack-konversation till informant 1. Inte optimalt men den enkla och snabba vägen." (Informant 4)*

Under följdfrågorna berörande dokumentering vid testning av produkten kom det fram att dels är det bristen på tid men även bristen på struktur som leder till att testningen inte dokumenteras. Testningen av webbapplikationen hanteras för det mesta av utvecklarna själva när de utvecklar nya funktioner eller åtgärdar buggar. Men vid större releaser testas den av andra anställda på företaget.

4.4 Organisationens syn på technical debt

Intervjuerna avslutades med en presentation av TD begreppet och vad det innebär. Vi frågade informanterna om vad de tycker om denna arbetsmetodik och fick varierande svar från informanterna. Cheferna hade en positiv inställning till TD men ena utvecklaren hade en negativ inställning till det.

> *"Jag skulle säga att det här är ett väldigt vanligt sätt att arbeta på i mindre företag. [...] Man har inte tiden att lägga på detta heller på samma sätt som man kanske har eller som det krävs också när man blir fler."* (Informant 4)

> *"[...] så är det när man är mindre företag, man måste prioritera vad som är viktigast."* (Informant 4)

> *"Det är en annan sak om du har resursen, om technical debt, om du har resursen till att definiera den exakta outputen då kan du sen utveckla, skriv… här borta ska du generera de här två fälten, med de här värdena baserat på det här. Ja, då kan en erfaren utvecklare fixa det med den perfekta koden direkt. Men när du jobbar med en plattform som tar in nya saker, nya hanteringar… då är det korkat att sitta med att skriva fantastisk kod och dokumentation — men om du vet om, som jag säger till informant 2, skriv vad du vill, bara jag får se "proof of concept"(PoC) på den här lösningen. För ofta sitter en utvecklare och skriver kod, och så skjuter han här (fel) i stället för här (rätt). Då är den första frågan, hur långt bak måste vi backa så att du kan skjuta rätt? Och då vill du inte gå långt bak för det tar en jävla tid. Så man vill ha PoC så förfinar du efter."* (Informant 1)

> *"[...] I think it's gonna be a big problem in the future [...]"* (Informant 2)

5 Diskussion

Denna studie har fokuserat på att undersöka hur TD kan appliceras på mindre mjukvaruföretag. Detta avsnitt kommer att presentera en analys och diskussion på resultatet och en jämförelse med hur IT-företaget såg på TD jämfört med teori. Därefter kommer det att diskuteras reflektioner kring studien och ramverket för att sedan avsluta med förslag till framtida forskning.

5.1 Analys av resultat

I detta avsnitt kommer resultatet från föregående avsnitt att analyseras och jämföras med det teoretiska ramverket presenterat i avsnitt 2.2. Informationen vi har fått från informanterna kommer att analyseras utifrån tre teman. Varför tar de på sig TD, vilka typer och klassificeringar är det samt vilka aspekter har de och hur påverkades företaget av TD? Patel och Davidson (2011) säger att diskussionen ska helst följa samma struktur med resultat avsnittet. Men vi har valt att inte följa samma struktur utan har valt att strukturera upp det med ramverket för att göra det lättare för läsaren att förstå hur TD kan appliceras. Eftersom frågeställningen syftar på hur TD kan användas för att beskriva beslutsfattande och utfallet därefter ansågs detta vara en enklare struktur för läsaren.

Strukturen kommer ske enligt följande. Först kommer det att diskuteras varför IT-företaget tog på sig TD. Eftersom det här området ligger först i ramverket är det logiskt att det presenteras först. Därefter kommer det att presenteras vilka klassificeringar, typer och aspekter vi har identifierat hos IT-företaget. Här vill vi förmedla på vilket sätt TD har etablerat sig i företaget. Avslutningsvis diskuteras utfallet av TD där det kommer att presenteras resultatet av att IT-företaget följt denna arbetsmetodik. Med hjälp av denna struktur anser vi det lättare för läsaren att få en förståelse för hela TD-cykeln hos IT-företaget och på det sättet enklare att få en förståelse för varför TD, vad för TD och utfallet av TD. Analysen visualiseras i 5.1.4 där vi återanvänder tabell 1 för att visa vilka delar av ramverket vi har lyckats identifiera.

