CHALMERS

# Utvärdering av KPI:er för agil utveckling

En fallstudie av Delphi

# Evaluation of KPIs for agile development

A case study of Delphi

Kandidatarbete i Industriell ekonomi

ERIK ABRAHAMSSON
ANDREAS ECKHOFF
JACOB HOLMÉN
MARKO IVANOVIC
JOHAN SELIN
EDVIN TEGBRANT

Institutionen för Teknikens ekonomi och organisation
*Avdelningen för Entrepreneurship and Strategy*
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016
Kandidatarbete TEKX04-16-05

**Förord**

Rapporten utgör resultatet av ett kandidatarbete som utförts på halvtid under våren 2016. Studien genomfördes på avdelningen Entrepreneurship and Strategy vid institutionen *Teknikens ekonomi och organisation* på Chalmers tekniska högskola, Göteborg. Handledare Marouane Bousfiha, som bidragit mycket under arbetsprocessen med stöd och vägledning, förtjänar ett stort tack.

Kandidatuppsatsen framkom tack vare uppdragsgivaren Delphi. Kandidatgruppen vill tacka Marcus Hedberg för inledande diskussioner, Thomas Frenning och Ross Fitkin för handledning samt Matti Larborn och Hans Lindau för givande möten. Därtill ägnas ett stort tack till de anställda på Delphi och Volvo Cars som ställt upp på intervjuer.

Göteborg 2015-05-19

Erik Abrahamsson
Andreas Eckhoff
Jacob Holmén
Marko Ivanovic
Johan Selin
Edvin Tegbrant

## Sammandrag

### Problem
Mjukvara har traditionellt sett utvecklats enligt vattenfallsmodellen. Då digitaliseringen ställer krav på flexibilitet, när nya produkter och tjänster ska tas fram och spridas till allmänheten, är den traditionella vattenfallsmodellen inte tillräcklig längre. Anledningen är att projekt som genomförs under lång tid, i vilka respons från kunden fås i ett mycket sent skede, riskerar att misslyckas med att uppnå kundkraven och bli kostsamma för de inblandade parterna. Som lösning på dessa problem har agila utvecklingsmetoder växt fram. En vanligt förekommande variant är Scrum.

Rapporten behandlar agil projektstyrning på företaget Delphi, en global leverantör av elektroniska och tekniska lösningar till fordonsindustrin. Delphi har nyligen startat upp ett pilotprojekt inom agil mjukvaruutveckling i samarbete med Volvo Cars, vilket kommer att sträcka sig över flera år. Full agil utveckling är mindre vanligt på Delphi för projekt av den här storleken, varför det finns en osäkerhet kring hur projektet bör utvärderas och styras på ett önskvärt sätt.

### Syfte
Syftet med studien är att kartlägga Delphis styrning av projekt inom agil utveckling, med avseende på KPI:er och dess effekter på projektstyrning, samt lämna rekommendationer utifrån rådande forskning.

### Teoretiskt ramverk
Det teoretiska ramverket lägger en grund för den vidare studien med inledande definitioner, tillsammans med djupgående beskrivningar av vanligt förekommande KPI:er inom mjukvaru-utvecklingsprojekt. Både användnings- och problemområden redogörs för, i syfte att kunna jämföras med Delphis styrning.

### Metod
För att uppnå studiens syfte har både ett teoretiskt ramverk utformats, i form av en litteraturstudie, samt empirisk data inhämtats, med hjälp av intervjuer. Litteraturstudien har legat till grund för analysen och diskussionen av inhämtad empirisk data. Utifrån dessa har rekommendationer formulerats. Arbetsgången har utvärderats med avseende på validitet och reliabilitet i en metoddiskussion, i vilken exempelvis intervjuformen samt antalet intervjuer diskuteras.

### Resultat och implikationer
En analys av litteraturen och empirin identifierade både likheter och skillnader mellan Delphis arbetssätt och vad litteraturen förespråkar. Analysen visade på brister inom områden såsom KPI:ers koppling till övergripande företagsmål, uppföljning av kundvärde, mätning av produktkvalitet och säkerställning av medarbetarnöjdhet. För att möta de identifierade problemen föreslås förbättringsåtgärder. Genom att använda sig av en tydlig modell för framtagning av KPI:er kan bristerna relaterade till anpassning mellan KPI:er och företagsmål undvikas. Specifika KPI:er som kan införas för en förbättrad uppföljning av kundvärde är Earned business value och Net promoter score. Även KPI:n Happiness kan införas för att säkerställa en hög medarbetarnöjdhet. Technical debt bör dessutom kvantifieras för att kunna bibehålla en hög kvalitet. Slutligen rekommenderas utbildning inom agila processer då det är en central del av agil utveckling, vilket i sin tur är avgörande för att Delphi ska kunna möta framtidens krav.

# Abstract

**Problem**
Software has traditionally been developed according to the waterfall model. As the digitalization creates a demand for flexibility in the development and dispersion of new products and services, the traditional waterfall model is no longer sufficient. The reason is that long term development projects, in which customer feedback is received at a late stage, carries the risk of failure and becoming costly for the involved parties. As a reaction to these problems, agile development methods have emerged. A common variety is Scrum.

The report addresses agile project management at the company Delphi, a global supplier of electronic and technical solutions to the automotive industry. They have recently started a pilot project within agile software development in cooperation with Volvo Cars, stretching over several years. Full agile development is unusual at Delphi within projects of this size, which is why there is an uncertainty concerning how the project should be evaluated and governed in a desirable way.

**Aim**
The aim of the study is to map out Delphi's management of projects within agile development, concerning KPIs and their effects on project management, and provide recommendations based on existing research.

**Theoretical framework**
The theoretical framework provides a basis for the continued study with initial definitions along with in-depth descriptions of regularly occurring KPIs within agile software development. Both the area of use and the associated problems are outlined, for the purpose of comparison with Delphi's management of projects.

**Method**
To achieve the aim of the study a theoretical framework has been defined, consisting of a literature study, and empirical data has been collected through interviews. The literature study has acted as a foundation for the analysis and discussion of collected empirical data. Based on this, recommendations have been formulated. The work process has been evaluated with regards to validity and reliability in a method discussion where, for example, the interview form and the number of interviews are discussed.

**Results and implications**
An analysis of the literature and the empirical data identified both similarities and differences between Delphi's way of working and what the literature advocates. The analysis revealed flaws within areas such as connection between KPIs and company goals, follow-up on customer value, measurement of product quality and assurance of employee satisfaction. In order to solve the identified problems, improvement measures are suggested. By using a clear model for deciding which KPIs to use, the flaws concerning the alignment between KPIs and company goals can be avoided. Specific KPIs that can be introduced for an improved follow-up of customer value are Earned Business Value and Net Promoter Score. Furthermore, the KPI Happiness can be introduced to ensure a high level of employee satisfaction. Technical debt should also be quantified, in order to be able to maintain a high quality. Finally, education about agile processes is recommended. It is needed in order to secure the success of the current pilot project and is also a prerequisite for the success of future projects.

# Innehållsförteckning

## Figurförteckning



## Tabellförteckning

## Ordlista

**Agil utveckling** - Ett samlingsnamn för iterativa och inkrementella systemutvecklingsmetoder.
**Daily stand-up meeting** - Ett kort, oftast endast 15 minuter långt, dagligt möte under vilket utvecklingsteamet ska diskutera vad de har åstadkommit, vad de ska göra vidare samt vilka hinder de ser.
**Epic** - Ett mer övergripande arbetspaket som kan bestå av ett flertal user stories.
**Feature** - Funktionalitet i en produkt.
**GQM** - Goal-Question-Metric, vilket är en målbaserad modell för framtagning av KPI:er.
**IHU** - Infotainment Head Unit, vilket syftar på huvudenheten för människans interaktion med mjukvara i bilen.
**JIRA** - Ett verktyg för uppföljning och inrapportering om hur utvecklingsprojekt fortlöper. Genom verktyget kan till exempel grafer över KPI:er visas.
**KPI** - Key Performance Indicator är kvantifierbara mätetal som företag använder sig av för att jämföra hur de presterar i förhållande till strategiska och operationella mål
**Lean** - En filosofi om hur man hanterar resurser med syftet att eliminera allt som inte tillför värde.
**Product backlog** - En lista över user stories, ordnade efter prioritet, som innefattar allt som behöver göras i projektet. Listan sammanställs av produktägare och kan uppdateras kontinuerligt.
**Produktägare** - Person som representerar kundens intresse och är ansvarig för att product backlog är uppdaterad och korrekt prioriterad.
**PSM** - Practical Software & System Measurement, vilket är en modell som baseras på bästa praxis inom mjukvaruutveckling. Modellen används vid framtagning av KPI:er.
**Scrum** - Ett ramverk för agil mjukvaruutveckling.
**Scrum master** - Person i varje team som är ansvarig för att Scrum-konceptet följs. Rollen innebär även ansvar för kommunikation med andra team.
**Sprint** - En iterationscykel med förutbestämd varaktighet under vilken ett team tar på sig att färdigställa på förhand valda user stories.
**Sprint backlog** - User stories hämtade ur product backlog inför varje sprint som ska färdigställas.
**Story point** - Ett mått för att estimera hur mycket arbete som krävs för att färdigställa en user story eller epic. Används i estimeringssyfte.
**User story** - Ett behov från en tänkt användare som sammanfattats i ett fåtal meningar. Beskrivningen omfattar frågorna vem är användare, vad är behovet, och varför finns behovet?

# 1. Inledning

Rapporten är en studie som utvärderar företaget Delphis styrning med KPI:er inom ett agilt mjukvaruutvecklingsprojekt. Inledningen innefattar bakgrund, syfte, frågeställningar, avgränsningar och disposition.

## 1.1 Bakgrund

Teknikens utveckling går allt snabbare. Alla människors arbete och vardag påverkas. Små barn lär sig skolämnen på läsplattor och det kommer inte att dröja länge förrän befolkningen transporteras av självkörande bilar. Såväl konsumenters användarbeteende som leverantörers arbetsprocesser utvecklas i hög takt. Digitaliseringen ställer krav på hög flexibilitet när nya produkter och tjänster ska tas fram och spridas till allmänheten. Utvecklingen ställer även stora krav på hållbarhet som har blivit en kritisk del i företagens styrning och i dagens samhälle.

Mjukvara till tekniska lösningar har traditionellt sett utvecklats enligt vattenfallsmodell. Det är en sekventiell process i vilken de ingående faserna, såsom kravspecifikation, konstruktion och test, sker i ett enkelriktat flöde. Med tanke på dagens höga förändringstakt av teknik har vattenfallsmodellen visats sig ha begränsningar (Martano & Sedehi 2012, s. 99). Projekt som genomförs under lång tid, i vilka respons från kunden fås i ett mycket sent skede, riskerar att misslyckas med att möta kundkraven och bli oerhört kostsamma för de inblandade parterna. Stora IT-utvecklingsprojekt är kända för att inte kunna uppfylla projektplanerna (Magnusson & Nilsson 2014, s. 95). Som reaktion på detta arbetssätt har nya principer börjat tillämpas inom mjukvaruutvecklingen.

Agil utveckling är ett samlingsnamn som innefattar ett flertal metoder, i vilka fokus ligger på iteration och inkrementella framsteg. Samarbete och kommunikation med kunden framhävs som nyckelfaktorer för lyckade projekt. På så sätt uppnås en högre flexibilitet och nya, eller förändrade, krav som uppstår under tiden kan på ett lättare sätt fångas upp. Ett exempel på en specifik metodik inom agil utveckling är Scrum, som formaliserades 1995 och har vuxit kraftigt därefter. Kort beskrivet arbetas en prioriterad lista fram med arbetspaket av produkten. Dessa betas av del för del, uppdelat på tidsperioder om mellan en till fyra veckor, kallade sprintar. Efter varje sprint är det tänkt att en testbar del av slutprodukten ska vara utvecklad. Sprintar genomförs om och om igen tills kundens krav är uppfyllda. Därav Scrums beskrivning som inkrementell och iterativ. Fokus är att slutligen kunna leverera så hög affärsnytta som möjligt till kunden (Sutherland 2014, s. 234-238).

Med tanke på att uppdragen, som leverantörer av avancerade tekniska lösningar tar på sig, ofta spänner över flera år och involverar många intressenter är det av stor vikt att kunna följa och rapportera framstegen. Statusmätningen inom traditionell utveckling, med på förhand specificerade faser, skiljer sig från exempelvis Scrum inom agil utveckling. Det iterativa arbetssättet med hög förändringstakt riskerar att skapa frustration och överflödig dokumentation om Key Performance Indicators (hädanefter KPI:er) hanteras på samma sätt som de traditionellt gjorts. Nya, anpassade KPI:er används därför i syfte att uppfylla intressenternas krav samtidigt som bördan på utvecklarna hålls ned (Jeeva Padmini, Dilum Bandara & Perera 2015, s. 312). Som alltid när projekt, processer eller resurser mäts är det viktigt att se vilka effekter som mätningen får. Styrs verksamheten på fel sätt riskerar suboptimeringar att uppstå.

För att belysa den problematik som uppstår vid användning av KPI:er inom agil utveckling kommer denna studie att genomföra ett praktikfall, avseende en undersökning av ett pågående projekt på företaget Delphi. Delphi är en global leverantör av elektroniska och tekniska lösningar till fordonsindustrin. Närmare 170 000 anställda arbetar från 33 olika länder. Tre ledord genomsyrar hela verksamheten: säkerhet, miljövänlighet och uppkoppling (Delphi 2015, s. 10). I Sverige representeras Delphi av två kontor i Göteborgsregionen. Ett av dessa, beläget i Göteborgs stad, arbetar endast med mjukvara och kommer att ligga till grund för studien.

Ett omfattande projekt, kallat Infotainment Head Unit (hädanefter IHU), har nyligen startats upp i samarbete med Volvo Cars som kommer att sträcka sig över flera år. Från Delphis sida arbetar i huvudsak personal från avdelningen Infotainment and Driver Interface, under divisionen Electronics & Safety, med projektet. Slutprodukten är ett avancerat infotainment-system med hög användarupplevelse. Utvecklingen av mjukvaran genomförs i enlighet med Scrum-metodiken, uppdelat på fler än 20 globala team med medlemmar från Sverige, Tyskland, Polen och Indien. Full agil utveckling är mindre vanligt på Delphi för projekt av den här storleken. IHU-projektet bedrivs därför som en pilotstudie som ett steg i den globala spridningen av agil utveckling. Av den anledningen är det av intresse att utvärdera styrningen inför kommande projekt. Ledning, medarbetare och kunden har alla olika förhållningssätt till projektets framsteg och styrning. För att skapa en förståelse för de befintliga KPI:ernas effekter på styrningen av projektet krävs alltså att hänsyn tas till alla parter i studien. En aspekt som dock begränsar agil utveckling är att IHU är ett fastprisprojekt, det vill säga all den funktionalitet som Delphi och Volvo Cars kommit överens om från upphandlingen ska levereras till ett på förhand bestämt pris.

Studien har ett antal intressenter som påverkas olika av resultatet. Delphis intresse ligger inte i att erhålla en lösning på ett specifikt problem. Företaget är istället ute efter en insikt i, och utvärdering av, deras KPI:er samt hur dessa påverkar styrningen av det aktuella projektet. Kunden, i det här fallet Volvo Cars, har också förmedlat ett intresse för studien, men är inte den högst prioriterade mottagaren av resultatet. Det primära syftet är alltså att förse Delphi med efterfrågad information. Eftersom flera av KPI:erna ändå tillhör standarden inom mjukvaru-utveckling med Scrum finns det ett värde även för andra aktörer att ta till sig av resultatet. Därtill väntas studien bidra till den rådande forskningen gällande mätningar inom agil mjukvaruutveckling. Området har tidigare studerats, men det finns ett behov av ytterligare forskning och empiriska studier för att öka förståelsen (Gustafsson 2011, s. 21; Kupiainen, Mäntylä & Itkonen 2014, s. 27). En av grundprinciperna för agil utveckling betonar att hållbar utveckling bör främjas (The Agile Alliance 2011b). Därför kommer detta ämnesområde att integreras i rapporten, med fokus på ekonomisk och social hållbarhet.

## 1.2 Syfte

Syftet med studien är att kartlägga Delphis styrning av ett projekt inom agil utveckling, med avseende på KPI:er och dess effekter på projektstyrning, samt lämna rekommendationer utifrån rådande forskning.

## 1.3 Frågeställningar

För att kunna uppnå studiens syfte kommer de fyra nedanstående frågeställningarna att undersökas.

***1. Hur föreslår litteraturen att företag bör styra agila mjukvaruutvecklingsprojekt med KPI:er?***
Hela studien kommer att utföras som en fallstudie på Delphi, men det finns många andra företag i liknande situationer. Genom att dra lärdom av framgångar och misstag som tidigare begåtts av andra kan agil projektstyrning förbättras. För att kunna utvärdera Delphis KPI:er krävs alltså en genomgång av litteratur, speciellt om det finns några mönster i hur mer och mindre framgångsrika agila projekt mäts. Frågan besvaras i kapitel 2, Teoretiskt ramverk.

***2. Hur styr Delphi det aktuella agila mjukvaruutvecklingsprojektet med KPI:er?***
För att få en förståelse kring hur Delphi styr utefter sina KPI:er krävs en kartläggning av dessa. Det som undersöks är bland annat definition samt användnings- och problemområden. Projektet är i sin uppstartsfas, varför en dokumenterad sammanställning inte finns i dagsläget och motiverar att tid läggs på en utredning av den nuvarande situationen. Frågeställningen är också av intresse att undersöka för att kunna finna suboptimeringar och skillnader mellan ledningens och utvecklarnas syn på mätningarna. Detta utreds i kapitel 4, Empiriska resultat.

***3. Hur förhåller sig Delphis projektstyrning med KPI:er till uppfattningar i litteraturen och vad får det för konsekvenser?***
Frågan avser att undersöka hur Delphis styrning med KPI:er står sig gentemot vad litteraturen förespråkar. Vad som anses vara sant i ett generellt fall enligt litteraturen behöver inte stämma i Delphis specifika fall, men att vara öppen för hur andra företag framgångsrikt styrt agila projekt är viktigt. Eventuella avvikelser identifieras och konsekvenser diskuteras. Svaret på frågeställningen presenteras i kapitel 5, Analys och diskussion.