Analysen kommer att utgå ifrån vår analys av informanternas svar där vi kopplar informationen med ramverket presenterat tidigare. Med hjälp av resultat och teori är målet att ge en förståelse på hur TD kan användas för att beskriva arbetsprocessen hos mindre mjukvaruföretag. I avsnitt 5.1.1 kommer det användas stöd från avsnitt 2.2.4 varför och 2.1.2 mindre mjukvaruföretag. Avsnitt 5.1.2 använder stöd från avsnitten 2.2.1 klassificeringar, 2.2.2 typer och 2.2.3 aspekter. Vid diskussioner kring resultatet av beslutet att ta på sig TD kommer det kopplas till teori i avsnitt 2.2.5 resultat av TD.

5.1.1 Varför tar organisationen på sig technical debt

Tom, Aurum och Vidgen (2013) beskrev att det fanns olika anledningar till varför organisationer tar på sig TD. De är prioriteringar, pragmatism, processer inom företaget och personalens attityd och kunskap. Med hjälp av dessa anledningar och svar från intervjun är målet att beskriva varför IT-företaget tog på sig TD.

En av de stora riskerna hos mindre mjukvaruföretag är att marknaden är i konstant förändring där det krävs att företagen är snabba på att reagera. Samtidigt som det finns ett beroende på att få nya kunder då det inte finns den stora mängden resurser större företag har (Giardino et al. 2014). Mindre mjukvaruföretag behöver göra en prioritering över vad det är som behöver göras och skall göras först. Detta går att se hos IT-företaget där de gjorde valet att prioritera snabbhet över kvalitet. De behöver få ut ny funktionalitet för att behålla marknadsandelar och det gjordes på bekostnad av andra aspekter av produkten. Vi anser också att mycket av besluten tagna hos IT-företaget bygger på pragmatism. När cheferna presenteras med ett val utgår de från magkänsla i stället för rutiner. Exempel på detta är när chefen för utveckling (informant 1) fick välja mellan att åtgärda tio buggar eller utveckla ny funktionalitet. Hen valde ny funktionalitet för det är vad gynnar kunden nu. Buggarna kan de hantera vid ett senare tillfälle.

Det vi såg var att det fanns en brist på struktur inom företaget. Det fanns inga etablerade regler till hur exempelvis testning och dokumentering skulle ske och när det fanns regler var det inte alltid de följdes. Utan de gjordes utefter medarbetarnas egna regler. Denna brist på struktur kunde även ses i företagets testningsmetodik. Det fanns inga etablerade protokoll för hur testning borde ske – utan man utgår från medarbetarnas egna regler. Vi såg även att det fanns en brist på kunskap i företaget till hur man borde arbeta för att förebygga TD. Framförallt hos de som inte jobbar direkt med utvecklingen av produkten – utan hade mer supportroller i organisationen.

En utav de stora anledningarna till varför IT-företaget tar på sig TD begrundar sig i attityden hos medarbetarna – framförallt cheferna. Det vi såg var att både VD:n och chefen för utvecklingen hade en positiv inställning till ett arbetssätt med TD. De upplevde inga problem med att prioritera vissa processer över andra. Exempelvis att VD:n under hens testning av den mobila applikationen hade inga problem med att bara göra en minimal testning innan den skickades till Apple för publicering. Kopplad till chefen för utvecklingsfokus på att få koden att fungera först och få det korrekt senare samtidigt som hen anser att dokumentering inte är viktigt. Går det att dra direkta kopplingar till motiveringarna bakom deras arbetsprocesser med anledningarna ramverket presenterar.

Det finns en anledning från ramverket vi inte anser är en orsak till TD hos IT-företaget. Tom, Aurum och Vidgen (2013) beskrev ett fall där en medarbetare är medveten om vad de gör men inte bryr sig om konsekvenserna. Detta kunde inte hittas stöd i vår analys och kan därför inte säga att detta är en orsak till varför TD uppstår. Dock vill vi poängtera att bara för att det inte gäller i det här fallet, betyder det inte att det inte kan ske i andra fall.

5.1.2 Vilka klassificeringar, typer och aspekter

Detta avsnitt kommer att behandla vilka klassificeringar, typer och aspekter av TD det går att identifiera utifrån resultatet. Avsnittet baseras på svaren från informanterna där de har jämförts mot ramverket presenterat i avsnitt 2. Det som är viktigt att ha i åtanke är att informanterna inte har berättat att det är exakt så här det är. Utan vi har gjort vårt bästa att koppla respondenternas svar till den korrekta benämningen.

5.1.2.1 Klassificeringar av technical debt

Utifrån svaren från intervjun har vi kunnat identifiera att samtliga klassificeringar går att identifiera hos IT-företagen. Det togs aktiva beslut om att prioritera vissa processer av utvecklingen över andra för att få vinster nu. Detta karakteriseras med strategisk TD i ramverket där det är ett mer långsiktigt perspektiv. Företaget var medveten om att de kommer behöva gå tillbaka och betala tillbaka sin TD men valde att ta på sig en skuld för de såg fördelarna. Ett exempel på det är när informant 1 bad informant 2 att inte försöka skriva den perfekta koden från början – utan att få det att funka först så att det finns en funktion de kan få ut till kunden. Därefter går de tillbaka och städar koden.

Det kunde även dras kopplingar till den mer kortsiktiga taktiska TD. De hade en väldigt hög prioritet att få ut produkten till kund och så länge den inte hade några kritiska problem släpptes produkten först och åtgärdades därefter. Detta går att se genom exempelvis mängden testning de gör. Informant 1 beskrev hur det var väldigt lite testning innan release. Likt hur mindre mjukvaruföretag karaktäriseras – där det är stort fokus på att få ut produkten så snabbt som möjligt för att säkerställa kunder (Giardino et al. 2014). Man kan även se hur de använder det som en form av reaktions TD där även om det uppstår buggar vid testning väljer de att släppa den på marknaden så länge buggarna är osynliga för användarna.

Under analysen av resultatet kunde slutsatsen dras att de arbetar mycket med att ta många små genvägar eller lån för att koppla det till TD. Ramverkets benämning är inkrementell TD och McConnell (2007) drog parallellen med kreditkortslån; att det är en sammanställning av många små uttag. Det vi kunde se var att det togs många små genvägar i att exempelvis följa kodrutiner, dokumenteringsrutiner och testningsrutiner. Vilket gjorde att varje gång en liten genväg togs byggdes denna skuld upp.

Den sista klassificeringen vi identifierade var oavsiktlig TD. Vi såg att vissa beslut från medarbetarna hade en påverkan på mängden TD men att de inte gjorde ett aktivt val att öka skulden. Ett exempel på detta var när informant 4 valde att inte dokumentera fullständigt vid testning av den mobila applikationen utan skickade ett meddelande till informant 4. Det var vanligt att rutiner och best practice inte följdes helt – oftast var det inte ett medvetet val att öka företagets TD utan var en effekt de inte hade tänkt på.

Slutligen kan det sägas att samtliga klassificeringar befinner sig i företaget men att mängden hos de olika klassificeringarna varierar. Utifrån vår analys anser vi att majoriteten är strategisk TD och taktisk TD.

5.1.2.2 Typer av technical debt

Det har presenterats olika typer av TD från både Tom, Aurum och Vidgen (2013) och Li, Avgeriou och Liang (2015) i avsnitt 2. Avsnittet kommer att diskutera vilka typer vi har identifierat utifrån informanternas svar och motiveringarna bakom besluten. De typerna vi har tittat efter är krav TD, arkitektur TD, design TD, kod TD, test TD, bygg TD, dokumentations TD, infrastruktur TD, versionshanterings TD och defekt TD. Något vi vill poängtera är att det som vi presenterar nedan är de typerna vi har identifierat utifrån intervjusvar. Bara för att en typ inte lyfts här betyder inte att den inte är förekommande i organisationen eller i andra mindre mjukvaruföretag.

Vi ser idag att deras arbetssätt för kodning stämmer överens med definition för kod TD. Där deras beslut att inte skriva perfekt kod från början bidrar till ökningen av projektets skuld i form av kod TD. Informant 1 var väldigt öppen med deras arbetssätt – vilket gjorde det enkelt att identifiera denna typen av TD.

Den andra stora typen av TD vi har identifierat är dokumentations TD. Det vi kunde se var att trots informant 2 positiva inställning och dedikation till dokumentering fanns det fortfarande en brist på dokumentation. Informant 1 hade en väldigt negativ inställning till dokumentering – vilket leder oss till att dra slutsatsen att hen inte gör det trots att hen höll med att kommentarer i kod är viktiga. Men i det här fallet var det inte bara ett enstaka fall utan majoriteten av företaget hade bristande dokumenteringsrutiner.