***4. Hur bör Delphi styra sina agila mjukvaruutvecklingsprojekt med KPI:er?***
Med utgångspunkt i svaret på frågeställning tre kommer slutligen rekommendationer att lämnas angående den fortsatta användningen av KPI:er. Dessa är framtagna för Delphis specifika fall och det aktuella projektet. Rekommendationerna ämnar också ligga till grund för framtida styrning inom agila projekt på företaget. Förslagen lämnas i kapitel 6, Slutsatser och rekommendationer.

## 1.4 Avgränsningar

Studien genomfördes under en begränsad tid. Ett antal avgränsningar gjordes i syfte att minska omfattningen och möjliggöra en djupgående analys. Detta kapitel kommer att presentera och förklara de avgränsningar som har gjorts.

Delphi är verksamma över hela världen, men studien avgränsades till IHU-projektet. Hundratals anställda från olika länder arbetar på projektet, dock fokuserades datainsamlingen och analysen till kontoret i Göteborgs stad.

Då studien behandlar Scrum, vilket är framtaget för mjukvaruutveckling, följde en naturlig avgränsning i att inte undersöka KPI:er inom hårdvaruutveckling. Scrum är en erkänd och välanvänd process för mjukvaruutveckling, varför nackdelar och eventuella alternativ inte behandlas. Resultatet kommer istället att visa på hur Scrum-projekt av denna stora storlek styrs effektivt med hjälp av KPI:er.

KPI:er går att kategorisera i flera olika nivåer och användningsområden. Studien avgränsades från de mer tekniska detaljerna och inriktades snarare på övergripande organisationsaspekter. Därför var KPI:er för produktprestanda, så som reaktionstid och minnesutnyttjande, inte i åtanke under studien. Motiveringen är grundad i att studien har bedrivits vid institutionen *Teknikens ekonomi och organisation*, vars expertis ligger inom verksamhetsledning snarare än mjukvaruoptimering.

Även inom övergripande, organisationsinriktade KPI:er har ett urval gjorts för att möjliggöra en mer djupgående analys, snarare än en överblick av samtliga KPI:er. Denna sållning har genomförts med avseende på relevans för Delphis projekt samt hur vanligt förekommande KPI:erna har varit i den litteratur som undersökts.

På grund av mjukvaruutvecklingens virtuella natur finns det ingen stark koppling till ekologisk hållbarhet. Därmed är det ett område utanför denna rapports omfång och fokus ligger istället på att lyfta fram hållbarhetsperspektivets kopplingar till agil mjukvaruutveckling, vilka främst hör till områdena ekonomisk och social hållbarhet.

## 1.5 Disposition

Rapportens disposition är strukturerad i sex kapitel. Hållbar utveckling är behandlat på ett integrerat sätt genom alla rapportens kapitel. Kapitel 2, Teoretiskt ramverk, inleder med en genomgång av litteratur för att svara på studiens första frågeställning, och ligger till grund för rapportens senare delar. Därefter följer kapitel 3, Metod, som innefattar en beskrivning av hur datainsamlingen genomförts. Därpå presenteras resultatet av de genomförda intervjuerna i kapitel 4, Empiriska resultat. På så sätt besvaras studiens andra frågeställning. Svar till den tredje frågeställningen följer i kapitel 5, Analys och diskussion, vari skillnader och likheter sammanställs mellan det teoretiska ramverket och det empiriska resultatet. Dessutom inkluderas ett resonemang om hur avvikelser påverkar Delphis arbetssätt. I kapitel 6, Slutsatser och rekommendationer, rundas rapporten av genom att presentera förslag för den fortsatta styrningen. Därmed besvaras också den fjärde och avslutande frågeställningen för studien.

# 2. Teoretiskt ramverk

Under denna rubrik presenteras den litteraturstudie som gjorts. Syftet med det teoretiska ramverket är att skapa en översikt inför det kommande analys- och diskussionskapitlet, vari litteraturen jämförs med det empiriska resultat som samlats in genom intervjuer. Den första frågeställningen i rapporten kommer att besvaras:

***1. Hur föreslår litteraturen att företag bör styra agila mjukvaruutvecklingsprojekt med KPI:er?***

Kapitlet är indelat i följande rubriker:

- ***2.1 Hållbarhet:*** Ämnesområdet hållbar utveckling presenteras först, eftersom det kommer att behandlas som en integrerad del i rapportens efterföljande delar.
- ***2.2 Mjukvarurelaterad projektstyrning:*** Det övergripande ämnesområdet projektstyrning presenteras. Här tas definitionen av projektstyrning upp, skillnader mellan vanlig och mjukvarurelaterad projektstyrning samt två konkreta modeller för mjukvaruutveckling.
- ***2.3 KPI:er:*** Begreppet KPI:er introduceras tillsammans med användningsområden. Dessutom nämns hur KPI:er bör behandlas i en agil kontext.
- ***2.4 Modeller för framtagning av KPI:er:*** För att mäta rätt saker inom projektstyrning, som är i linje med företagsmålen, är det fördelaktigt med KPI:er som är framtagna i samband med en modell. Här presenteras två sådana modeller, GQM och PSM.
- ***2.5 Centrala områden och KPI:er för mjukvaruutveckling:*** De KPI:er som rapporten behandlar presenteras under rubrikerna planering, kvalitet och kundvärde, vilket är de övergripande mål som KPI:erna ämnar att uppnå. I slutet av kapitlet sammanfattas informationen i tabell 1.

## 2.1 Hållbarhet

En vedertagen definition av hållbar utveckling lyder ”Sustainable development is a development that meets the needs of the present without compromising the ability of future generations to meet their own needs” (United Nations 1987, s. 16). Dessa behov kan sedan delas in i de tre dimensionerna ekologisk, social och ekonomisk hållbarhet (Murugesan & Gangadharan 2012, s. 64).

Hållbarhet inom mjukvaruutveckling innebär att mjukvaran är framtagen med hänsyn till dess positiva och negativa påverkan på miljön, samhället och ekonomin, samtidigt som det värde för vilken mjukvaran är ämnad levereras (Murugesan & Gangadharan 2012, s. 65). Hållbar mjukvaruutveckling upplever i dagsläget en avsaknad av väletablerade standarder och praxis för att effektivt hantera hållbarhet i de tre dimensionerna. Det största fokuset läggs endast på underhåll av äldre system. Det är nödvändigt att denna hantering genomgår en förändring, i vilken alla tre hållbarhetsdimensioner uppmärksammas med en helhetssyn och anpassas till affärsmålen för olika mjukvaruprojekt (Murugesan & Gangadharan 2012, s. 64).

Social hållbarhet kan vara svår att definiera, men det är en allmän uppfattning att hållbarhet i detta avseende rör samhället och individen. Faktorer som kan medverka till social hållbarhet är bland annat jämställdhet, hälsa och livskvalitet samt delaktighet och inflytande (Folkhälsomyndigheten 2014).

Exempel på hur mjukvara påverkar omgivningen i den sociala dimensionen är hur bristen på anpassningar för människor med funktionsnedsättning exkluderar dem från arbetskraften. Vidare kräver dåligt designad och svåranvänd mjukvara mer träning och användarsupport, vilket i sin tur kan leda till ett ökat utsläpp av växthusgaser, i synnerhet om det innebär utskrifter och distribution av träningsmaterial och ytterligare resor (Murugesan & Gangadharan 2012, s. 64).

Ekonomisk hållbarhet innebär självbärande ekonomisk tillväxt och utveckling, vilket kan benämnas "the business of staying in business". De affärsstrategier som används för att uppnå detta länkas till koncepten effektivitet och ändamålsenlighet. Ekonomisk hållbarhet kan sägas vila på vinster, avkastning till intressenterna och lyckade investeringar som garanterar överlevnaden av företaget. Detta koncept kan definieras som "meeting the needs of a firm's direct and indirect stakeholders without compromising its ability to meet the needs of future stakeholders as well" (Murphy & Lang 2014, s. 7).

## 2.2 Mjukvarurelaterad projektstyrning

Mjukvarurelaterad projektstyrning är en underkategori till det mer övergripande ämnet projektstyrning. För att kunna svara på vad mjukvarurelaterad projektstyrning är behöver först definitionen av ett projekt och projektstyrning beröras.

### 2.2.1 Definition av ett projekt

Ett projekt är en tillfällig ansträngning som görs för att skapa en unik produkt, en unik tjänst eller ge ett unikt resultat. Projektet är temporärt eftersom det oftast finns ett förutbestämt start- och slutdatum och eftersom det är begränsat i innehåll och resursanspråk. Det är unikt eftersom det inte är en rutin för företaget, utan är något som skapats för att uppnå ett särskilt mål (Project Management Institute 2016).

Ett projekt delas ofta in i fem olika faser (Project Insight 2016):

1. ***Konceptformulering och initiering:*** En idé för ett projekt skapas och undersöks. Här behöver beslutsfattare klargöra om idén kommer att vara av värde för företaget samt om det är möjligt att genomföra den.
2. ***Definiering och planering:*** En projektplan skapas, vilken besvarar vilka resurser som kommer att krävas, hur stort omfånget ska vara, hur lång tid det kommer att ta och vad det kommer att kosta.
3. ***Genomförande:*** Team skapas och tilldelas arbetsuppgifter.
4. ***Övervakning och kontroll:*** De som leder projektet jämför den aktuella statusen med den planerade statusen och försöker se till att de ligger i fas.
5. ***Stängning:*** När projektet är avslutat och accepterat av alla intressenter bör en utvärdering av projektet ske, i vilken företaget undersöker vad som gick bra och vad som kunde ha gjorts bättre.

### 2.2.2 Definition av projektstyrning

Projektstyrningen är viktigast under övervaknings- och kontrollfasen, men genomsyrar samtliga faser. Enligt Project Management Institute innebär projektstyrning användningen av kunskap, färdigheter, verktyg och tekniker inom flera olika aktiviteter för att uppnå målen med ett specifikt projekt (Project Insight 2016). En annan definition som styrker ovanstående är "Project Management is the discipline of defining and achieving targets while optimizing the use of resources (time, money, people, materials, energy, space, etc) over the course of a project (a set of activities of finite duration)." (Terry 2016).

### 2.2.3 Nyttor med projektstyrning

Att identifiera intressenters mål och se till att de uppfylls är de grundläggande kraven för att ett projekt ska kunna bli framgångsrikt. Detta är målet inom det mer övergripande ämnet projektstyrning och även inom mjukvarurelaterad projektstyrning (Hughes & Cotterell 2009, s. 1). Utan en bra styrning riskerar projekten att bli försenade, överskrida budget och misslyckas med avseende på innehåll. Vid en undersökning i USA år 2003, som omfattade 13 522 IT-projekt, konstaterades att hela 82 % av projekten misslyckades med avseende på tid och att 43 % misslyckades med avseende på budget. Orsaken till dessa misslyckanden går ofta att härleda till styrningen (Hughes & Cotterell 2009, s. 2).

### 2.2.4 Skillnader mellan generell och mjukvarurelaterad projektstyrning

Det finns många likheter mellan generell projektstyrning och mjukvarurelaterad projektstyrning, exempelvis att de innehåller i stort sett samma faser (Hughes & Cotterell 2009, s. 5), men det finns också ett antal viktiga skillnader. Fred Brooks, författaren av boken The Mythical Man-Month: Essays on Software Engineering, har identifierat fyra huvudsakliga skillnader (Hughes & Cotterell 2009, s. 3-4):

1. ***Osynlighet:*** Framsteg inom ett mjukvarurelaterat projekt är inte alltid direkt synliga, vilket de ofta är inom vanlig projektstyrning. Ett viktigt mål med mjukvarurelaterad projektstyrning är därför att göra det osynliga synligt.
2. ***Komplexitet:*** En mjukvarurelaterad produkt innehåller högre komplexitet per spenderad krona än andra produkter.
3. ***Anpassning:*** Mjukvaruutvecklare måste anpassa sig mer efter de krav som kunder ställer. Problemet är inte bara att individer kan vara inkonsekventa, utan stora problem uppstår också när det finns bortfall inom intern kommunikation och kollektivt minne.
4. ***Flexibilitet:*** Mjukvara är enklare att förändra än fysiska produkter, vilket ses som en styrka. Samtidigt medför det högre ställda krav på att mjukvaran alltid är anpassningsbar.

### 2.2.5 Från vattenfallsmodellen till agil utveckling

Nedan kommer konkreta utvecklingsmodeller för mjukvara att presenteras. Först presenteras den mer traditionella vattenfallsmodellen och därefter agil utveckling. I dagens komplexa samhälle har det visat sig att agila projekt ger positiva resultat och förekommer därför allt oftare (Martano & Sedehi 2012, s. 1). Sist kommer Scrum att tas upp, vilket är den agila modellen som Delphi använder sig av för det studerade projektet.

**Vattenfallsmodellen**
Vattenfallsmodellen är en enkelriktad process för mjukvaruutvecklingsprojekt, vilken definierades år 1970 av Winston W. Royce och innehåller fem olika steg (Hughey 2009):

1. ***Kravspecifikation:*** Krav som kunden har för produkten presenteras och förhandlas.
2. ***Design:*** En design utarbetas som uppfyller de kundkrav som inhämtats i kravspecifikationen.
3. ***Implementering:*** Team får arbetsuppgifter för att tillsammans sätta ihop ett system som har den givna designen.
4. ***Verifiering:*** Kunden får möjlighet att avgöra om systemet uppnår kraven eller ej.
5. ***Underhåll:*** Defekter som undgick utvecklingsteamet åtgärdas och design kan förändras för att bättre passa kundkraven.

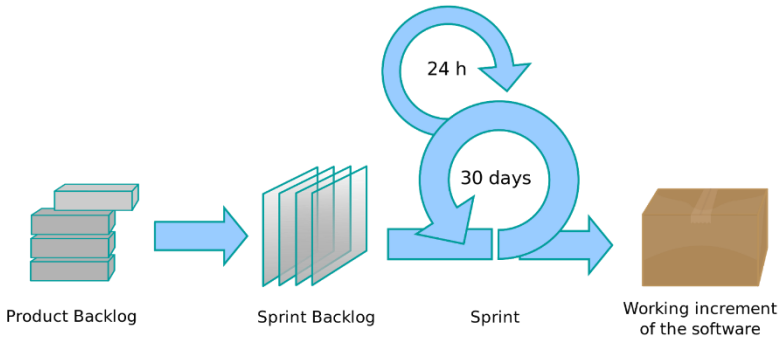
Vattenfallsmodellen har många likheter med den modell som presenterades för generell projektstyrning i kapitel 2.1. Utvecklingsmetoden tar inte hänsyn till att mjukvarurelaterade projekt är unika i många avseenden, bland annat med avseende på komplexitet, anpassning och flexibilitet. Detta gör att vattenfallsmodellen blir otillräcklig och får flera brister. Ett stort problem är att kunden sällan vet vad den faktiskt vill ha på en sådan abstrakt nivå som i kravspecificeringsstadiet. När kunden väl får se produkten och förstår vad den vill ha är det svårt att genomföra förändringar (Hughes & Cotterell 2009, s. 82-83).

**Agil utveckling**
Agil utveckling är ett samlingsnamn för de metoder som förespråkar att produkten utvecklas i flera iterationer. Metoderna utgår från en cyklisk modell, till skillnad från vattenfallsmodellen. 2001 släpptes The Manifesto for Agile Software Development när flera forskare inom programvaruutveckling samlades för att utvärdera de metoder som användes för mjukvaruutveckling (Hughes & Cotterell 2009, s. 92-93). De kom fram till fyra prioriteringar som bör hållas i åtanke vid mjukvaruutveckling (The Agile Alliance 2001a):

- Individer och interaktioner framför processer och verktyg
- Fungerande programvara framför omfattande dokumentation
- Kundsamarbete framför kontraktsförhandlingar
- Anpassning till förändring framför att följa en plan

**Scrum**
Det finns flera konkreta varianter av agil utveckling, såsom Extreme programming, Feature-driven development och Scrum (Hughes & Cotterell 2009, s. 92-93). Scrum är ett agilt ramverk för mjukvaruutveckling och går ut på att en produkt utvecklas kontinuerligt i cykler, kallade sprintar. Inom Scrum delas utvecklarna in i flera team, vanligtvis bestående av fem till tio personer. Det finns tre olika roller inom varje team, vilka är utvecklare, produktägare och Scrum master (Scrum Alliance 2016).

I en så kallad product backlog prioriteras arbetsuppgifterna, user stories, av produktägaren. User stories är vanligtvis kvantifierade i story points, utifrån deras komplexitet. Inför varje sprint tar teamet på sig user stories från toppen av product backlog. Dessa överförs till det som kallas för sprint backlog och arbetsbördan ska motsvara så många story points som de tror sig kunna avverka under en sprint. En sprint varar vanligtvis mellan två och fyra veckor. Varje dag under sprinten ska en daily stand-up meeting genomföras, vilket är en kort planering för det kommande dygnet samt en genomgång av det föregående dygnet. Efter en avslutad sprint ska en produkt som potentiellt kan levereras till kund finnas tillgänglig. Då ska teamet även genomföra en sprint retrospective, vilket är en genomgång av processen under sprinten. Dessutom genomförs en sprint review, vilket är ett tillfälle då teamet och intressenter diskuterar det som producerats under sprinten (Scrum Alliance 2016). Hela arbetsgången åskådliggörs i Figur 1.




*Figur 1 Scrum*

## 2.3 KPI:er

Flexibilitet och kundvärde är centralt inom agil utveckling. Dessa koncept tillsammans med det inkrementella arbetssättet ställer nya krav på kontinuerlig styrning och uppföljning av arbetet. Lämpliga verktyg, i form av KPI:er, för hantering av dessa krav behandlas i detta avsnitt.

### 2.3.1 Definition av KPI

KPI står för Key Performance Indicator och är kvantifierbara mätetal som företag använder sig av för att jämföra hur de presterar i förhållande till strategiska och operationella mål (Investopedia 2016).