Test TD är en av de vi anser skulle få störst påverkan om de hade valt att ändra rutin. Teorin beskriver test TD med suboptimala procedurer (Tom, Aurum & Vidgen 2013). Det vi såg i resultatet var att dels använder de manuella tester i stället för automatiserade trots deras medvetenhet att manuella är sämre. Men även att buggar gick igenom tester och hamnade hos användaren. Framförallt att vissa kritiska buggar gick igenom var en stor anledning till varför test TD identifierades som en av typerna.

Även om de inte har orsakat några stora problem än har vi valt att medräkna användandet av tredjepartsbibliotek som en form av TD. Det kategoriseras som en form av design TD men vi anser att mängden TD inte är av stor mängd vid det här tillfället. Motiveringen till varför vi valde att räkna med det beror på två saker. Dels har det beskrivits av informant 2 att många tredjepartsbibliotek har bristande dokumentering vilket gör dem svåranvända – vilket leder till att de blir svåra att underhålla, en nyckelaspekt i design TD. Andra motiveringen är att användandet av tredjepartsbibliotek låser in projektet. Det skapas ett beroende till tredjeparten och när det går bra är det inga problem men risken finns alltid och därför är den med på listan. Vi håller dock med informant 1 att risken är relativt låg vid användandet av bibliotek från etablerade källor.

Den sista typen av TD vi kunde identifiera var defekt TD. Utifrån resultatet gick det att identifiera eftersom produkten IT-företaget utvecklar har problem med buggar och defekter. Under

analysen av resultatet gick det att se att bristen på mer djupgående tester bidrog till att buggar kom igenom och ökade produktens defekt TD.

Vid analys av resultatet kunde vi inte identifiera med säkerhet att det fanns krav TD, arkitektur TD, bygg TD, infrastruktur TD eller versionshanterings TD. Detta betyder inte att de inte är aktuella i mindre mjukvaruföretag eller IT-företaget men att frågorna och informanternas svar följde en annan riktning än den som leder till de här typerna.

5.1.2.3 Aspekter på organisationens technical debt

Utifrån resultatet identifierade vi fyra av ramverkets aspekter till IT-företagets TD. Vi identifierade en monetär kostnad, ränta och återbetalning av TD, leverage och McConnells (2007) kreditkorts associering där TD ses som många små uttag och återbetalningar.

Under resultatet går det att se att mycket tid har behövt läggas ner för att åtgärda buggarna orsakade från deras beslut. Den här tiden kostar företaget pengar i form av lön till utvecklarna. Även om de sparar tid och pengar på att minimera dokumentering och testningstiden blir det fortfarande en monetär kostnad i slutändan som läggs på utvecklarna. Det är inte bara buggar som kräver arbete; utan som informant 1 beskrev uppmuntras det att skriva kod att funktionen funkar först sen därefter gå tillbaka och renskriv det. Den tiden det tar att renskriva kod kostar företaget pengar som kunde har lagts på utvecklandet av nya funktioner eller exempelvis mer tid för testning.

Det gick att se hur företaget arbetade aktivt med att betala tillbaka deras skuld genom det Tom, Aurum och Vidgen (2013) benämner som paying back the principal. Samt att det fanns en ränta i form av en påverkan på medarbetarnas moral, produktivitet och kvalitet på produkten. En mer djupgående diskussion om effekterna av TD i organisationen kommer i avsnitt 5.1.3. Ett utav de viktigaste reglerna med TD är att betala tillbaka sin skuld. Det vi såg var att de arbetade med återbetalning – men bara när de behövde. Utifrån resultatet drogs slutsatsen att om inte de behöver gå tillbaka kommer de inte göra det. Att de valde att fokusera på nya funktioner i stället för att åtgärda buggar eftersom det inte var ett problem vid det tillfället. Det tyder på att buggarna bara kommer att lösas när kunderna börja reagera på dem men tills dess kommer de ligga kvar. Ett annat exempel kommer från informant 1 där hen beskrev att om inte de kommer att bygga vidare på koden kommer det inte att läggas tid på att renskriva det. Anledningen till detta har att göra igen med prioriteringar. Mindre mjukvaruföretag måste prioritera vad de ska lägga resurser på och här väljer de att lägga tid på saker användaren kommer se nu.