### 2.3.2 Nyttor med KPI:er

I kapitel 2.2.4 beskrevs att mjukvarurelaterad projektstyrning skiljer sig från traditionell projektstyrning bland annat eftersom framstegen inom en mjukvara är mindre synliga. KPI:er möjliggör att intressenter får en bättre uppfattning om hur projektet presterar och ifall det går enligt plan. KPI:er är därför särskilt aktuella inom mjukvarurelaterad projektstyrning. Om ett team inte vet hur de presterar relativt de mål de själva satt upp kan de omöjligen förutse om de kommer att bli klara i tid till nästa sprint. Om ledningen inte får reda på hur teamen presterar relativt uppsatta tidsgränser har de ingen chans att omplacera resurser för att säkerställa att projektet lyckas inom tidsramen (Broadus 2013, s. 52).

Genom att följa upp verksamheten med prestationsmätningar ges en större kontroll över vad som pågår relativt planen. Problemområden kan identifieras, vilka efter lärandeprocesser kan bidra till förbättringar och bättre planering framöver (Gustafsson 2011, s. 6). Förbättringarna kan i sin tur leda till reducerade cykeltider, högre produktivitet och lägre kostnader i längden (Gopal et al 2002, s. 863). Dessutom är medarbetarmotivering en vanlig orsak till användandet av mätningar. Efter uppsatta mål kan prestationer belönas (Gustafsson 2011, s. 6).

### 2.3.3 Svårigheter med KPI:er

Risken med att belöna efter ett mätningssystem är att fokus läggs på att förbättra mätparametern snarare än att eftersträva det underliggande målet. Aktiviteter som kan ge ett ökat kundvärde, men som inte kvantifieras i KPI:erna som en medarbetare blir bedömd efter, kan då hamna i skymundan. För att undvika denna suboptimering är det av vikt att använda mätningarna i informationssyfte, snarare än motivationssyfte. Därtill att engagera medarbetare vid införandet av mätningssystemet samt att vara försiktig med styrning efter mål (Gustafsson 2011, s. 5)

Många företag misslyckas med att implementera sina initiativ för processförbättringar. Till viss del beror det på sociala och organisatoriska faktorer, det vill säga hur acceptansen ser ut hos medarbetarna. En anledning till motstånd kan vara att team vill undvika att bli jämförda på mätningar som inte avspeglar verkligheten på ett trovärdigt sätt. Det är viktigt för medarbetarna att känna kontroll över sin arbetssituation. Ledningen tror ofta att utvecklarna inte är mottagliga för datainsamlingen som ligger till grund för mätningssystemet, vilket också formar utvecklarnas faktiska acceptans. Det är mer eller mindre standard inom mjukvaruutveckling att ljuga om tidsrapporteringen, det vill säga att arbetet beskrivs enligt vad ledningen önskar (Umarji & Seaman 2008, s. 129-130).

Automatiserade rapporteringsverktyg minskar individens arbetsbörda och bör således också minska det allmänna motståndet mot mätningarna. Det krävs dock alltid en viss insats från utvecklarna, och det kan finnas anledning att knyta incitament till korrekt inrapportering (Umarji & Seaman 2008, s. 129-130). Konkreta exempel på problem som uppstår är att medarbetare undanhåller information som kan skada en kollega, att tomma dokument bifogas vid rapportering samt att skript används för att fylla i fält (Umarji & Seaman 2008, s. 133). En svårighet i att finna en allmängiltig lösning för problematiken med mätningar inom mjukvaru-utveckling är att processerna är unika, både mellan företag och mellan personer eller team inom ett visst företag (Umarji & Seaman 2008, s. 129-130).

### 2.3.4 KPI:er i en agil kontext

Hartmann & Dymond har sammanställt en lista över principer för hur mätningar bör hanteras inom agil utveckling. De agila mätningsprinciperna säger bland annat att (Hartmann & Dymond 2006, s. 1-2):

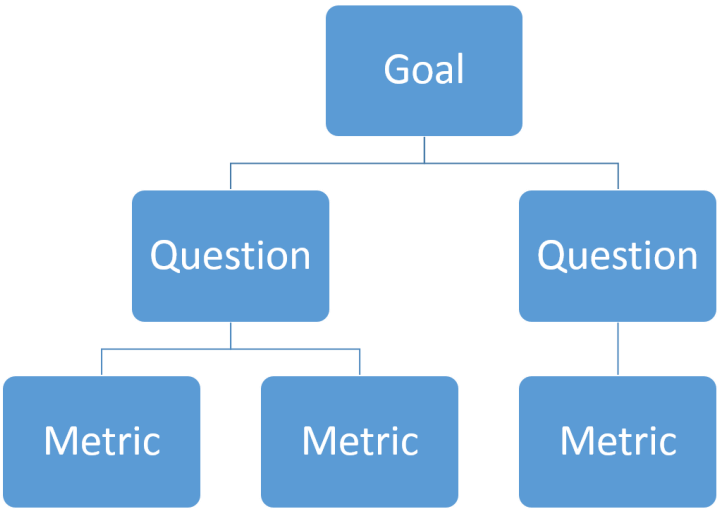
- Resultatet av ett utfört arbete ska mätas, inte själva arbetet i sig
- Mätningarna ska följa trender, inte statiska tal
- Det ska endast vara ett mindre antal KPI:er
- KPI:erna ska vara lätta att samla in
- Mätningarna ska främja meningsfull kommunikation och frekvent feedback
- Kvalitet som är ”bra nog” ska uppmuntras
- KPI:er kan även användas till att mäta värde eller processer

## 2.4 Modeller för framtagning av KPI:er

Det finns ett antal vanliga problem som företag stöter på angående styrningen med KPI:er. Exempel på dessa problem är att data samlas in på fel sätt eller att den inte analyseras tillräckligt. Problemen för med sig en mängd olika konsekvenser och resulterar ofta i att en misstro mot olika mätningar och KPI:er etableras inom företaget (Basili et al 2010 s. 1-2). Inom mjukvaruutveckling har målorienterade modeller vuxit fram för att styra insamlingen och analysen av data på ett systematiskt sätt. Namnet härstammar från det faktum att modellerna sammanfogar de övergripande affärsmålen med olika mjukvarustrategier och resultatet översätts sedan till en kvantifierbar projektstyrning. Exempel på målorienterade modeller som används inom mjukvaruutveckling är Goal-Question-Metric-modellen (GQM) och Practical Software Measurement (PSM) (Basili et al 2010, s. 3). Dessa har valts ut eftersom de är vanligt förekommande i litteraturen, och kommer att behandlas nedan.

### 2.4.1 GQM-modellen

GQM-modellen bygger på organisationers övergripande affärsmål. För att undersöka huruvida målen uppnås används KPI:er som även visar hur organisationer ska agera för att uppnå målen. Ett mål bryts först ner till ett specifikt format och ett antal frågor formuleras, ”Question”, som är till för att sätta målet i ett sammanhang. När frågorna besvaras erhålls en kunskap om huruvida målet är uppfyllt eller ej. Det är KPI:er som fungerar som svar på de olika frågorna (Prowareness 2016, s. 2). En grafisk representation av GQM-modellen presenteras i figur 2 nedan.



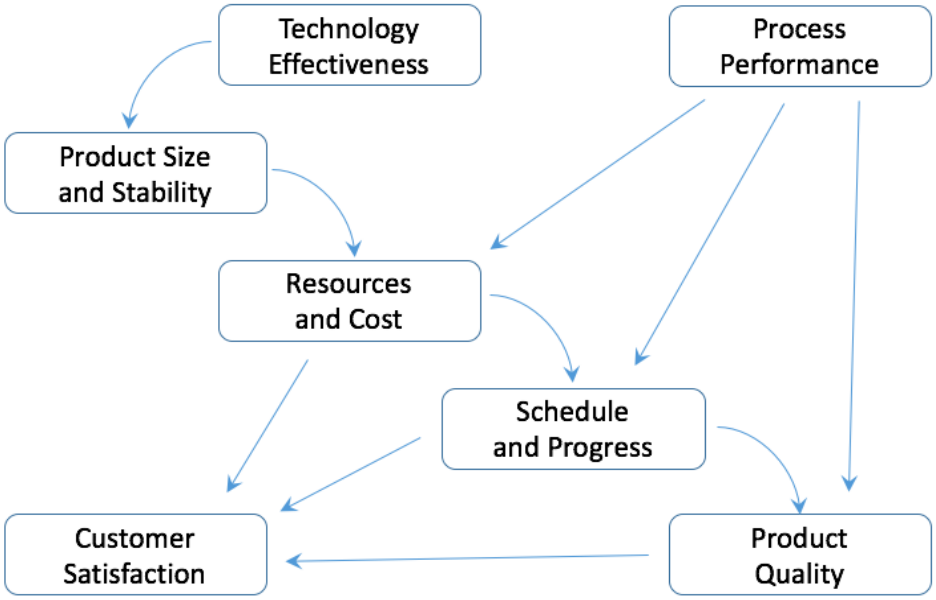

*Figur 2 GQM-modellen*

GQM-modellen användes först av NASA under 80-talet, då som ett sätt att utvärdera defekter. Sedan dess har GQM blivit ett paradigm för kvalitetsförbättring inom mjukvaruutveckling (Prowareness 2016, s. 2).

### 2.4.2 PSM-modellen

PSM är en mätningsprocess som adresserar en organisations unika affärsmål och tekniska mål. De riktlinjer som presenteras inom PSM bygger på bästa praxis från områdesexperter inom bland annat mjukvara. PSM är även sponsrad av den amerikanska armén och försvarsmakten vilket ger stor tillgång till praktisk erfarenhet. Chefer får den information som behövs för att de framgångsrikt ska kunna hantera kostnader, planering och de tekniska målen (PSM Support Center 2016).

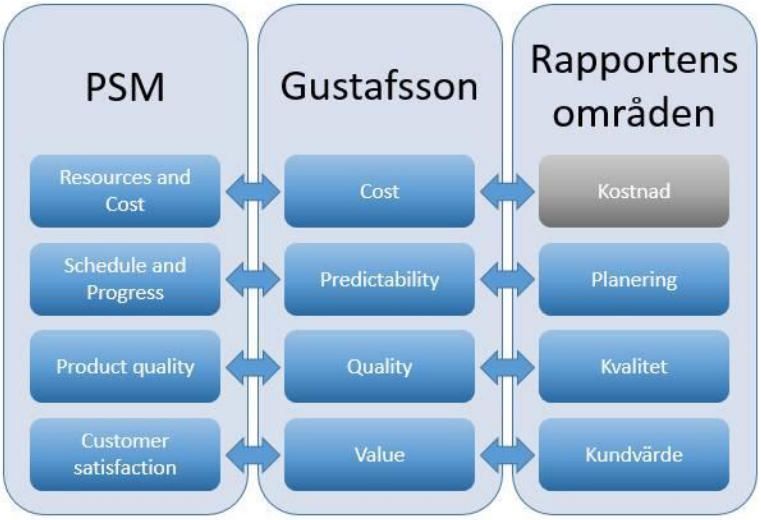
PSM innehåller detaljerade riktlinjer för hur man ska utföra mjukvarumätningar. Den inkluderar en katalog av specifika KPI:er samt medföljande information om hur man ska applicera dessa inom en organisation. PSM innehåller också en process som kan avgöra vilka KPI:er som ska användas beroende på olika problem och mål med det specifika mjukvaru-utvecklingsprojektet (Basili et al 2010, s. 3). Inom PSM delas informationsbehovet in i ett antal olika kategorier och inom varje kategori kan man sedan få exempel på specifika KPI:er. Kategorierna presenteras nedan i figur 3 tillsammans med deras inbördes beroenden.




*Figur 3 PSM-modellen*

## 2.5 Centrala områden och KPI:er för mjukvaruutveckling

I det föregående kapitlet presenterades flera anledningar till att använda modeller för framtagning av KPI:er. Den huvudsakliga anledningen är att KPI:erna får en starkare koppling till övergripande företagsmål. Inom PSM-modellen talar man bland annat om målen Schedule and Progress, Product quality, Customer Satisfaction och Resources and Cost (PSM Support Center, 2016). Dessa mål bekräftas av ytterligare en källa, i vilken de uttrycks som Predictability, Quality, Value och Cost (Gustafsson, 2011). Nedan presenteras de olika källornas mål i figur 4.




*Figur 4 Gemensamma kategorier*

Vidare kommer Predictability, Quality och Value att presenteras. Cost kommer att uteslutas eftersom KPI:er inom agil utveckling, som i enlighet med Delphis intresse ligger i fokus för rapporten, är ovanliga inom området (Jeeva Padmini, Dilum Bandara & Perera 2015, s. 315). Under vart och ett av områdena, som nedan kallas planering, kvalitet och kundvärde, kommer specifika KPI:er för varje område att tas upp. Dessa presenteras utefter användningsområde samt förutsättningar och problem för att belysa relevanta aspekter för studien. Strukturen kommer även att följas under kapitel 5, Analys och diskussion, för att möjliggöra en tydlig jämförelse. I slutet av kapitlet kommer en sammanfattande tabell över samtliga KPI:er att redovisas. Urvalet av KPI:er har baserats på hur vanligt förekommande de är inom agil utveckling samt hur stort intresse Delphi har visat för dem. Tidigt i studien redogjorde Delphi för vilka KPI:er som används i projektet och därefter kompletterades dessa med KPI:er från litteraturen.

### 2.5.1 Planering

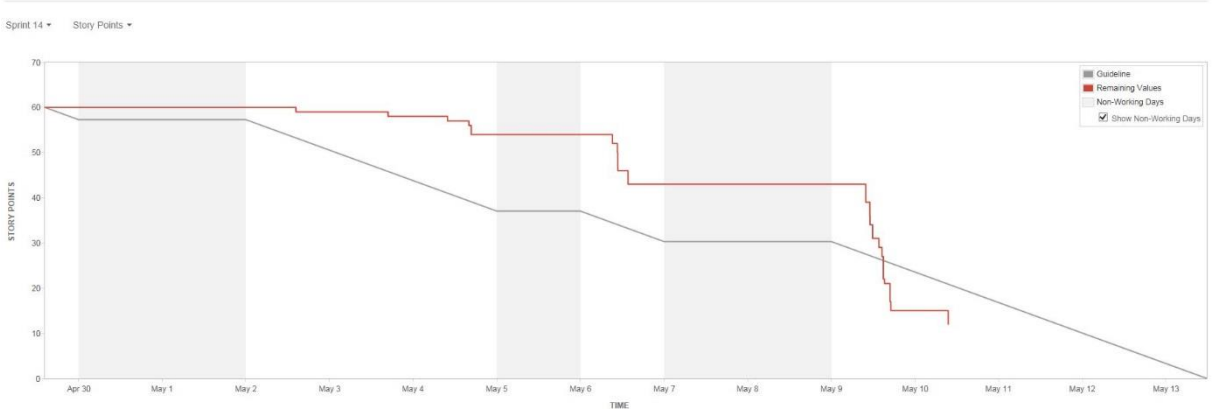
Inom området planering ingår KPI:er som är relaterade till förutsägandet av ett visst resultat. Samtliga av dem är alltså proaktiva och kan användas till att ge en indikation på om de mål som satts upp kommer att nås. De områden som kommer att presenteras i avsnittet är följande:

- ***Sprint burn-down:*** En graf över sprintens förlopp.
- ***Velocity:*** Kvantifieras för att ett team ska kunna estimera hur mycket arbete de kan ta på sig under en sprint.
- ***Burn-up:*** Ett diagram som används för att visualisera hur stor del av det totala arbete som ska levereras som har blivit avklarat. Används för att uppskatta om arbetet kommer bli klart i tid.
- ***Happiness:*** En genomsnittligt framtagen medarbetarnöjdhet. Ger en indikation på hur framtida prestationer kommer att se ut och kan även användas till att förebygga problem.

**Sprint burn-down**

En av de grundläggande byggstenarna till hela Scrumkonceptet är Sprint burn-down. KPI:ns enkelhet och överskådlighet för projekt har gjort den populär och frekvent använd hos mjukvaruutvecklande företag som använder Scrum som utvecklingsmetod (Mittal 2013).

Sprint burn-down ger information om en specifik sprint. KPI:n visualiseras i en graf, med tid på x-axeln och antal story points på y-axeln. Denna graf innehåller två linjer, vilka kan ses i figur 5. Den första är en jämnt sluttande linje som visar på en optimal lutning, då detta innebär en jämn arbetstakt för teamet under samtliga veckodagar för hela sprinten. Den andra linjen följer det faktiska antalet story points som teamet har avklarat. Om den faktiska linjen är brantare än den optimala jobbar teamet i högre takt än estimeringen (Prowareness 2016, s. 11).




*Figur 5 Graf över Sprint burn-down*

*Användningsområden*

Syftet till att man använder Sprint burn-down är olika, beroende på vem intressenten är. Ur utvecklingsteamets perspektiv är det viktigt med kontinuerlig och detaljerad information om hur arbetet fortlöper. Detta för att kunna se vad som händer med projektet, kunna diagnostisera problem som uppstår samt identifiera områden för förbättring. Sprint burn-down kan hjälpa till med att svara på dessa frågor genom att bland annat ge svar på hur väl teamet hanterar uppskattningen av arbetsbördan (Prowareness 2016, s. 11). En viktig aspekt med Sprint burn-down är att KPI:n ger information som motiverar utvecklingsteamet att fortsätta jobba. Om teammedlemmarna får direkt feedback på att mer och mer arbete kontinuerligt slutförs kan teamet lättare se när projektet kommer bli färdigt och att framsteg faktiskt görs (Prowareness 2016, s. 11). Ur ledningens och kundens perspektiv är det av vikt att få en generell överblick över projektet, främst genom att veta om projektet är inom rätt tid, budget och omfång (Broadus 2013, s. 52-53).

*Förutsättningar och problem*

Utöver fördelarna med Sprint burn-down finns det en del problem som främst hör till hur projektet är definierat. Ett av dessa problem är att flera stories kan ha en gemensam uppgift. Detta kan leda till att en uppgift rapporteras in som löst under flera stories, vilket ger uppfattningen om att mer arbete är gjort än i verkligheten (Mittal 2013).

Ett annat problem är att uppgifterna är för detaljerade eller för stora. Detta gör det svårt för utvecklingsteamet att uppskatta hur mycket som är färdigt och de kan rapportera in felaktig data, vilket gör att KPI:n kan ge missvisande information. Detta löses främst genom att göra arbetsuppgifterna mindre (Mittal 2013).