Utifrån resultatet går det att dra kopplingar till Tom, Aurum och Vidgens (2013) leverage – där vi kunde klart och tydligt se att de använde genvägar för att få kortsiktiga fördelar. I det här fallet var den fördelen en ökad produktivitet. Genom att öka produktiviteten kunde de leverera sin produkt till användarna på kortare tid. Exemplen från resultatet är mindre testning av mobilapplikationen, mindre dokumentering och en prioritering på fungerande kod över ren kod.

Vi kunde se hur mycket av IT-företagets TD var av en mindre och mer oplanerade typ. Lik McConnells (2007) liknelser att eftersom det var så enkelt att ta genvägen så gjordes det. Exemplet att informant 4 väljer att skicka hans resultat genom ett chattmeddelanden till utvecklaren efter testning av applikationen i stället för att göra en ticket stödjer vårt påstående. Eftersom det finns mycket små TD blir det svårt för företaget att identifiera all TD utan en del hamnar utanför organisationens synfält.

Två aspekter vi inte kunde identifiera var amnesty och att ett projekt går i konkurs på grund av TD. Vi kan inte konkret säga varför men en hypotes är att eftersom de är fortfarande relativt nya på marknaden har det inte behövt ta ett beslut med dem alternativen än.

5.1.3 Vad blir utfallet av technical debt

Det har beskrivits kort ovanför om effekterna av företagets TD men i detta avsnitt kommer det att diskuteras mer genomgående. Tom, Aurum och Vidgen (2013) säger att TD har en påverkan på personalens moral, deras produktivitet, produktens kvalitet och projektets risk. Det har identifierats samtliga effekter av TD hos IT-företaget och en diskussion kring implikationerna presenteras nedanför.

Från resultatet går det att se att arbetsmetodiken hos IT-företaget både har en positiv och negativ påverkan på moralen hos anställda. Vi ser att informant 2 likt det Tom, Aurum och Vidgen (2013) säger är stolt över sitt arbete och gillar inte att producera kod hen inte är stolt över. Generellt var moralen hög under tiderna genvägarna togs. Men när de var tvungna att hantera problemen som uppstod långsiktigt sjönk moralen. Exempel är när den mobila applikationen inte fungerade och informant 4 beskrev hur frustrerad hen var på sig själv. Vi såg även hur ackumuleringen av genvägar i koden ledde till att informant 2 blev rädd när hen såg koden för första gången.

Genom att de på IT-företaget arbetade med prioriteringar och genvägar kunde vi se att de fick en ökning i produktivitet på kort sikt. De kunde få nya versioner av applikationen till användaren snabbare på bekostnad av den långsiktiga produktiviteten. Även om vi såg att de inte valde att åtgärda buggar och problem direkt går det att anta att det är bara en tidsfråga tills snöbollseffekten har blivit tillräckligt stor att halta utvecklingen av nya funktioner. Vi har sett glimtar av den effekten exempelvis när webbläsarens auto-fill skapade problem med textinnehållet och problemen med den mobila applikationen. Men vi såg ett exempel från informant 2 där det visar vilken effekt bristande dokumentering får på produktivitet. Hade dokumenteringen varit tillräcklig hade upplärningstiden kunnat förminskas och om dokumenteringen för hur nya funktioner borde byggas upp var bättre hade de minskat tiden där informant 2 behövde vänta på informant 1.

TD får inga positiva effekter på kvalitet utan kvalitet får alltid en negativ påverkan (Tom, Aurum & Vidgen 2013). Något vi instämmer med efter analysen av resultatet. IT-företagets produkt fick alltid en sämre kvalitet på grund av besluten att ta genvägar. Det grundar sig i att under alla fyra intervjuer var det ingen som berättade om hur deras beslut gjorde produkten

bättre från ett kvalitetsperspektiv. Utan allting handlade om att öka produktiviteten på bekostnad av kvalitet. Besluten av att testa mindre, skjuta upp buggar och lägga mindre tid på dokumentering har alla resulterat i att kvaliteten har påverkats negativt. Det är negativa konsekvenser både kortsiktigt och långsiktigt.

Utifrån resultatet drogs slutsatsen att projektets risk inte hade påverkats på samma nivå som de andra kategorierna men att den fortfarande påverkades. De beskrev på IT-företaget att med hjälp av att skjuta upp vissa uppgifter kunde de få ut nya versioner i tid och på så sätt hålla projektets tidsram. På det viset går det att kategorisera att projektets risk minskas kortsiktigt men att den extra arbetsbördan inför nästkommande version ökade den framtida risken. Det intressanta dock är användandet av tredjepartsbibliotek i produkten. För tillfället är det en stor fördel då det hjälper företaget att implementera nya funktioner utan att behöva bygga dem från grunden. Men som informant 2 beskriver blir de låsta till dem. Skulle något stort förändras från leverantören finns det stor chans att projektets risk ökar enormt.

5.1.4 Sammanställning av analys

För att göra det lättare för läsaren att förstå resultatet och diskussionen har vi tagit tabell 1 från teoriavsnittet och ändrat den till att nu spegla vilka motiveringar, klassificeringar, typer, aspekter och utfall vi har identifierat (se tabell 2). Allt vi har identifierat återspeglas i tabellen fetmarkerat. Det vi kan se är att det har identifierats samtliga motiveringar till varför de ådrog sig TD och att alla klassificeringar finns. Vid identifiering av typer och aspekter hittades det många men inte alla. Till sist lyckades vi hitta stöd att TD har påverkat IT-företaget på alla delar ramverket presenterade.

Det vi kan se är att många av besluten tagna har att göra med att IT-företaget är ett mindre mjukvaruföretag. Deras beslut grundar sig i att till skillnad från större företag har de begränsat med resurser. De väljer att inte testa produkten mer genomgående, inte för att de vill att produkten skall innehålla buggar utan för att de måste. Det läggs en prioritering på att lägga resurser på att utveckla i stället för att testa. Detta tankesätt fortsätter med deras beslut med dokumentering. Vi ser hur detta arbetssätt är nästintill ett krav för deras överlevnad till skillnad från större företag där TD tas på av andra anledningar.

Tabell 2: Sammanställning av analys

Varför	Klassificeringar	Typer	Aspekter	Utfall
Prioritering **Pragmatism** **Processer** **Attityd** **Kunskap**	**Strategisk** **Taktisk** **Inkrementell** **Oavsiktlig**	Krav Arkitektur **Design** **Kod** **Test** **Dokumentation** Infrastruktur Bygg **Defekt** Versionshantering	**Monetär kostnad** **Ränta och principal** **Leverage** Amnesty Konkurs **Utdrag och återbetalning**	**Moral** **Produktivitet** **Kvalite** **Risk**

5.2 Organisationens syn på technical debt

Vid avslutande diskussioner om TD såg vi att cheferna hade en positiv inställning till det. De ansåg att det var ett måste, speciellt när man är ett mindre företag. Det måste ske någon form av prioritering i arbetet. Det finns inte resurserna att arbeta på samma sätt större företag gör. Men vi såg att informant 2 som kom direkt från yrkeshögskolan hade en annan inställning till det. Hen ansåg inte att det borde tas bort men att det kommer att skapa framtida problem för företaget. Under analysen av resultatet spekulerades det till varför informant 2 hade denna åsikt. Vi kom fram till att antagligen under hens skoltid fanns det en prioritering på att när man lämnar in arbeten skall det vara kodade på ett korrekt och välskrivet sätt som följer ramverk. Vi tror att den här inställningen till att skriva perfekt kod har följt med från utbildning till arbetslivet men att denna åsikt kan komma att förändras med tiden och erfarenhet.

Jämför man informanternas svar med teorin presenterat i avsnitt 2.3 finns det många likheter. Att det finns generellt en positiv inställning till TD men att företagen måste förbättra hur de arbetar med det. De måste bli bättre på att hantera det för att inte riskera snöbollseffekten. Det är som McConnell (2007) och Tom, Aurum och Vidgen (2013) säger technical debt kommer alltid att finnas i företag, det som är viktigt nu är att lära sig hur man hanterar det. För med det finns otroliga möjligheter för företag. Denna nyfikenhet och drivet att lära sig var något vi upplevde under intervjun och efteråt.

5.3 Reflektioner kring studien och ramverket

Denna undersökning bygger sitt resultat på intervjuer hos ett mindre mjukvaruföretag. Genom att tolka informanternas svar har vi kunnat analysera hur de arbetar och dragit kopplingar till TD. Dock kan vi inte vara helt säkra på att deras svar på frågorna är egentligen hur de arbetar. Eftersom vi inte kan observera hur de faktiskt arbetar måste vi lita på att det de har sagt under intervjun är hur de arbetar. För att säkerställa att deras svar är legitima hade en längre studie behövts. Där det används både intervjuer och observationer. Med två datainsamlingsmetoder

går det att dels stärka intervjusvaren men potentiellt hitta nya mönster informanterna själva inte visste om. Dock med tidsramen för denna undersökning hade det varit nästintill omöjligt.