Enligt Mittal är ännu ett misstag att man missuppfattar kvarstående arbete som färdigt arbete. Detta är något som inträffar hos mjukvaruutvecklare som är nya till Scrum, och något som kan lösas genom utbildning. Dessutom, likt alla andra KPI:er, krävs ett kontinuerligt och dedikerat uppdaterande av KPI:n i mjukvara (exempelvis JIRA), för att KPI:n ska ge korrekt information om projektet (Mittal 2013).

Om Sprint burn-down kopplas till ett socialt hållbarhetsperspektiv kan vissa negativa egenskaper urskiljas. Framförallt handlar det om att utvecklingsteamet måste jobba övertid för att uppnå kraven som ställs. Ofta ställs dessa krav av andra intressenter, men ibland även av utvecklingsteamet själva. En möjlig orsak till att utvecklingsteamet själva ställer orimliga krav är ”superman syndrome”. Syndromet beskrivs som tendensen att överestimera sin möjliga prestationsförmåga i syfte att göra ledningen nöjd, samt att inte ta in osäkerhet i sin estimering (Matarelli 2011).

Effekterna av den ovan beskrivna ohållbara arbetssituationen är många. En av konsekvenserna är en lägre motivation, vilket i sin tur kan leda till en försämring av arbetstempot. Utvecklingsteam som måste föra över arbete, som inte blivit klart, från en sprint till nästa upplever frustration och stress. Samtidigt upplever ofta de utvecklingsteam som lyckas göra färdigt allt planerat arbete till en sprint en höjd moral (Matarelli 2011).

För att främja social hållbarhet är det viktigt med en stark kommunikation mellan utvecklingsteam och externa kravsättare. Framförallt påpekas vikten av att ge produktägaren större befogenhet att fatta beslut om hur mycket utvecklingsteamet klarar av att utföra i en sprint. Det finns alltid mer arbete som måste göras, men att lägga in mer arbete i en sprint innebär inte att mer blir gjort, utan kan leda till ovan nämnda konsekvenser (Matarelli 2011).

**Velocity**

Velocity, mätt på ett team, är antalet story points som teamet har gjort klart under en sprint. Om ett team gör färdigt två user stories under en sprint, värda två respektive fyra story points, blir teamets Velocity under den sprinten sex story points (Prowareness 2016, s. 10-11). Velocity är en grundläggande KPI som används inom agil mjukvaruutveckling. Närmare 80 % av företag utnyttjar den enligt en studie och informationen kan hämtas direkt från programvaror, som exempelvis JIRA (Jeeva Padmini, Dilum Bandara & Perera 2015, s. 315-317).

Teamet kvantifierar en user story i story points och därefter fungerar denna som en utgångspunkt för fortsatt estimering. Olika team kommer därför inte att ha matchande estimat för samma uppgift, vilket medför att team inte kan jämföras med varandra. Istället sker jämförelser av tillväxten på Velocity över tid med syftet att undersöka om teamet ökar kvantiteten av arbete som levereras (Prowareness 2016, s. 10-11).

När utvecklingen av Velocity studeras kan eventuella avvikelser bero på flertalet faktorer. Till exempel kan medlemmar i teamet ha drabbats av sjukdom eller blivit omplacerade. Om avvikelsen beror på att en user story var för komplex, med andra ord att teamet inte var kapabelt till att slutföra den inom den aktuella sprinten, borde produktägaren vidta åtgärder och granska product backlog (Prowareness 2016, s. 10-11). Ett sätt att hantera felaktiga estimeringar som leder till en missvisande Velocity är att dela in arbetsmomenten i ännu mindre delar. Genom att medarbetarna får analysera funktionalitet i mer detalj under planeringsstadiet kan precisionen för Velocity ökas (Kupiainen, Mäntylä & Itkonen 2015, s. 154).

*Användningsområden*
Velocity kan användas i flera syften. Exempelvis används den för att öka förutsägbarheten. Velocity blir då en grund att stå på när leveranskapaciteten ska fastställas. KPI:n gör att teamen, baserat på tidigare resultat, kan prognostisera hur mycket arbete de kommer att kunna slutföra inom en bestämd tidsram, till exempel under nästa sprint (Hartmann & Dymond 2006, s. 5). På längre sikt kan ett genomsnitt för Velocity användas för att planera leveranser (Kupiainen, Mäntylä & Itkonen 2015, s. 150).

Utifrån Velocity kan olika nivåer av leveranser definieras för den kommande sprinten. En miniminivå med krav på viss funktionalitet samt ett högre mål med lägre prioriterade uppgifter om det finns tid över (Kupiainen, Mäntylä & Itkonen 2015, s. 150). Velocity kan även användas för att mäta produktivitet, men den ger inte alltid en rättvisande bild då den bland annat inte tar hänsyn till kundvärde (Kupiainen, Mäntylä & Itkonen 2015, s. 157). Om Velocity används som ett mått på produktivitet riskerar kvaliteten att försämras (Gustafsson 2011, s. 9).

Ett alternativt användningsområde för Velocity är att dividera den med Work capacity. Work capacity är den totala mängden avklarat arbete under en sprint. Andelen man får ut ligger mellan noll och ett och benämns Focus factor. Focus factor visar alltså på den andel tid som ett team lägger på åtaget arbete från sprint backlog i relation till deras totala tid. Värdet på KPI:n ska öka med tiden då teammedlemmar lägger mindre tid på ej åtagna aktiviteter i och med att de lär sig att arbeta med Scrum-metodiken. Eftersom Focus factor använder procentsatser och andelar går KPI:n att jämföras mellan olika team. En Focus factor förväntas öka med tiden och ett värde på 0,8 anses vara acceptabelt (Prowareness 2016, s. 7-9). Låga värden på Focus factor indikerar att team har svårt med att omvandla planerat arbete till accepterat arbete. Höga värden brukar i sin tur indikera att teamet har underskattat sin förmåga för att få fram ett värde som anses vara perfekt (Downey & Sutherland 2013, s. 4874-4875).

*Förutsättningar och problem*
Velocity baseras bara på de arbetsuppgifter som är schemalagda, det vill säga estimerade i story points (Prowareness 2016 s. 10-11). Denna omständighet medför att problem kan uppkomma med att arbetsuppgifter som inte är förknippade med story points, exempelvis tester, prioriteras ner och kvaliteten försämras. Ett för stort fokus på Velocity, och att hålla en speciell nivå på den, kan även leda till att genvägar tas och att kvaliteten prioriteras ned (Kupiainen, Mäntylä & Itkonen 2015, s. 154-155). Sådan hantering ligger inte i linje med principerna för agil utveckling. Åtgärden bör snarare vara att minska omfånget genom att utelämna viss funktionalitet för den aktuella arbetsperioden (Kupiainen, Mäntylä & Itkonen 2014, s. 27). Prestation i tidigare sprintar har i vissa fall ignorerats och lett till allt för optimistiska estimeringar. Till exempel om deadline för nästa leverans sätter press på ett utvecklingsteam kan de tänkas utlova mer funktionalitet än vad som är möjligt att arbeta fram under den kommande sprinten (Mahnic & Zabkar 2012, s. 74). Undersökningar visar att hälften av alla agila mjukvaruutvecklingsteam inte känner till deras Velocity, samt att de upplever svårigheter i att hitta förbättringsmöjligheter (Downey & Sutherland 2013, s. 4870).

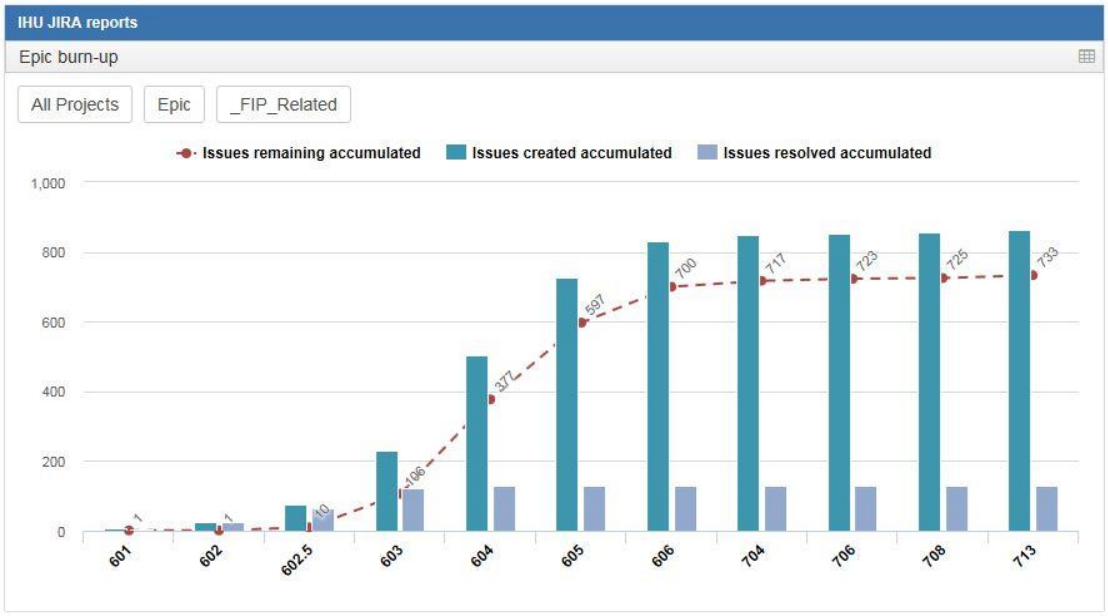
Om ledningen styr efter, och uppmuntrar, att teamen ska ha en hög Velocity kan teamen estimera fler story points till samma arbete, vilket medför att Velocity förlorar i verkan och inte längre kan användas för att prognostisera (Hartmann & Dymond 2006, s. 5).

Att lägga till ytterligare personal till ett team för att höja Velocity kan få motsatt effekt. Det blir ett avbrott i inarbetade mönster och orsakar istället produktivitetsförluster (Mahnic & Zabkar 2012, s. 74). En kundvärdesaspekt kan kombineras med Velocity för att skapa en djupare bild. En ökning i Velocity kan bero på att lättare problem tacklas, och en minskning kan bero på att svårare uppgifter behandlas som är kopplade till mer värdeskapande. Det är alltså inte syftet att bara sträva efter en högre Velocity om detta kommer till kostnaden av minskat kundvärde (Prowareness 2016, s. 8-9).

**Burn-up**

Burn-up är en KPI som mäter hur mycket av ett projekt som har blivit implementerat jämfört med hur mycket som finns kvar (Prowareness 2016, s. 11). Under ett mjukvaruprojekt omdefinieras kraven ständigt, vilket innebär att projektets storlek ofta inte är detsamma vid starten som när projektet är klart. Därför implementeras med fördel en Burn-up chart istället för en Burn-down chart i och med att det är enklare att lägga till mer i en Burn-up (Arafeen & Bose 2009, s. 109)

KPI:n avläses på en graf med arbete på y-axeln och tid på x-axeln. Arbete kan representeras av olika saker, till exempel features, epics eller stories. Längs x-axeln går en linje för antaget arbete och en för avklarat. Tanken är att de två linjerna ska gå ihop vid projektets slut för att visa att allt har blivit implementerat (Prowareness 2016, s. 11). Ett exempel på en graf över Epic burn-up visas nedan i figur 6.




*Figur 6 Graf över Epic burn-up*

*Användningsområden*

En Burn-up-graf är ett verktyg för att estimera när ett projekt förväntas bli klart. Om företaget vet sin Velocity, och räknar med att behålla denna, kan ett slutdatum för när projektet beräknas vara färdigt bestämmas (Arafeen & Bose 2009, s. 113). KPI:n kan också användas för att motivera medarbetare. De kan se hur mycket som är kvar av projektet i förhållande till hur mycket som har avverkats (Broadus 2013, s. 52). Kommunikationen och förståelsen för produktens och projektets status underlättas således (Power 2011, s. 207). Status i det här fallet kan till exempel innebära arbetstakt och datum för leveranser (Vásquez & Díaz 2011, s. 174). Tillförlitligheten till vad som har levererats är hög tack vare att enbart funktioner som är fullt integrerade och testade visas i grafen (Cockburn 2006, s. 14).

*Förutsättningar och problem*
Om ett projekt har ett fast omfång uppfylls inte nyttorna med att kunna följa tilläggen under projektets gång, vilka beskrevs i det inledande stycket till detta kapitel. Samma information fås i så fall från en Burn-up-graf som en Burn-down. Burn-down-grafen skulle i så fall vara att föredra tack vare dess enkelhet (Clarios Technology 2016a). Burn-up-grafer beskrivs vara komplexa och innehålla mer information än Burn-down. Därför kan det behöva läggas mer kraft på att förklara resultateten, och nyttorna kan falla undan till exempel om det gäller presentationer inför större grupper (Clarios Technology 2016b).

**Happiness**
Meningen med Happiness är att den ska återspegla hur väl medarbetare trivs på sitt arbete. För att de ska trivas krävs det att självständighet, skicklighet och ett syfte upplevs (Sutherland 2014, s. 153). Utvärderingen ska ske en gång efter varje sprint, i samband med sprint retrospective (Sutherland 2014, s. 150). Utgångspunkten är fyra frågor som Jeff Sutherland, en av grundarna av Scrum, har formulerat (Sutherland 2014, s. 151):

1. På en skala från 1 till 5, hur känner du angående din roll på företaget?
2. Med samma skala, hur känner du angående företaget överlag?
3. Varför känner du på det sättet?
4. Nämn en sak som skulle göra dig lyckligare under nästa Sprint.

Det är viktigt att utvärderingen sker ofta. Anledningen till detta är att Scrum försöker efterlikna Lean Production och åstadkomma ”Kaizen”, vilket innebär ständiga förbättringar. Allt kan inte göras på en gång och perfektion kan aldrig uppnås, men varje steg i rätt riktning räknas (Sutherland 2014, s. 149-150).

*Användningsområden*
En undersökning som gjordes 2005 med 275 000 deltagare kom fram till att (Sutherland 2014, s. 148):

”Happiness leads to success in nearly every domain of our lives, including marriage, health, friendship, community involvment, creativity, and, in particular, our jobs, careers, and businesses.”

Enligt samma undersökning är lycka något som leder till dessa framgångar, snarare än något som genereras av framgångarna. Detta innebär att medarbetare måste vara lyckliga för att prestera bra (Sutherland 2014, s. 148). Medarbetarnas prestationer är i sin tur direkt relaterade till företagets prestation och lönsamhet. Av den här anledningen krävs det att ledningen håller koll på sina medarbetares fysiska och psykiska tillstånd, och att de hittar ett effektivt sätt att mäta det på (Sutherland 2014, s. 152).

Syftet med mätningar av Happiness varierar beroende på intressent. Syftet för en medarbetare kan vara att denna troligtvis kommer att må bättre på sin arbetsplats och därmed få en högre livskvalitet (Sutherland 2014, s. 148). För ledningen är syftet med mätningen att förbättra företagets prestation för att kunna öka förutsättningarna för framtida vinster (Sutherland 2014, s. 152). Eftersom lycka uppges leda till framgångar är Happiness en KPI som säger något om framtiden, varför den anses vara en förutsägande KPI. Därmed kan den användas för att förhindra att exempelvis lägre produktivitet inträffar, och på så sätt vara proaktiv (Sutherland 2014, s. 152).

*Förutsättningar och problem*
En förutsättning som nämnts ovan är att mätningarna måste ske ofta, helst en gång efter varje sprint. Eftersom Happiness är en indikation på hur verksamheten kommer att prestera framöver är det viktigt att snabbt fånga upp och hantera problem. Tidigare nämndes också att självständighet, skicklighet och syfte behöver uppnås för att lycka ska erhållas. För att varje individ på företaget ska känna ett syfte krävs det att de känner delaktighet för att uppnå företagets övergripande mål (Sutherland 2014, s. 153). Om detta ska åstadkommas krävs det i sin tur att allt är synligt för alla medarbetare. När en utvecklare läser resultaträkningen för koncernen ska de veta vilka delar de har bidragit med för företaget som helhet (Sutherland 2014, s. 154). Ytterligare en förutsättning är att ledningen agerar på de klagomål och problem som identifieras efter varje sprint. Att ta upp problem utan att någon sedan bryr sig om det har en negativ effekt på medarbetarnöjdheten (Harnish & Ross 2013, s. 6).

Ett problem är att vissa personer definierar lycka som möjligheten att jobba mindre, men fortfarande få samma ersättning. Företaget bör sträva efter att anställa folk som använder lycka som något som driver dem, som stärker deras arbetsmoral och får dem att vilja jobba ännu mer. Därför är det viktigt att ständigt jämföra Happiness mot prestation. De ska korrelera väl, annars används Happiness på fel sätt i företaget (Sutherland 2014, s. 161). Ett annat problem är den så kallade ”lyckobubblan”. Medarbetare kan hamna i lyckobubblan när de känner att de har gjort stora framsteg. Framstegen är så pass stora att vissa av medarbetarna får känslan av att de inte behöver förbättras mer. Arbetarna i fråga kommer fortsatt att visa positiva resultat när de mäts på Happiness, men det kontinuerliga förbättringsarbetet har upphört. Återigen är lösningen att noggrant jämföra Happiness med faktiska resultat (Sutherland 2014, s. 162).

### 2.5.2 Kvalitet

En av de viktigaste komponenterna vid mjukvaruutveckling är kvaliteten. Det är inte tillräckligt att koden endast är körbar. Kraven och förutsättningarna ändras i hög takt och det krävs därför flexibilitet i form av högkvalitativ mjukvara för att svara på dessa utmaningar. En väl utvecklad kod kan svara mot komplexa frågeställningar och ger lägre kostnader i längden (Petersson & Zhang 2013, s. 1).

Närmare hälften av alla resurser i ett mjukvaruutvecklingsprojekt spenderas på omarbete som hade varit möjligt att undvika. Att upptäcka felaktigheter tidigt i processen är därför viktigt för att minska den skada som skett och måste repareras. Att hitta och rätta till ett fel efter leverans jämfört med att göra det under krav- och designfasen är i storleksgraden hundra gånger så dyrt (Boehm, Rombach & Zelkowitz 2005, s. 426-427). En grundläggande anledning till att defekter uppmärksammas under ett projekt är alltså att undvika omarbete längre fram. På det viset fungerar mätningarna som insatser för framtida produktivitetsförbättringar (Boehm, Rombach & Zelkowitz 2005, s. 427).

I kapitlet kommer följande tre KPI:er att beskrivas:

- ***Defect count:*** Totala antalet defekter i projektet.
- ***Bug correction time:*** Stängningstiden för defekter, det vill säga hur lång tid det tar att åtgärda dem från det att de upptäckts.
- ***Technical debt:*** Den skuld, arbete som behöver göras om senare, som uppstår på grund av slarvigt kodade avsnitt.

**Defect count**
Antal defekter kan mätas antingen som ett absolut tal eller ställas i relation till något, exempelvis tid eller antal kodrader. För en utvecklare är båda av intresse för att ta reda på hur kvalitativ kod som skrivs och för en kund är det främst av intresse att känna till det totala antalet defekter (Kan 2003, s. 92).

*Användningsområden*
Ett användningsområde fås genom synen på defekter som ett mått på kvalitet i mjukvara. Kunder vill ha felfria leveranser och antalet defekter ska därför minimeras. För att motivera medarbetare till ökat fokus på kvalitet och lågt antal defekter kan status visualiseras genom exempelvis tavlor i korridorer. Genom hög transparens om den aktuella problemsituationen tros medarbetarna få större incitament till att snabbt hitta lösningar. KPI:er är inte endast viktiga för ledningsnivå, utan hela organisationen kan dra nytta och förstå kvalitetsnivån (Cheng, Jansen & Remmers 2009 s. 33). Så väl ett mått på kvaliteten i dagsläget som en förutsägelse om framtida kvalitetsnivå fås (Kupiainen, Mäntylä & Itkonen 2014, s. 26). Utvecklarna assisteras då de lättare kan fatta beslut om att lösa problemen. För högre projektledare är det viktigast att förstå om den samling av defekter som byggts upp kan arbetas ned till noll innan projektets slut. Här ligger fokus alltså snarare på mängden samt hur trenden ser ut, det vill säga hur nivåerna utvecklas av defekter (Staron, Meding & Söderqvist 2010, s. 1078). Generellt sett behöver ledning på högre nivå ingen detaljerad information om specifika defekter, utan de behöver endast förstå påverkan på projektets kostnader, tidsplan och omfång (Broadus 2013, s. 52).

Även på kortare sikt kan nyttor uppnås. Det finns företag som ser över nivåerna av öppna defekter efter varje sprint för att få ett mått på kvaliteten från den arbetsperioden. Problem som uppstått kan då tas upp i sprint retrospective för att försöka undvika dem i framtiden (Jeeva Padmini, Dilum Bandara & Perera 2015, s. 315). Efter det att leveransen väl har skett kan mått som antal defekter samt förändringsförfrågningar från kunderna användas för att bedöma kvaliteten (Kupiainen, Mäntylä & Itkonen 2014, s. 26). Att hålla nere defektantalet är ett mycket viktigt mål för utvecklingsteam eftersom kundnöjdhet är det som prioriteras allra högst i slutänden. Fokus bör ligga på resultat snarare än ren output, vilket talar för att KPI:er inte ska användas om de inte slutligen leder till ett ökat affärsvärde (Hartmann & Dymond 2006, s. 1-2).

*Förutsättningar och problem*
En viktig förutsättning för att Defect count ska vara användbar för att utvärdera kvalitet är att testningsprocessen är uttömmande. Antalet defekter är starkt relaterat till hur bra testnings-processen är och brister i den kan således leda till den felaktiga bedömningen att mjukvaran är felfri (Prowareness 2016, s. 12).

Det finns flera problemområden kopplade till mätningar av antalet defekter inom mjukvaruutveckling. Om allt för mycket vikt läggs vid att nivåerna ska minimeras kommer inte medarbetarna att kunna utföra sina arbetsuppgifter på ett optimalt sätt. Om till exempel nya idéer och principer för mjukvaran identifieras, som kan tänkas göra systemet mer kostnads-effektivt, bör dessa också implementeras. Däremot finns det en risk att medarbetarna avskräcks från detta i och med att de måste klassificera ändringarna som defekter. Defekterna på kort sikt ser då bättre ut, men på längre sikt kan de tänkas öka och även bidra till ett suboptimalt kundvärde (Boehm, Rombach & Zelkowitz 2005, s. 427). Ett annat problem är att KPI:n inte avslöjar något om hur allvarliga defekterna är. Utan hänsyn till detta är det svårt att uppskatta den egentliga kvaliteten (Prowareness 2016, s. 12).

Att identifiera ett behov av att skjuta fram datum för en leverans, genom att se över antalet öppna defekter, togs tidigare upp som ett användningsområde. Detta strider dock mot de principer som agil utveckling är uppbyggda kring. Istället för att arbeta längre eller ta in mer personal bör snarare omfånget minskas till en mer hanterlig grad (Kupiainen, Mäntylä & Itkonen 2014, s. 27).

**Bug correction time**

En annan dimension är tidsåtgången från öppning till stängning av felen. På det sättet visas hur snabba medarbetarna är på att lösa defekterna (Jeeva Padmini, Dilum Bandara & Perera 2015, s. 316).

*Användningsområden*

Om företaget har en god uppfattning om hur lång tid det tar att åtgärda defekter av olika typer underlättar detta planeringsarbetet (Jeeva Padmini, Dilum Bandara & Perera 2015, s. 316). Måttet kan också användas för att prioritera arbetsuppgifter. Ny funktionalitet måste ständigt utvecklas men det får inte gå ut över kvaliteten allt för mycket. Om genomsnittstiden för att åtgärda defekter växer kan det betyda att för stort fokus läggs på ny funktionsutveckling, vilket inte är hållbart i längden. Kvaliteten måste alltså höjas och i värsta fall kan sprinten tvingas avbrytas, alternativt att tänkt arbete i senare sprintar blir försenade (Mannila 2013, s. 31).

*Förutsättningar och problem*

En korrekt uppskattning av tiden det tar att åtgärda en defekt förutsätter att en bra modell för uträkningen används. Enligt en studie som gjorts på University of California, USA, finns det i dagsläget inga sådana modeller. De modeller som används idag har en förutsägelsekraft i spannet 30-49 % och är i behov av fler oberoende variabler för att kunna förutspå Bug correction time på ett mer tillförlitligt sätt (Bhattacharya & Neamtiu 2011, s. 1).

Ett problem är att det finns delade meningar om nyttan med att mäta tidsåtgången från öppnad defekt till stängd. Ett mått på lösningseffektiviteten fås, men det blir snarare en utvärdering av individuella utvecklare än teamets förmåga. Agil utveckling uppmuntrar arbete i team och ämnar dra bort fokus från den individuella prestationen. Det agila teamet uppnår därför inget värde av att använda måttet. En del hävdar att måttet inte är intressant alls, bara defekterna till slut löses (Jeeva Padmini, Dilum Bandara & Perera 2015, s. 315).

**Technical debt**

Inom mjukvaruutveckling finns ett koncept som kallas Technical debt, vilket beskrivs som det ackumulerade omarbete som följer av en sänkt kvalitet över tiden. Mycket fokus inom agil mjukvaruutveckling läggs på värdeadderande arbete som ger snabba leveranser och hög anpassningsbarhet till kundens föränderliga krav. Med det följer en risk att för lite tid ägnas åt defekthantering och omstrukturering av koden. I längden blir koden allt mer svårhanterlig och Technical debt byggs upp. Problemsituationen som uppstår har fått namnet “Start-n-stop” hos företaget Ericsson, med anledning av att funktionsutveckling under perioder får pausas i syfte att höja den begränsande kvaliteten på mjukvaran. Detta ställer till problem för leverantörerna, då förmågan att kontinuerligt kunna sköta kundleveranser är en vital konkurrensfördel på marknaden. Målet bör snarare vara en proaktiv utveckling som ständigt håller kvaliteten uppe (Sandberg, Staron & Antinyan 2015, s. 1-2).

*Användningsområden*
Att koda slarvigt, utan användning av bland annat kommentarer och möjlighet till återanvändbar kod, är oftast ett snabbt sätt att implementera funktionalitet. Eftersom funktionalitet är det som motsvarar värdet för kunden är det svårt att motivera ett högre pris för snyggt skriven kod. Av den anledningen kan det, i vissa lägen, vara lönsamt att bruka en hög grad av Technical debt (Wolff & Johann 2013).

Eftersom Technical debt är ett mått på hur väl koden har designats i projektet är det också ett mått på hur lätt eller svårt det är att utveckla koden i framtiden. En hög grad av Technical debt motsvaras av att det är svårt att lägga till funktionalitet och vice versa. Ett användningsområde är därför som ett planeringsverktyg, för att exempelvis kunna uppskatta tidsåtgången för att utveckla ett redan existerande system (Wolff & Johann 2013).

*Förutsättningar och problem*
För att kunna uppskatta hur mycket Technical debt ett projekt innehåller krävs en bra modell. Att mäta Technical debt är lika svårt som att kvantifiera exempelvis mjukvaruproduktivitet och mjukvarukvalitet. Därför ska måttet enbart ses som en uppskattning (Sharma & Smarthyam 2015).

Det är inte alltid lönt att åtgärda Technical debt. Det är dock svårt att avgöra i vilka fall en åtgärd är nödvändig och i vilka fall den inte är det. Investerad tid i att omstrukturera koden idag måste kunna tjänas igen genom att implementering av kod går snabbare i framtiden. I större projekt eller tidigt i projekt är valet oftast inte svårt. I mindre projekt eller i slutet av ett projekt kan övervägningen bli svårare (Wolff & Johann 2013).

### 2.5.3 Kundvärde

Då kunder ofta baserar sina inköpsbeslut på hur mycket nytta de får ut av produkten eller tjänsten är det viktigt att mäta kundvärde. Därför är det viktigt att som företag veta hur sitt värdeerbjudande ska anpassas för att inte förlora kunder till konkurrenter. Många företag upplever dock svårigheter med att definiera hur mycket värde de levererar i deras produkter eller tjänster (Anderson & Narus 1998).

När kundvärde betraktas ur ett Scrum-perspektiv blir begreppet ännu viktigare. Stora mängder kod ger ingen garanti för att ett högt kundvärde levereras, viktigt är istället att möta kundens krav. Framförallt stora och detaljerade projekt kan skapa frågetecken om kundvärde. Då utvecklingsteamen är självständiga måste de vara noggranna i vad de utvecklar för att undvika överutveckling av mjukvara, det vill säga att lägga till ytterligare funktioner eller funktionalitet som inte skapar något kundvärde. Det är viktigt med en stark kommunikation mellan utvecklingsteamen och ledningen för att säkerställa att rätt egenskaper utvecklas i mjukvaran, och på så sätt inte förlora tid på att utveckla funktionalitet som inte skapar något kundvärde (Geffel 2011).

Kapitlet behandlar följande två KPI:er:

- ***Net promoter score:*** Beskriver hur stor sannolikheten är att en kund rekommenderar företagets tjänster till en vän eller kollega.
- ***Earned business value:*** Kvantifierar hur mycket kundvärde som levererats i ett projekt.

**Net promoter score**
NPS används för att bestämma kunders lojalitet till företaget på en skala från -100 till 100. En högre siffra innebär att kunder rekommenderar företagets produkter och tjänster till andra kunder. NPS räknas ut genom kundundersökningar, i vilka kunder får gradera företaget på en 11-gradig skala med avseende på om de skulle rekommendera företaget till en vän eller kollega. Frågan lyder mer precist: ”På en skala från 0 till 10, hur sannolikt är det att du skulle rekommendera företagets produkter eller tjänster till en vän eller kollega?” Svarsgivarna delas sedan in i tre grupper. De som givit 0-6 poäng kallas kritiker, 7-8 kallas passiva och 9-10 kallas ambassadörer. Det slutgiltiga resultatet, NPS, beräknas genom att från andelen ambassadörer subtrahera andelen kritiker. Då fås en siffra mellan -100 och 100 (Medallia 2016).

Till skillnad från andra verktyg för kvantifiering av kundvärde, som kan vara både komplicerade och tidskrävande, är NPS graderingsbaserat, lättanvänt och ger snabba resultat (Reichheld 2003). Resultatet är lätt att förstå och tolka, vilket är en orsak till dess stora popularitet inom olika branscher (Medallia 2016). Enligt en studie som gjorts, omfattande 51 olika verksamheter, över de mest använda KPI:erna inom agil utveckling var resultatet att NPS användes i 54 % av de undersökta företagen (Jeeva Padmini, Dilum Bandara & Perera 2015, s. 315).

*Användningsområden*
Resultatet från NPS kan enkelt förmedlas ner i företagshierarkin, då resultatet är lättförståeligt. Därmed kan anställda som jobbar mot kund förstå om de behöver göra något annorlunda för att öka NPS. Resultatet ger även motivation till de anställda att arbeta bättre i syfte att öka företagets NPS (Medallia 2016). Den resulterande NPS har i undersökningar visat sig ha tydliga kopplingar till företags tillväxt, samt att branschledande företag förknippas med en dubbelt så hög NPS som genomsnittet i branschen (Markey 2014).

*Förutsättningar och problem*
Trots Net promoter scores enkelhet och övriga fördelar finns det flera nackdelar. Exempelvis svårigheten i att identifiera orsakerna bakom respondenters svar på frågan som ställs. Orsakerna till varför en kund rekommenderar eller inte rekommenderar ett företag kan inte tolkas från en siffra och därmed finns svårigheter i hur företag ska arbeta för att förbättra sin NPS. Om ett företag inte kan göra förbättringsarbete på grund av bristande data på vad som behöver förbättras finns en risk att kunder blir frustrerade och väljer konkurrenter istället (Cooil et al 2016).

Det ovanstående problemet kan kringgås genom att ställa kompletterande frågor, men då på bekostnad av Net promoter scores tidigare beskrivna primära fördelar som är enkelhet och att snabbt få resultat. Frågan som NPS ställer är riktad mot kundens krav och inte företagens behov. Att då lösa detta genom att komplettera med ytterligare frågor för att ge företaget mer data står i direkt kontrast till Net promoter scores huvudsyfte (Cooil et al 2016).

En annan aspekt är kulturella skillnader i betygsättning på den globala marknaden. I vissa länder ges betyg 9 och 10 oftare ut än i andra länder, i vilka dessa betyg reserveras till att företaget givit en exceptionell standard. En viktning av betygen finns inte (Cooil et al 2016).

Företag som arbetar i industriella miljöer har ofta ett fåtal företag som står för den största omsättningen. Detta benämns som paretoprincipen eller 80/20-regeln, vilket innebär att 20 % av kunderna står för 80 % av omsättningen (Lavinsky 2014). Då NPS inte tillämpar en viktning med avseende på kundernas bidrag till omsättningen riskerar måttet att bli ineffektivt (Cooil et al 2016).

**Earned business value**
I takt med att den traditionella vattenfallsmodellen överges och istället byts ut mot agila utvecklingsmetoder uppstår även frågan om hur värde ska hanteras. I den tidigare vattenfallsmodellen fastställs budget i början av projektet beroende på projektets omfång och komplexitet. Genom övergången till agil utveckling med iterativ utveckling krävs nya hjälpmedel, som exempelvis Earned business value, för att kunna uppskatta hur mycket värde som skapas under utvecklingsfasen (Katham 2015).

Earned business value beskrivs som en metod för att kommunicera hur agila projekt levererar värde. I praktiken innebär det att user stories från product backlog läggs till i projektets Earned business value när de blir färdiga. Med denna data kan en graf ställas upp med Earned business value på y-axeln och antalet sprintar på x-axeln. Ju mer objekt från product backlog som blir färdigt, desto högre Earned business value kommer grafen att visa (Broadus 2013, s. 53).

*Användningsområden*
Earned business value ger ett informationsunderlag till flera intressenter. Både ledningen och kunden kan få information som kan hjälpa dem att ta snabba beslut om hur projektet ska fortlöpa. Earned business value ger information till ledningen om hur mycket nytta som levererats hittills till kunden. Kunden får information om hur mycket värde de fått levererat. Andra fördelar kan uppnås genom att Earned business value kombineras med andra KPI:er, exempelvis Burn-up. Genom att kombinera dessa kan intressenter hålla sig uppdaterade om hur projektet fortlöper. Ytterligare en fördel är att ledningen kan observera när ett projekt uppvisar minskande avkastning, det vill säga då det börjar kosta mer att utveckla mjukvaran än värdet som mjukvaran skapar. Då bör mjukvaran gå in i en stabil fas inför leverans. Slutligen kan Earned business value användas för att jämföra olika projekt för att avgöra hur resurser bör allokeras och investeras optimalt (Rawsthorne 2010, s. 5).

*Förutsättningar och problem*
Problemen med Earned business value hör främst till svårigheten att definiera hur mycket värde som skapas i projektet och till kunden. Faktorer som exempelvis alternativkostnader måste vägas in, vilket är frågan om pengar kan investeras i något annat som ger högre avkastning. Hänsyn bör också tas till att immateriella tillgångar försvårar värderingen av värdeskapandet ytterligare. Exempelvis om en anställd på projektet väljer att avsluta sin anställning, då är en nyrekrytering till projektet en kostnad som måste räknas in (Balbes 2012).

### 2.5.4 Sammanställning

En sammanställning över användnings- och problemområden för de KPI:er som har berörts i litteraturstudien presenteras i tabell 1 nedan.