Vår studie har sin grund i det teoretiska ramverket presenterat i avsnitt 2 och överlag har den varit användbar. Dock har det påverkat hur vi bearbetar information eftersom vi granskar intervjusvaren utifrån ramverkets perspektiv. Detta gör att det har funnits en risk att svar från informanter kan bortses då de inte passar in på ramverket men i verklighet hade varit intressant att fördjupa oss i. Dock anser vi att de positiva delarna är större än de negativa då redan efter första intervjun sågs det direkta kopplingar mellan IT-företagets arbetssätt och TD. Med ramverket kunde vi på ett strukturerat sätt förklara grunderna till varför TD togs på, vilken form den tog och vad utfallet blev.

5.4 Förslag till fortsatt forskning

Det hade varit intressant att göra en longitudinell studie som observerar IT-företaget under en längre tid – med målet att studera hur TD påverkar företaget, vilka förändringar som sker och hur de hanterar det, samt att det hade tillfört med mer information i jämförelse med enbart intervjuer. Samtidigt hade det även varit spännande att se en komparativ studie med flera företag inblandade, både större och mindre företag – för att se vad som skiljer sig mellan deras sätt att hantera TD. Tyvärr fanns inte möjligheten att göra detta för oss på grund av den begränsade tiden – men hoppas att denna uppsats inspirerar framtida forskare att forska vidare inom detta område.

6 Slutsats

Innan studien genomfördes hade den syftet att bidra till forskningen kring hur TD kan appliceras på mjukvaruföretags arbetssätt. Framförallt mindre mjukvaruföretag vilket ledde till frågeställningen: "*Hur kan technical debt användas för att beskriva mindre mjukvaruföretags beslutsfattande och konsekvenserna därefter?"*.

Efter studiens genomförande har vi kommit fram till att även om inte företaget är medveten om TD liknar deras arbetssätt, motiveringar och resultat med hur TD ser ut idag. Det vi såg var en stark prioritering på att få ut produkten till användaren även om genvägar behövde tas för att nå målet. De här genvägarna skapade både möjligheter med ökad produktivitet men hade samtidigt negativa konsekvenser i form av försämrad kvalitet och moral inom företaget. Ibland var problemen så katastrofala att produkten inte fungerade.

Det vi har lyckats visa med denna undersökning är att det går att använda en anpassning av Tom, Aurum och Vidgens (2013) ramverk över TD tillsammans med andra författare och applicera det på ett mindre mjukvaruföretag. Ett företag där de inte arbetar med TD. Det visar att det går att beskriva mindre mjukvaruföretags beslutsfattande och konsekvenser med hjälp av TD. Dock är vår teori enbart beprövad på ett mindre företag. Fortsättningsvis hade en studie med flera olika företag för att stärka vår slutsats varit optimal.

7 Referenser

Bailey, J. (2008). First steps in qualitative data analysis: transcribing. *Family Practice*, 25(2), ss. 127-131. doi:10.1093/fampra/cmn003

Bell, J. (2014). Introduktion till forskningsmetodik. 5. uppl., Lund: Studentlitteratur AB.

Bohnet, J. & Döllner, J. (2011). Monitoring Code Quality and Development Activity by Software Maps. I *Proceedings of the 2nd Workshop on Managing Technical Debt*. Waikiki, Honolulu, Hawaii, USA 23 maj 2011, ss. 9-16. ACM.

Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K. & Zazworka, N. (2010). Managing Technical Debt in Software-Reliant Systems. I *Proceedings of the FSE/SDP workshop on Future of software engineering research*. Santa Fe, New Mexico, USA 7-8 november 2010, ss. 47-52. ACM.

Cambridge Dictionary (2018). Pragmatism. https://dictionary.cambridge.org/dictionary/english/pragmatism [2018-05-14]

Cunningham, W. (1992). The WyCash portfolio management system. I *Proceedings of the 7th Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'92)*. Vancouver, British Columbia, Canada 5-10 oktober, ss. 29-30.