*Tabell 1 Sammanställning över KPI:erna i det teoretiska ramverket*

| **KPI** | **Område** | **Användningsområden** | **Problemområden** |
|---|---|---|---|
| Sprint burn-down | Planering | ● Följa arbetsutvecklingen kortsiktigt<br>● Upptäcka problem<br>● Motivera medarbetare | ● Definition av avklarat arbete<br>● Hårt ställda krav |
| Velocity | | ● Öka förutsägbarhet<br>● Mäta produktivitet<br>● Focus factor | ● Icke schemalagt arbete inte inkluderas<br>● Optimistiska estimeringar<br>● Technical debt<br>● Svårigheter i estimering |
| Burn-up | | ● Projektuppföljning på hög nivå<br>● Överblick över adderat arbete | ● Nyttor gentemot Burn-down förloras vid ett bestämt omfång<br>● Svårigheter i tolkning<br>. |
| Happiness | | ● Förbättra psykiskt och fysisk arbetsmiljö<br>● Förbättra produktivitet<br>● Planering och allokering | ● Behov av frekventa mätningar<br>● Behov av uppföljning<br>● Koppling mellan lycka och förbättring |
| Defect count | Kvalitet | ● Mäta kvalitet<br>● Planera projekt<br>● Motivera medarbetarna | ● Behov av uttömmande testning<br>● Hård styrning<br>● Avsaknad av allvarlighetsgrad |
| Bug correction time | | ● Planera projekt<br>● Prioritera arbete | ● Svårigheter i tidsuppskattning<br>● Behov av lätthanterliga verktyg<br>● Utvärdering av individer |
| Technical debt | | ● Säkerställa framtida kvalitet och effektivitet | ● Svårigheter i att motivera kostnader<br>● Svårigheter i att kvantifiera |
| Net promoter score | Kund-värde | ● Mäta kundlojalitet<br>● Motiverar medarbetarna | ● Behov av kompletterande frågor<br>● Kulturella skillnader<br>● Behov av viktning |
| Earned business value | | ● Underlätta beslutsfattande<br>● Prioritera arbete | ● Svårigheter i att definiera värde |

# 3. Metod

I detta avsnitt förklaras metoden som använts i kandidatarbetet, det vill säga hur processen såg ut för att frågeställningarna skulle besvaras. Fokus ligger på utförlighet, motivering och kritisk granskning för att intressenter ska kunna förstå utförandet samt bilda sig en uppfattning om resultatets validitet och reliabilitet. Metoden innehåller delkapitlen forskningsansats, planering av studien, litteraturstudie, empirisk datainsamling, analys av insamlad data och metodkritik.

## 3.1 Forskningsansats

Bakgrunden till studiens uppkomst är en diskussion mellan Delphi och två av studiens gruppmedlemmar. Delphis syfte med studien är att skapa en större förståelse för ämnesområdet snarare än att lösa ett specifikt problem. Av den anledningen kommer rapporten i huvudsak att vara av utredande karaktär. Detta är i enlighet med studiens syfte, det vill säga att kartlägga Delphis styrning av ett projekt inom agil utveckling, med avseende på KPI:er och dess effekter på projektstyrning, samt lämna rekommendationer utifrån rådande forskning.

Undersökningen har genomförts som en fallstudie, vilket har flera fördelar. Däribland förbättrade möjligheter till konkretisering. Med tanke på att företaget som har studerats också är uppdragsgivare var det viktigt att finna slutsatser som berör det specifika fallet, snarare än strävan att göra framstående, allmängiltiga framsteg i forskningen kring KPI:er inom agil utveckling. Därför har fallstudien främst fokuserat på KPI:er som är intressanta för Delphi och filtrerat bland de KPI:er som finns tillgängliga i litteraturen.

Efter att syfte och frågeställningar fastställts undersöktes vilken forskningsansats som bäst överensstämmer med kandidatarbetets inriktning. Valet styrde hur metoden utformades. Forskningsansatsen för detta kandidatarbete följer en abduktiv ansats, med andra ord en systematisk kombination av olika delar i forskningsprocessen. Denna går ut på att arbeta med teori och empiri parallellt under studiens gång för att komma fram till beslutsunderlag och rekommendationer i enlighet med kandidatarbetets frågeställningar (Le Duc 2011a).

## 3.2 Planering av studien

Planeringen av studien utfördes efter det att syftet fastställts. Planeringen utgjordes av en tidsplan och en planeringsrapport. Tidsplanen ämnade utgöra en översikt av studiens arbetsgång med milstolpar tillsammans med ett Gantt-schema. Planeringsrapporten redogjorde huvudsakligen för studiens syfte och problembild. Den behandlade även delar såsom avgränsningar, datainsamling och metod. Samtliga delar av planeringsarbetet validerades i samband med handledare på Chalmers och Delphi.

## 3.3 Litteraturstudie

Det teoretiska ramverket utgjordes av en litteraturstudie. Litteraturen har huvudsakligen inhämtats genom Chalmers biblioteks hemsida. Förutom dessa vetenskapliga publikationer har även artiklar, böcker och hemsidor använts som är skrivna av personer som är verksamma inom branschen. Hänsyn har tagits till eventuella subjektiva influenser. Samtlig information har dock genomgått en utvärdering av trovärdighet tillsammans med författarnas kompetens och erfarenhet. Sökorden som användes var bland annat: *agile metrics, Scrum, KPI, software development* och *sustainability*.

## 3.4 Empirisk datainsamling

För att besvara frågeställningarna har det, utöver teori, krävts empirisk data. Insamling av data från Delphi har gjorts genom intervjuer. Den första fasen av intervjuer syftade till att kartlägga de KPI:er Delphi använder för projektet som ligger till grund för studien. Intervjuobjekt för den första delen var personer involverade i styrningen för projektet: Thomas Frenning som är Technical Manager för projektet och handledare åt kandidatgruppen, Matti Larborn som är Systems Lead, Ross Fitkin som är Engineering Group Manager samt Hans Lindau som är projektledare för mjukvaruutvecklingen inom IHU. Intervjuerna utfördes veckovis och var en mix mellan ostrukturerade och semi-strukturerade intervjuer. I de semistrukturerade var det bestämt på förhand vilka KPI:er som skulle diskuteras. Ledningen valdes som intervjuobjekt i den första fasen eftersom de var handledare för projektet och eftersom de antogs ha en bättre överblick över styrningen än andra roller. När begreppet ledning nämns i rapporten syftar det till chefer och ledare inom projektet, alltså inte företagsledningen. Intervjumallen för den första fasen av intervjuer går att finna i bilaga 1.

Efter att relevanta KPI:er för projektet identifierats fördjupades diskussionerna med fokus på användningsområden, problem och förutsättningar. För att få en förståelse över ett potentiellt problems hela omfång har utöver Delphis ledning även Volvo Cars ledning samt team-medlemmar, varav en Scrum master och två utvecklare, varit intervjuobjekt. Dessa utvecklare och Scrum mastern kommer från olika team. Intervjuobjektet från Volvo Cars ledning är en anställd som har en roll i projektet som länk mellan Volvo Cars ledning och Delphis produktägare. Ingen produktägare har intervjuats, vilket beror på att intervjun med Volvo Cars ledning ansågs uppfylla samma syfte. Intervjuerna med ledningen ämnade klargöra deras mål med projektet för att kunna säkerställa att rätt problem behandlas. Vidare syftade intervjuerna med ledningen till att undersöka hur de styr projektet med hjälp av KPI:er.

*Tabell 2 Översikt över intervjuobjekt*

| **Antal personer** | **Roll** |
|---|---|
| 4 | Delphis ledning |
| 1 | Volvo Cars ledning |
| 1 | Scrum master |
| 2 | Utvecklare |

Den andra fasen av intervjuer var av semi-strukturerad natur. Här intervjuades två utvecklare, en Scrum master och en person från Volvo Cars ledning. Att det inte utfördes fler intervjuer med utvecklare berodde på en svårighet i att frilägga tid ifrån deras pressade scheman, alltså inte en bristande motivation eller motvilja mot att bli intervjuade. För att få en bredare syn på processen har intervjuobjekt med olika roller valts ut. Urvalet av intervjuobjekt hanterades dock av handledaren på Delphi. För att uppmuntra att intervjuobjekten svarar ärligt har intervjuerna varit anonyma. För att behålla anonymiteten i resten av arbetet kommer Scrum mastern att refereras till som en utvecklare och Volvo Cars ledning kommer ingå i gruppen ledning. En kvalitativ metod under intervjuerna möjliggjorde flexibilitet och utförligare svar. Helt strukturerade intervjuer tros inte kunna fånga upp eventuella suboptimeringar och dolda effekter som KPI:erna har på personal eller andra intressenter, eftersom det blir svårare att uttrycka sig fritt. Därför var en mer öppen intervjuform att föredra.

Syftet med intervjumallen sammanställdes i fem punkter:

- Identifiera ledningens syn på KPI:er
- Identifiera medarbetarnas syn på KPI:er
- Identifiera vad som uppnås med KPI:erna
- Identifiera eventuella problem med KPI:erna samt vad de får för konsekvenser
- Identifiera eventuella andra KPI:er som skulle kunna underlätta arbetet

Vid de semistrukturerade intervjutillfällena har tre intervjuare deltagit varav en ställt frågor och resterande två tagit anteckningar. För att underlätta senare analys har intervjuerna spelats in. Direkt efter intervjun behandlades intervjuanteckningarna och inspelningen för att undvika misstolkningar. Intervjumallen för den andra fasen av intervjuer går att finna i bilaga 2.

## 3.5 Analys av insamlad data

Efter att data hade samlats in jämfördes den med litteraturstudien. Intressanta skillnader och likheter diskuterades sedan för att kunna nå en slutsats om hur Delphi framöver bör styra sina agila mjukvaruprojekt.

## 3.6 Metodkritik

”Med validitet, dvs giltighet, menas mätinstrumentets förmåga att mäta det som det påstås mäta. Med reliabilitet, dvs pålitlighet, menas att en mätmetod är okänslig för slumpens inverkan. Reliabilitet är hög när testen ger samma resultat vid upprepade mätningar.” (Lundequist 1995, s. 38)

Valet av metod samt tillförlitligheten på insamlad data är viktig för resultatet samt dess användbarhet. Först och främst ställs krav på validiteten och reliabiliteten. Att säkerställa validiteten innebär säkerställandet av att det som avses mätas verkligen mäts. Reliabiliteten handlar istället om tillförlitligheten i den specifika mätningen (Le Duc 2011b).

Validiteten har säkerställts genom ständig avstämning med handledare, från Chalmers och Delphi, att rätt saker undersöks för att kunna uppnå syftet och besvara frågeställningarna. Genom att omvandla syftet i rapporten till syften i en intervjumall, och därefter basera samtliga frågor på denna, har risken för att fel saker undersöks i intervjuerna minimerats. Risken för brister i reliabiliteten har minskats genom ett flertal åtgärder som beskrivs nedan.

Tillförlitligheten i undersökt litteratur har upprätthållits på en hög nivå genom att använda primära källor i största möjliga utsträckning. Genom att använda sig av en kombination av flera olika källor blir förtroendet för informationen större. Detta minskar risken för att informationen blir subjektiv och enbart baserade på en källas uppfattning av verkligheten.

Potentiella problemområden med intervjuer är flera. Intervjuobjektens svar måste ifrågasättas angående tillförlitligheten, de kan exempelvis vara partiska. Anställda med ansvar för rapportering av KPI:er kan vara mindre villiga att berätta om problem och suboptimeringar. En utvecklare på Delphi kan tänkas vilja påverka arbetet med KPI:er efter en personlig agenda, såsom att frånsäga sig ansvar. Ledningen å sin sida kanske vill få det att verka som att Scrum följs mer noggrant än vad som faktiskt sker. Vidare kan individer ha varierande förmåga att uttrycka sig i tal och olika intresse för att ställa upp. Ett hjälpmedel för detta är att bygga upp motivation genom att tydligt beskriva syftet med studien och intervjuerna. Även relationen mellan intervjuare och intervjuobjekt kan påverka den data som samlas in. För att öka reliabiliteten för intervjuerna har samma person ställt frågorna under samtliga intervjuer, förutom vid ett tillfälle då det inte var möjligt, för att på så sätt minska variationen i hur frågorna ställs. Två faktorer att ha i åtanke är antalet intervjuer och urvalet av intervjuobjekt. Antalet begränsades av tillgängligheten på Delphi och urvalet gjordes med hjälp av handledaren på Delphi. Båda dessa faktorer bidrar till en sämre reliabilitet.

Nackdelar med fallstudier är exempelvis att det är svårt att generalisera resultatet, räckvidden är begränsad, enskilda aktörers roll kan överbetonas och olika aktörers beteenden kan isoleras. För att kunna få en komplett överblick över hur Delphi arbetar med Scrum skulle alla aktörer behövts intervjuas. Under projektets period fanns det inte utrymme för detta, varför det måste finnas i åtanke att endast åtta intervjuobjekt har behandlats. I detta specifika arbete är dock fallstudie som metod motiverat, då generalisering och räckvidd inte är ett övergripande mål. Även överbetoningen och isoleringen av enskilda aktörers roller och beteenden har liten påverkan på studiens syfte. Om resultatet skulle användas i andra sammanhang än för det i studien blir dessa nackdelar mer betydande och bör hållas i åtanke.

# 4. Empiriska resultat

Här presenteras information från intervjuerna med anställda på Delphi och Volvo Cars. Informationen presenteras utifrån om den kommer från ledningen eller från utvecklare, det vill säga att ingen särskiljning görs mellan Delphi och Volvo Cars. Anledningen är att det inte ligger inom denna rapports fokus att identifiera skillnader dem emellan. I slutet av kapitlet sammanställs resultatet i tabell 3. Syftet med den insamlade informationen är att besvara den andra frågeställningen:

***2. Hur styr Delphi det aktuella agila mjukvaruutvecklingsprojektet med KPI:er?***

Under inledande intervjuer med ledningen identifierades sex centrala KPI:er som är de viktigaste och mest använda inom projektet. Delphis KPI:er baseras på erfarenhet från tidigare projekt. Delphi använder således ingen modell för framtagning av KPI:er, varför de sex KPI:erna inte kommer att grupperas efter mål likt det teoretiska ramverket i kapitel 2. Istället presenteras dessa var för sig med en beskrivning av användningsområden samt eventuella problem. Det område som i teorikapitlet presenterades som Burn-up är i detta kapitel uppdelat på de två skilda KPI:erna Epic burn-up och Requirements burn-up. Detta eftersom Delphi använder KPI:erna på olika sätt och eftersom de har olika problemområden. Avslutningsvis följer kommentarer från intervjuerna på kompletterande KPI:er som har identifierats under litteraturstudien. Det område som presenterades som Earned business value och NPS i teoridelen kommer att slås samman i den här delen under rubriken Kundvärde. Anledningen är att den information som framkom angående KPI:erna i detta avsnitt är gemensam.

## 4.1 Sprint burn-down

Enligt ledningen mäter Sprint burn-down sprintarnas fortgång hos Delphi. Enheten för denna KPI är story points. Rapporteringen av den data som ligger till grund för KPI:n sker genom rapporteringsverktyget JIRA direkt när arbetsuppgifter har avklarats. Inledningsvis i projektet finns en koppling mellan en story point och traditionella tidsmått. Anledningen är att underlätta för medarbetare som är ovana vid Scrum-metodiken kring tidsuppskattning av arbetsuppgifter. Allt eftersom projektet fortgår släpps denna översättning från story points till tid då utvecklingsteamen arbetar och hanterar sin estimering enligt olika principer.[1] En annan del av ledningen använder KPI:n en gång i veckan, och enbart på team som ledningen vet har en leverans.[2]

Enligt en intervjuad person i ett utvecklingsteam är teamets kapacitet cirka 2,3 story points om dagen. Siffran är specifik för deras team och de jämför inte den med andra team. Anledningen till detta är att olika team har olika estimeringstekniker. Teamet kom fram till 2,3 genom att de tog en "baseline-story" och estimerade den till fem story points. Därefter poängsattes nästkommande stories i relation till denna "baseline". Den intervjuade teammedlemmen menar att de inte använder dagar eller timmar internt i deras team, men att det förekommer konvertering av story points till dagar och timmar. Deras träffsäkerhet i estimeringen har varit bra så de har inte behövt vidta några åtgärder för att ändra på sin "baseline-story". Det finns för detta projekt inte några riktlinjer gällande hur felaktig ens träffsäkerhet får vara.[3]

[1] Ledning. 2016. Intervju 15 februari
[2] Ledning. 2016. Intervju 29 april
[3] Utvecklare C. 2016. Intervju 14 april

En annan intervjuad medlem i ett utvecklingsteam menar att en story point har en relation till absolut tid och inte till komplexitet. En story point likställs i dagens läge med en dags arbete. Det förekommer dock diskussioner internt i teamet kring att istället byta till story points som är baserade på komplexitet. Rapporteringen av stories varierar mellan person till person. Vissa rapporterar in direkt när arbetet är klart, medan andra personer väntar till slutet av dagen för att rapportera in dagens avklarade arbete.[4]

### 4.1.1 Användningsområden

För en del av ledningen är syftet med Sprint burn-down att få kontroll över projektets omfång och att säkerställa daglig utveckling för att uppnå projektets mål. Ledningen menar att KPI:n ämnar besvara huruvida projektet kommer att bli färdigt i tid. Vidare menar ledningen att Sprint burn-down är av intresse för utvecklingsteamen som ett dagligt hjälpmedel för att se till att de ligger i fas, samt för Delphis ledning för att säkerställa att målen kommer att uppnås.[5]

En annan del av ledningen som intervjuades använder KPI:n eftersom den ger bra transparens och framförhållning. I intervjun nämner den intervjuade "Inte bra om man samma dag som leveransen inser att det kommer levereras sju epics istället för nio". Om avvikelser mot det som utvecklingsteamet tagit på sig att göra uppkommer behöver en diskussion föras så att anledningen till avvikelsen kan identifieras. Slutligen menar den intervjuade att det är viktigare att övergripande mål med sprinten uppnås än att alla enskilda user stories blir avklarad.[2]

I en intervju med en utvecklare framgick det att den intervjuade tittar på grafen över Sprint burn-down tre till sex gånger per dag, huvudsakligen för att se hur teamet ligger till i förhållande till sin planering. Teamet kollar även på sprint burn-down i varje daily stand-up meeting. Sprint burn-down är den KPI som den intervjuade tycker bäst om. Rapporteringen av KPI:n sker automatiskt då stories klarmarkeras, vilket skapar en enkel och snabb rapportering. KPI:n är dessutom överskådlig och tydlig, samtidigt som effekter kan ses redan på den andra dagen av en tvåveckors sprint. Om sprint burn-down visar att projektet halkar efter så analyseras anledningen till problemet genom en mängd varför-frågor. På så sätt kan eventuella handlingar identifieras som skulle kunna motverka problemen.[3]

En annan intervjuad utvecklare läser av Sprint burn-down en gång per dag, eller ibland varannan. Anledningen till detta är att skapa en förståelse kring hur de ligger till i sprinten. Beroende på teamets status kan produktägaren behöva kontaktas för att försöka förbättra effektiviteten.[4]

### 4.1.2 Förutsättningar och problem

Ett problemområde för Delphi i samband med Sprint burn-down kan, enligt ledningen, vara att tala i story points, samtidigt som "en poäng" är ett oklart uttryck. Story points kopplas sedan till tidsenheten timmar, och även här råder det oklarheter då det kan vara svårt att avgöra om det är ideella timmar eller verkliga. Ytterligare problem är att det förekommer sidoaktiviteter som inte mäts i story points. En fråga som behöver besvaras är vilka aktiviteter som ska räknas med. Styrningen utifrån KPI:n är komplicerad då den inte alltid är rättvisande och kan uppvisa ett "hockey-stick"-mönster, beroende på att framsteg först rapporteras in när arbetet är klart. Detta leder till att kurvan visar på att projektet inte kommer hinna nå målet inledningsvis.[1]

[4] Utvecklare A. 2016. Intervju 29 april
[5] Ledning. 2016. Intervju 15 februari

Ett problem kan, enligt en annan del av ledningen, uppstå om ledningen börjar gå in och försöka påverka avvikande nivåer utan att först föra en diskussion om varför de uppkommit. Det är vanligt att företagsledningen inte har koll på varför mjukvaru-KPI:er egentligen används, och de väljer att använda avvikelser i resultat som prestationsmått för teamen.[2]

I en intervju med en utvecklare framgick det att mindre än 10 % av arbetet inte är fångat av story points. Den låga andelen medför att arbete utan story points inte påverkar planeringen i större utsträckning. Det mesta av dessa uppgifter kommer från det dagliga arbetet likt att planera möten, förbereda desamma och eventuella tester som genomförs sista dagen på sprinten när arbetet med sprinten redan är avslutat.[3]

Enligt en annan intervjuad utvecklare förs loggar över hur lång tid uppgifter tar. Vid senare planering kan detta användas som feedback för att förbättra estimaten. Story points mäts här alltid i timmar. Ett alternativ är att arbeta med estimering efter ansträngningsnivå. Problemet uppges vara medarbetarnas varierande kompetens och erfarenhet. Samma arbetsuppgifter är olika ansträngande för olika personer, varför måttet skulle kunna bli missvisande. Olika team hanterar dock sin estimering olika, vissa utifrån timmar och andra utifrån ansträngningsnivå. Teamen är i enstaka fall dynamiska och resurser kan flyttas däremellan, vilket gör att produktiviteten kan försämras om medarbetarna måste alternera mellan olika estimerings-tekniker. Angående arbete som inte är poängsatt bedömer utvecklaren att det skulle uppgå till närmare 50 %.[6]

Vidare fungerar inte alltid Sprint burn-down som tänkt. Konsekvensen blir att nyttan faller bort med KPI:n och att den därför inte används på utvecklarnivå. Under vissa perioder ligger grafen konstant, motsvarande att inget arbete utförs. Sanningen är dock att arbete faktiskt har utförts, men inte kunnat markeras som färdigt på grund av beroenden av andra team. Grundproblemet är att hela projektet är organiserat vågrätt medan måtten är vertikala och skär genom lagren. Lagren har i sin tur olika mognadsgrad, vilket gör att team som arbetar med samma funktion på olika nivåer kan blockera varandra. Exempelvis kan ett team ha ansvar för en uppgift, men fastän 95 % av arbetet är gjort kan uppgiften inte rapporteras som klar eftersom definitionen av klar innefattar moment som måste utföras av till exempel hårdvaruteamet. När chefer och projektledare läser graferna och tror att arbetet inte går framåt uppstår nervositet och utvecklarna pressas att arbeta mer. Utvecklarna arbetar redan efter full kraft, varifrån intervjupersonen drar slutsatsen “Bevis nog på att måtten man har inte funkar”.[6]

Anpassningar har gjorts som minskar problemet. När nya stories skapas läggs fokus på att hålla ned gränsytorna mot andra team, i syfte att kunna färdigställa arbetsuppgifter inom ett team. Intervjupersonen uttryckte att det har gjorts ”Anpassning mot de [KPI:erna] som finns, men jag tycker inte de måtten som finns är bra”. Effektiviteten hämmas genom att onödig uppmärksamhet uppkommer över arbetsuppgifter. Chefer påpekar att vissa saker måste bli gjorda, varpå utvecklarna får förklara varför det inte är möjligt gång på gång. Påföljande risker är otrevnad, stress och dålig gruppstämning. Team som egentligen gör sitt jobb drabbas av att andra team inte utför sina uppgifter. Intervjupersonen säger “Jag på utvecklarnivå ska inte behöva hantera projektets tillkortakommanden, det får projektet fixa.”. Ett alternativ som kan övervägas är att se över “definition of done” för att tidigare kunna bocka av det arbete som gjorts, varpå andra team kan ta över enligt sin egen prioritering.[6]

[6] Utvecklare B. 2016. Intervju 22 april

Av allt arbete menar en intervjuad utvecklare, från ett annat team, att cirka 60-70 procent av arbetet mäts av story points. Det arbete som är poängsatt prioriteras även över icke poängsatt. Personen sade att ”Av ren transparens är det bättre, dels för teamet men även för produktägaren. Något som inte bokförs syns inte.”. Utvecklaren ser även ett annat problem med Sprint burn-down, vilket personen benämner som ketchup-effekten. Problemet innebär att projektet, till synes, går långsamt i början men mot slutet kommer ketchup-effekten och en stor del av arbetet registreras som slutfört. Konsekvenserna blir att folk eldar på i onödan, när det egentligen bara krävs lite tid att komma över tröskeln. Detta kan dock vara svårt för folk att förstå. Den intervjuade ser även ett problem med definitionen av en story point, personen sade att ”Det finns ett problem med att vi använder absolut tid för story points istället för komplexitet. Det blir väldigt svårt att få en stabil hastighet och en jämförelse mellan två olika stories”.[4]

## 4.2 Velocity

Velocity definieras av ledningen som antalet story points som färdigställts under en sprint. Inrapporteringen av story points ska ske direkt när teamen gjort klart något, på samma sätt som beskrivet för Sprint burn-down. En sprint är två veckor lång, varför grafen uppdateras varannan vecka.[7]

### 4.2.1 Användningsområden

Enligt ledningen mäts Velocity för att teamen ska bli bättre på att estimera hur många story points de kan klara av och sedan planera in dessa i nästa sprint. Detta görs varannan vecka innan teamen åtar sig nytt arbete. Andra intressenter än teamet är projektledare, som har i uppgift att reagera om teamen inte når upp till sina estimat.[7]

Att ledningen reagerar om teamen inte når upp till sina estimat framgick även under en annan intervju med ledningen, i vilken personen sade “När det börjar gå dåligt flera leveranser i rad börjar jag gå in och kolla på detalj och se vad vi kan göra för då kanske vi har överbelastat ett visst team för mycket”.[2]

En utvecklare berättade i en intervju att han kollar på Velocity varannan eller var tredje dag. Personen sade ”It is one of the main tools we are using and we use it all the time”. I huvudsak motiveras detta baserat på att det är den viktigaste KPI:n med syftet ständiga förbättringar. KPI:n möjliggör en förståelse kring var teamet är på väg och vad de har för möjligheter. Velocity används främst på en längre horisont, över cirka tre sprintar, för att på så sätt gestalta en utveckling. Vidare kollar de även på förra sprintens Velocity när de ska planera nästa sprint. Är det något som är oklart, till exempel att Velocity var högre eller lägre än förväntat, sätter sig teamet ner och analyserar orsaken bakom avvikelsen.[3]

En annan utvecklare som intervjuats använder i dagsläget inte Velocity för att planera hur mycket arbete teamet ska ta på sig till nästa sprint. Andra team gör dock detta och det är något som personens team ska börja med. Istället används Velocity på en mer övergripande nivå för att se hur teamet ligger till, för att få sprinten att gå i mål samt för att uppnå det som teamet tagit på sig. Avläsningen sker varje, eller varannan, dag och värdet jämförs inte med andra teams Velocity.[4]

[7] Ledning. 2016. Intervju 22 februari

### 4.2.2 Förutsättningar och problem

Ett problem är att om ledningen går in och klagar över att teamen inte når upp till sina estimat kan teamen komma att börja mäta på ett annat sätt. De kan bli mer generösa med hur många story points något är värt.[7]

I en intervju med en utvecklare framgick det att Velocity inte används som ett prestationsmått. Enligt den intervjuade personen kan det bli problem med Velocity om perfektion eftersträvas. Om detta var fallet menar personen att KPI:n vore meningslös. Personen sade ”It is a good indicator if it is used in the right way”. Det framgick även att den intervjuade personen inte vet om ledningen kollar på teamets Velocity och att personen inte bryr sig om ifall de i själva verket gör det. Teamet är stolta över sin Velocity och har ett driv till att försöka förbättra den. Velocity fungerar då istället som ett verktyg för teamet att utveckla sin egen produktivitet. Den intervjuade tror inte heller att ledningen är intresserade av teamets Velocity, men visar gärna upp trenden på Velocity som speglar deras utveckling.[3]

Utvecklaren har lång erfarenhet av den agila arbetsmetodiken och gillar inte idén med att team jämförs. Personen sade “Det är inte rätt att skapa konkurrens mellan team utan de ska fokusera på sitt eget arbete och göra sitt bästa.”. Det handlar inte om konkurrens för att driva produktiviteten utan att hjälpas åt, personen kommenterade ”We are sharing information, but not comparing”. Intervjupersonen berättade också att de har haft lite problematik under de tolv sprintar teamet har varit aktiva i. Nya resurser har i vissa fall tillkommit och i andra fall har andra resurser fallit bort, vilket har haft påverkan på Velocity.[3]

En annan utvecklare som intervjuats bekräftar att Velocity används som prestationsmått. Det är ett viktigt index för ledningen, vilka blir nervösa om nivåerna är låga. Intervjupersonen menar att ledningen då inte funderar över om de själva gjort något fel, utan direkt skickar problemet vidare till lägre nivå. ” ‘Åh fy fan, nu har vi varit handlingskraftiga’, så går de och tar en kaffe” efterliknar utvecklaren cheferna med viss tillspetsning. Utifrån dagens förutsättningar använder inte utvecklaren KPI:n på grund av dess brister. I likhet med Sprint burn-down bygger bristerna på att stories inte kan bockas av på grund av beroenden med andra team. Velocity stämmer då inte med verkligheten. ”Om de inte visar vad som händer, då ser jag inget värde med dem” uppger den intervjuade. Koppling till både kundnytta och arbetsinsats saknas, vilket gör att det bara blir en siffra utan stöd i vad som verkligen händer. Intervjupersonen är alltid med och förfinar planeringen inför varje sprint, men när den väl är satt plockas uppgifter från product backlog, alternativt så utförs uppgifter som inte blev färdiga förra sprinten, utan fokus på mätningar under arbetets gång.[6]

Angående om Velocity används som ett prestationsmått svarade en tredje utvecklare att ”Det blir det ju. Säga vad man vill men Velocity används av många. Chefer tittar på den tillslut, produktägare tittar på den av ren överlevnadsinstinkt för att se var man ligger till.”. Ledningen har dessutom ett intresse av teamets Velocity. Anledningen är att teamet ska leverera en viss funktionalitet, vilket säkerställs av ledningen genom att de studerar teamens Velocity och analyserar om allt kommer att bli klart i tid. Det förekommer även en indirekt press från ledningen på att teamen ska uppnå en hög Velocity.[4]

## 4.3 Epic burn-up

Delphi använder Epic burn-up för att mäta avklarandet av arbetspaket, så kallade ”epics”. KPI:n bygger på samma inrapportering som Sprint burn-down. Skillnaden är att i Epic burn-up aggregeras story points till en högre nivå och mäts i arbetspaket. Processen för rapporteringen sker genom “drag and drop” i verktyget JIRA, vari teamen enkelt kan dra arbetspaket från en status till en annan och på så sätt markera att ett visst arbete har utförts. KPI:n och projekt-statusen presenteras sedan varje vecka i “chief-review meetings”.[1] Att mätningarna inte presenteras oftare än en gång i veckan beror på att KPI:n utvecklas relativt långsamt, med anledning av de stora arbetspaketen.[2]

### 4.3.1 Användningsområden

Epic burn-up används, enligt ledningen, främst ur ett projektledningsperspektiv för att klargöra hur helheten fortlöper och utvecklas. Ett exempel på detta kan vara: “Arbetar vi i en takt som är tillräckligt hög för att nå framtida krav?". En annan fördel med att mäta KPI:n är att det går att se vilka team som är klara med specifika delar, vilket innebär att Delphi får möjlighet att omfördela arbetskraft om så behövs. Projektanställda på ledningsnivå som är intresserade av denna KPI är Delphis interna och globala ledning samt Volvo Cars ledning för projektet. På lägre nivå kan produktägare och utvecklarteamen även vara intresserade av KPI:n.[1]

Tillsammans med en god kommunikation mellan ledning och team ger Epic burn-up, enligt en annan del av ledningen, en referenspunkt för att se hur projektet fortlöper. Det används för att veta hur Delphi ligger till i förhållande till nästa leverans. Delphi kommer överens med kunden om vissa epics inför en leverans, varav 85 % av dessa måste färdigställas enligt avtalet.[2]

Från en intervju med en av Delphis utvecklare framgår det att Epic burn-up inte är relevant i dennes vardag. Fokus är snarare mer kortsiktigt och är inriktat på vad som ska göras i nästa sprint och vilka epics som behandlas. Att se över hela projektets status genom Epic burn-up skulle snarare ligga på produktägarens agenda enligt intervjupersonen.[6]

Även under en intervju med en utvecklare inom ett annat team framgick det att KPI:er relaterade till epics är mer av intresse för en produktägare. Den intervjuade personen menar att KPI:n inte påverkar dennes situation, utan att intresset snarare ligger i det dagliga arbetet med dess tillhörande KPI:er. Epic burn-up uppges dock kunna vara användbar i syftet att förbereda sig inför framtiden. Intervjupersonen tittar på Epic burn-up cirka en gång per vecka, eller eventuellt oftare om milstolpar börjar närma sig.[3]

### 4.3.2 Förutsättningar och problem

Problem som Delphi upplever med Epic burn-up är, enligt en intervju med ledningen, att tolkningen ibland är svår och att det inte finns plats för klargörande kommentarer. På samma sätt som det finns svårigheter i att klarmarkera stories på grund av beroenden av andra team, som beskrevs i kapitlet om Sprint burn-down, finns också motsvarande problem för epics.[1]

En annan intervjuad anställd på ledningsnivå uppger att KPI:n inte får användas isolerat utan måste brukas tillsammans med en god kommunikation mellan ledning och team för att ge en bra uppfattning om statusen för projektet. “Ingen KPI får enskilt följas slaviskt. Den säger inte hela sanningen”.[2]

Ett annat problem med Epic burn-up framgick under en intervju med en tredje utvecklare. Personen förklarade att stora och långa projekt, med många epics, medför att träffsäkerheten inte är särskilt stor i början. Allt eftersom blir dock träffsäkerheten bättre och KPI:n viktigare.[3]

## 4.4 Requirements burn-up

Delphi mäter Requirements burn-up för bearbetningen av krav, vilket innebär de egenskaper och funktioner som Delphi kommit överens med Volvo Cars om enligt upphandlingen. Utöver bearbetningen av krav så är mätningar av antalet nya krav och förändringar av existerande krav ett användningsområde av intresse för Delphi. Inrapportering av KPI:n sker genom att teamen, i ett system, markerar kraven som implementerade. Detta exporteras sedan en gång i veckan för att ingå i en graf som visas upp på möten med ledningen och systemansvariga.[8] I samtliga intervjuer med utvecklare framgick det att deras kunskap kring Requirements burn-up är låg och att de inte använder KPI:n.[3] [4] [6]

### 4.4.1 Användningsområden

Ledningen på Delphi menar att Requirements burn-up mäts för att få koll på projektets omfång och status samt för att hitta gap. Gap visar huruvida det finns krav som inte är kopplade till någon epic eller komponent. KPI:ns intressenter är teammedlemmar, testare, projektledare och även högsta ledningen. Testare behöver kraven för att kunna bedriva sin testprocess. Ledningen vill se tillväxten och de ställer sig frågor som "är man vid 100 % eller inte?".[8]

### 4.4.2 Förutsättningar och problem

En förutsättning för användningen av KPI:n är, enligt intervjuer med ledningen, att teamen faktiskt matar in information och rapporterar att saker är färdiga. Utan information blir mätningar omöjliga, eller bristfälliga och missvisande. Inrapportering sker i ett annat system än JIRA, vilket utgör ett problem då det hela ses som extraarbete. Teamen tycker att de redan rapporterat in samma sak när de klarat av och rapporterat in sina epics eller arbetspaket i JIRA. Detta leder till att teamen exempelvis försöker avleda arbetsbördan till någon annan eller skjuta upp det till nästa vecka. Vidare är ledningens uppfattning att teamen tycker att det är tråkigt, samtidigt som kravkurvan blir hackig om ledningen pressar utvecklarna att rapportera in.[8]

Att projektet omfattar 10 000 krav utgör även det ett problem, då det kräver mycket jobb och blir svårt att följa upp alla krav. Dessa krav är i sin tur förknippade med en översättnings-osäkerhet vid överföring mellan olika system.[8]

Ett ytterligare problem är att när en defekt rapporteras så behöver det nödvändigtvis inte innebära något negativt, det kan vara så att kraven behöver förändras. En annan problematisk aspekt relaterad till KPI:n är att Volvo Cars ställer krav och förväntar sig att de ska uppfyllas. Sedan uppdaterar Volvo Cars inte alltid kraven i tillräckligt hög grad, vilket gör att dessa då blir mer som riktlinjer. Anledningen kan vara att det är tidskrävande för Volvo Cars att skriva om kraven. Projektets 10 000 krav är inte viktade i mängden jobb de representerar. Epics representerar en arbetsmängd och därför föredrar Delphi Epic burn-up över kravtillväxt.[8]