Cunningham, W. (2009). Debt Metaphor [video], 14 februari. https://www.youtube.com/watch?v=pqeJFYwnkjE [2018-05-14]

Ernst, N. A. (2012). On the Role of Requirements in Understanding and Managing Technical Debt. I *Proceedings of the Third International Workshop on Managing Technical Debt*. ss. 61-64. IEEE Press.

Fowler, M. (2003). Technical Debt. *Martin Fowler* [blogg], 1 oktober. https://martinfowler.com/bliki/TechnicalDebt.html [2018-05-14]

Fowler, M. (2009). Technical Debt Quadrant. *Martin Fowler* [blogg], 14 oktober. https://martinfowler.com/bliki/TechnicalDebtQuadrant.html [2018-05-14]

Gartner. (2010). *Gartner Estimates Global 'IT Debt' to Be $500 Billion This Year, with Potential to Grow to $1 Trillion by 2025*. https://www.gartner.com/newsroom/id/1439513 [2018-05-14]

Giardino, C., Unterkalmsteiner, M., Paternoster, N., Gorschek, T. & Abrahamsson, P. (2014). What do we know about software development in startups?. *IEEE software*, 31(5), ss. 28-32.

Lakoff, G. & Johnson, M. (1983). Metaphors We Live By: University of Chicago Press.

Li, Z., Avgeriou, P. & Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101, ss. 193-220

McConnell, S. (2007). Technical Debt. *Construx* [blogg], 1 november. http://www.construx.com/10x_Software_Development/Technical_Debt/ [2018-05-14]

OnTechnicalDebt. (2012). Steve McConnell: How to Categorize & Communicate Technical Debt. *OnTechnicalDebt* [blogg], 30 juli. http://www.ontechnicaldebt.com/blog/steve-mcconnell-on-categorizing-managing-technical-debt/ [2018-05-14]

Patel, R. & Davidson, B. (2011). Forskningsmetodikens grunder. 4:8. uppl., Lund: Studentlitteratur AB.

Tom, E., Aurum, A. & Vidgen, R. (2013). An exploration of technical debt. *Journal of Systems and Software*, 86(6), ss. 1498-1516.

Zazworka, N., Shaw, M. A., Shull, F. & Seaman, C. (2011). Investigating the Impact of Design Debt on Software Quality. I *Proceedings of the 2nd Workshop on Managing Technical Debt*. Waikiki, Honolulu, Hawaii, USA 23 maj 2011, ss. 17-23. ACM.

8 Bilagor

Bilaga 1: Intervjumall

1. Vad heter du? Vad har du för roll här på företaget?
2. Skulle du kunna beskriva din bakgrund?
3. Kan du beskriva en typisk arbetsdag för dig?
4. Kan du beskriva organisationens arbetsprocess när ni utvecklade den senaste funktionen. Från start till slut.
    a. *Exempel på följdfrågor beroende på svar.*
    b. Hur arbetar ni med kodstandarder? Kan du ge exempel på några ramverk ni använder?
    c. Hur ser er dokumenteringsprocess ut?
    d. Kan du beskriva er testningsprocess?
5. Har ni någonsin valt att bortprioritera en process av strategiska anledningar?
    a. Vilken process(er) valde ni bort?
    b. Vilka motiveringar fanns till att göra det?
    c. Hur upplevde ni resultatet av att göra det både kort och långsiktigt? Var de positiva eller negativa? Kan du ge exempel?
        i. Vad var din åsikt till det här beslutet?
        ii. Upplevde du några förändringar i er produktivitet?
        iii. Vad blev påverkan på produktens kvalite?
        iv. Såg ni några förändringar i riskerna hos projektet?
        v. Har ni behövt gå tillbaka och underhålla den funktionen?
6. Kan du beskriva ert sätt att behandla och fixa äldre problem i systemet?
7. Har du hört talas om technical debt innan? Om nej förklara kort vad det är.
    a. Är det något du har sett att ni arbetar med? Kan du ge exempel?
8. Något mer du skulle vilja tillägga som skulle kunna bidra till studien?

Bilaga 2: Inspelningsmedgivande

Inspelningsmedgivande

Som informant i vår undersökning kommer du att behandlas anonymt i den här studien.

Jag förstår att min intervju kommer att spelas in och ger tillåtelse att mina svar får användas av Jonathan Philipp och Kristian Gocko i deras studie för deras kandidatuppsats vid Göteborg universitet VT 2018.

Signatur:_____

Namnförtydligande:_____

Datum:_____