[8] Ledning. 2016. Intervju 9 mars

## 4.5 Defect count

Delphi mäter antalet öppna defekter i projektet och rapportering sker genom att teamen fyller i ett formulär med de defekter som inte är lösta efter en sprint. Olika fält måste fyllas i, bland annat allvarlighetsgrad. Defekter som uppkommit och kort därefter blivit lösta rapporteras och mäts därför aldrig.[9]

### 4.5.1 Användningsområden

Enligt ledningen används mätningar av Defect count för att säkerställa kvaliteten på produkten. Den ämnar även ge indikationer på huruvida Delphi uppfyller de krav som de åtagit sig samtidigt som oönskade kundeffekter motverkas. Utöver dessa intressenter så är även ledningen intresserade av att använda sig av KPI:n för att bilda sig en uppfattning om huruvida projektet kommer att nå sitt mål i tid. Utifrån informationen kan ledningen planera resurser för att stabilisera projektet.[9]

Enligt en intervju med en utvecklare har arbetet med defekter inte kommit igång än, på grund av projektets relativt tidiga stadium. I dagsläget kan det dyka upp defekter under mer informella tester med representanter från andra team, men dessa löses då direkt på plats. Intervjupersonen uppger att han inte känner till hur processen ser ut för att rapportera in en defekt.[6]

I en intervju med en annan utvecklare bekräftades det att det inte har uppstått så många defekter än. På grund av detta används inga KPI:er relaterade till defekter för tillfället, men det ska göras i framtiden.[4]

### 4.5.2 Förutsättningar och problem

En förutsättning för användningen av Defect count är, enligt ledningen, att antal defekter inte blir ett resultatmått. Defekter är en naturlig del i mjukvaruutveckling och Delphi bör istället fokusera på att uppfångningen och lösandet av defekterna hamnar i fokus. Ett problem i samband med KPI:n är att det inte alltid finns tillräckligt med dokumenterat underlag för att lösa defekterna. Även inrapporteringen av KPI:n genom ett formulär är förknippad med problem. Exempel på dessa problem är att teamen inte uppdaterar i tid, fyller i fel information eller så orkar de inte fylla i all nödvändig information. Delphi upplever att det är svårt att motivera teamen att fylla i formuläret riktigt. Andra problem rör synkningen mellan Delphi och Volvo Cars, vilket till exempel innebär att Volvo Cars rapporterar en defekt och blir oroliga om Delphi inte rapporterar att de kommer framåt i lösandet av denna. Detta kan leda till att ledningen jagar på teamen att lösa defekterna och att oönskade effekter, såsom stress, riskerar att uppstå.[9]

En annan aspekt som ledningen tar upp angående Defect count är hur anställda, som inte har tillräckligt mycket erfarenhet inom mjukvaruutveckling, reagerar på resultaten av KPI:n. “Har man inte mjukvarukunskap så tror jag inte man förstår vad defect innebär. Folk är livrädda för att ‘Oj, det finns 72 buggar.’ Låt det ligga 72 buggar, vad är problemet med dem här 72? Är det något som förstör för ditt system? ‘Nej.’ Låt det ligga en miljard sådana då om du vill. Det är teamet som ska prioritera.”.[2]

[9] Ledning. 2016. Intervju 29 februari

I en intervju med en utvecklare framgick det att personens team har stött på många defekter, beroende på att de har varit ett av de team som jobbat längst på projektet. De har dock inte använt några KPI:er relaterade till defekter, vilket intervjupersonen är lite orolig för. Anledningen till att det inte fungerar så bra uppges vara att det behövs tydligare riktlinjer och instruktioner om hanteringen av defekter.[3]

## 4.6 Bug correction time

Delphi mäter den genomsnittliga tiden det tar att stänga defekter. Enligt ledningen anses defekterna ha olika allvarlighetsgrad, vilket beror på en kombination av hur ofta defekten orsaker ett fel samt hur allvarligt detta fel är. Den allvarligaste klassificeringen är show stopper, vilket är den som får högst fokus.[9]

### 4.6.1 Användningsområden

Enligt ledningen behövs KPI:n för att kunna få en uppfattning om huruvida projektet kommer att nå sitt mål i tid. Den kan även användas för att planera och stabilisera projektet.[9]

Enligt en utvecklare har KPI:n inte börjat användas ännu. Detta är på grund av projektets tidiga stadium. Defekter har uppstått i mjukvaran men de har lösts under mer informella former, innan de nått kunden. Intervjupersonen känner inte till om det finns ett officiellt sätt att rapportera in defekter och i så fall hur det ska göras.[6]

En annan utvecklare är av samma åsikt som föregående utvecklare och berättar att det inte uppstått tillräckligt många defekter ännu för att några officiella KPI:er ska ha plockats fram.[4]

### 4.6.2 Förutsättningar och problem

Enligt ledningen är ett problem med KPI:n att det inte alltid finns tillräckligt med underlag för att lösa en bug. Det kan bero på otillräcklig dokumentation i rapporteringsprocessen och leder till att defekterna får vara kvar och bevakas. Detta i sin tur leder till en missgynnande Bug correction time eftersom defekterna är öppna längre än nödvändigt. Att rapportera utförligt och att ha väl fungerande testteam är en förutsättning för att KPI:n ska fungera bra.[9]

Ett annat problem som identifierats av ledningen är att den genomsnittliga tiden för att åtgärda defekter är högre för defekter av hög allvarlighetsgrad än vad den är för defekter med låg allvarlighetsgrad. Förhållandet borde egentligen vara det omvända.[9]

I en intervju med en utvecklare framkom det att det anses finnas otillräckligt med riktlinjer och instruktioner kring hur defekter bör hanteras för att en väl fungerande hantering ska kunna åstadkommas.[3]

## 4.7 Kompletterande KPI:er

Nedan presenteras intervjuobjektens åsikter angående tre KPI:er som inte används av Delphi i dagsläget, men som har berörts under litteraturstudien. Därför är intervjufrågorna av en mer öppen karaktär och den tidigare använda strukturen, med användningsområden och problem-områden, följs därmed inte.

### 4.7.1 Happiness

Medarbetarnöjdhet mäts, enligt ledningen, vartannat eller vart tredje år. Arbetet med Scrum har inte kommit tillräckligt långt ännu för att Happiness skulle kunna appliceras på ett bra sätt. Tyvärr kan det i vissa fall hända att de beslut som borde fattas av utvecklare ändå tas av ledningen. "I och med att de [utvecklarna] inte får vara med och fatta de rätta besluten nu är de antagligen konstant förbannade". [2]

Enligt en utvecklare har enstaka undersökningar gjorts på medarbetarnöjdhet men ingenting regelbundet, till exempel varje sprint. Happiness hade varit bra för att se hur stressnivåerna är inom ett team och hur samarbetet fungerar. Intervjupersonen menar att det är viktigt för medarbetare att känna sig avslappande och kunna ställa frågor. Det kan även finnas ett värde i att mäta och se över arbetstiderna, både på individ- och teamnivå. Teoretiskt sett kan någon tycka att det finns för lite att göra och går hem klockan tre varje dag, medan andra jobbar 50-60 timmar per vecka.[6]

En annan intervjuad utvecklare menar att en av de viktigaste sakerna att mäta med KPI:er är arbetarnas motivation. Personen sade att "team-motivation saknar jag faktiskt, den är viktig.". Mätningarna skulle kunna ske i form av ett formulär som fylls i någon gång i månaden. Det viktiga är att se till att folk mår bra och att det mäts. Happiness skulle enligt personen vara "... jättebra om man kunde införa den, en kanonidé.".[4]

### 4.7.2 Technical debt

Enligt ledningen är Technical debt ett viktigt område för att säkerställa kvaliteten på produkten och vore intressant att mäta. Problemet är att det inte finns något uttalat sätt att mäta detta på.[2]

Enligt en utvecklare försöker Technical debt undvikas genom statisk kodanalys. Vidare arbetar Delphi även med granskningar samt riktlinjer för formatering och vilka konstruktioner som bör användas. Arbetet med att ta fram enkel kod som är lätt att underhålla sker kontinuerligt och redan från projektets start.[6]

### 4.7.3 Kundvärde

Ledningen anser att de inte har något större incitament till att mäta kundvärde i det aktuella IHU-projektet. Anledningen är att projektet är av fastpriskaraktär, vilket innebär att all funktionalitet ska levereras oavsett värde. Om det inte hade varit ett fastprisprojekt skulle det dock ha varit mer intressant att mäta på kundvärde. När omfånget inte är bestämt kan det finnas en nytta i att se till att värde levereras kontinuerligt samt att rätt arbete prioriteras och utförs. Det kan vara så att värdet av en specifik funktionalitet inte motsvarar den arbetsmängd som krävs för dess framtagning.[7]

En annan del av ledningen menar att eftersom alla epics är prioriterade utifrån hur mycket kundvärde de levererar finns det en viss koppling till kundvärde i projektet ändå. I ett fullt agilt projekt hade eventuellt ett byte varit möjligt till ett mer flexibelt omfång. Leveransen avbryts då när kundvärdet för varje enskild sprint har blivit för lågt. Det är dock svårt eftersom det inte finns ett utpräglat sätt att mäta hur mycket en viss funktionalitet är värd för kunden. I dagsläget är det svårt att mäta hur mycket en viss typ av funktion används i bilarna, vilket försvårar processen.[2]

En intervjuad utvecklare anser att någon typ av indikator på Return on Investment hade varit användbar. Anledningen är att det inte alltid är helt enkelt att värdera och prioritera stories. En KPI som mäter kundvärde skulle underlätta produktägarens jobb med att prioritera product backlog. Bakgrunden anses vara att det är många stories som har beroenden mellan olika team. Det finns alltså stories som kan se ut att inte vara värda så mycket för ens egna team, men som i själva verket är kritiska för andra team. Det går inte heller att avgöra till 100 % vilka stories som påverkar andra team och vilka som inte gör det, men det existerar några tekniker som underlättar. För att balansera stories så att rätt värde hela tiden levereras behövs det alltså något som jämför hur mycket resurser som läggs ned kontra vilket resultatet blir. Det bör inte mätas i monetära termer, utan baseras på kundvärde.[3]

En annan utvecklare påpekar behovet av att testa funktionerna mot kunden. Det vill säga att de får pröva och uppge nöjdhet samt om det fungerar som tänkt. Det kan vara så att mjukvaran är fullt fungerande och enligt specifikation, men att slutkunden inte är nöjd. I det fallet är kvaliteten dålig. Om ingen hänsyn ägnas till kundnyttan riskerar mycket tid att läggas på komplicerade funktioner som i slutändan adderar lågt värde och bara används någon enstaka gång per år. Då kanske andra funktioner med betydligt högre synlighet inte hinns med. Kundvärdet bör alltså ligga till grund för prioritering. De funktioner som existerar har dessutom sålts in av anställda på en högre nivå i ett samarbete mellan Delphi och Volvo Cars. När Delphis utvecklare sedan sitter och tar fram funktionerna förstår de inte alltid nyttan med alla delar. “Vad ska vi med det här till?”. Funktioner kan då prioriteras ned av teamet i samråd med produktägaren, som ofta sitter på samma information. I efterhand kan ledningen komma att fråga varför funktionerna inte färdigställts. Samtidigt kan de dock inte förklara varför den specifika funktionen är viktig. Det sitter anställda från Volvo Cars sida på plats hos Delphi, men det är inte de som är beställarna. Det är helt enkelt för många mellanhänder mellan beställarna och Delphis utvecklare. Intervjupersonen menar att den kommunikationen måste in och att ”Kundnyttan måste vara huvudmålet med det hela.”. Mer konkret kan kundvärde mätas genom poängsättningssystem, vari kunderna exempelvis får avgöra vad som kan undvikas och vad som är viktigt.[6]

## 4.8 Sammanställning

En sammanställning över användnings- och problemområden för de KPI:er som har berörts i empirin presenteras i tabell 3 nedan.

*Tabell 3 Sammanställning över de empiriska resultaten*

| **KPI** | **Användningsområden** | **Problemområden** |
|---|---|---|
| Sprint burn-down | ● Följa projektutveckling på låg nivå<br>● Utvärdera planering<br>● Upptäcka problem | ● Definition av avklarat arbete<br>● Definition och värdering av en story point<br>● Arbete som inte är poängsatt<br>● Beroenden för klarmarkering<br>● Sen rapportering |
| Velocity | ● Förbättra estimering<br>● Uppnå ständiga förbättringar<br>● Identifiera framsteg och möjligheter | ● Definition av avklarat arbete<br>● Press på estimatuppfyllelse<br>● Användning som prestationsmått<br>● Koppling till kundnytta<br>● Bristande användning<br>● Arbete som inte är poängsatt |
| Epic burn-up | ● Följa projektutveckling på hög nivå<br>● Se status inför nästa leverans | ● Svårigheter i tolkning<br>● Beroenden av andra team för klarmarkering<br>● Låg tillförlitlighet initialt |
| Requirements burn-up | ● Kontrollera omfång<br>● Följa projektutveckling | ● Svårigheter i att motivera inrapportering<br>● Mängden krav<br>● Kopplingen till arbetsmängd |
| Defect count | ● Säkerställa kvalitet<br>● Uppfylla kundkrav<br>● Planera projekt | ● Otillräckligt med dokumentation<br>● Bristande riktlinjer kring rapportering<br>● Användning som resultatmått |
| Bug correction time | ● Planera projekt | ● Otillräckligt med dokumentation<br>● Bristande riktlinjer kring rapportering |

# 5. Analys och diskussion

Detta kapitel jämför det teoretiska ramverket med de empiriska resultaten. Nedan följer de identifierade likheterna och skillnaderna för respektive KPI samt vad konsekvenserna blir. KPI:erna presenteras tillsammans med tabeller som sammanfattar skillnaderna inom användnings- och problemområden mellan teori och empiri. KPI:erna är arrangerade under de övergripande mål som de ämnar uppfylla, vilka är planering, kvalitet och kundvärde. Detta eftersom det ger möjlighet att faktorisera ut gemensamma användnings- och problemområden för det övergripande området. Att strukturen är samma som i kapitel 2, Teoretiskt ramverk, möjliggör även en tydlig jämförelse. Avsnittet besvarar den tredje frågeställningen i rapporten:

***3. Hur förhåller sig Delphis projektstyrning med KPI:er till uppfattningar i litteraturen och vad får det för konsekvenser?***

## 5.1 Modeller

En övergripande skillnad mellan litteraturen och empirin är att Delphi inte använder sig av någon modell för framtagning av KPI:er. I litteraturen rekommenderas att GQM eller PSM ska användas. I GQM tas KPI:er fram genom att först identifiera mål och sedan frågor som besvarar målen. KPI:erna i PSM bygger på bästa praxis.

Att Delphi inte använder sig av en modell för framtagning av KPI:er i dagsläget ger en rad olika följder. Om de KPI:er som tagits fram inte har utgått från företagets övergripande mål finns risk att för mycket eller för lite data samlas in. Detta leder, enligt litteraturen, ofta till misstro mot mätningarna. En sådan misstro finns bland utvecklare på Delphi dagsläget och det skulle kunna vara en konsekvens av att mätningarna inte är tillräckligt tydligt kopplade till företagsmål. Dessutom upplever samtliga intervjuade utvecklare att det finns saker som borde mätas men som inte blir mätta idag. Även det kan vara en konsekvens av att det inte finns någon modell som används.

## 5.2 Planering

Planering innefattar de KPI:er som används för att hjälpa till i planeringsarbetet. De tre KPI:erna Sprint burn-down, Velocity och Burn-up tillhör praxis inom mjukvaruutveckling för Scrum. Trots att de tillhör standarden inom industrin finns det skillnader i hur de hanteras och vilka effekterna blir, såväl mellan företag som inom företag. Det empiriska resultatet, visar på variationen i användandet av KPI:erna och den upplevda nyttan. Några ser ett större värde i att följa upp projektutvecklingen med hjälp av KPI:erna och ser över status flera gånger i veckan, och i vissa fall även flera gånger om dagen. I andra fall, till exempel enligt en intervju med en utvecklare, används KPI:erna inte över huvud taget på grund av de upplevda bristerna i att visa vad som sker i verkligheten. Gemensamt för Sprint burn-down, Velocity och Epic burn-up är att mätningarna grundas i story points.

Användningen av story points är unik för Scrum, jämfört med exempelvis den traditionella vattenfallsmodellen. Hård press från ledningen på att Sprint burn-down ska se bra ut, eller att Velocity ska vara hög, riskerar att skapa oönskade effekter. För att motivera fortsatt användande av KPI:erna inom projektet är det av vikt för ledningen att hantera denna situation. De huvudsakliga brister som identifierats är otillräcklig kommunikation och en arbetsprocess som ger missvisande information till mätningarna. De starka beroendena mellan de olika teamen, vad gäller klarmarkering av stories och epics, förskjuter inrapporteringen och ger hack i graferna över framsteg i projektet. Enligt intervjuerna med ledningen finns det en medvetenhet om ”hockey-stick”-mönstret för Sprint burn-down, det vill säga att lite arbete rapporterats in initialt och att mycket kommer mot slutet. Samtidigt upplever dock utvecklare hård press och uppmärksamhet från ledningen med statusförfrågningar. Medarbetarna riskerar att uppleva förhöjda stressnivåer, vilket inte är hållbart i längden. Dessutom kan den ekonomiska hållbarheten tänkas drabbas om kvalitet och kundvärde påverkas negativt, då stort fokus ligger på hög funktionstillväxt.

I empirin framgick det att det råder en betydande skillnad mellan olika team i hur mycket av deras arbete som innefattas av story points, samt konsekvenserna av detta. En utvecklare ansåg att 90 % av teamets arbetsuppgifter var poängsatta medan en annan utvecklare, från ett annat team, ansåg att deras siffra låg på 50 %. I kombination med detta framkom det även att utvecklarna kan se ett värde i att prioritera uppgifterna som är poängsatta framför de som inte är det. Anledningen kan vara att uppgifter som inte är poängsatta inte bokförs och syns alltså inte på samma sätt. I litteraturen framgick det att nedprioriteringen av arbetsuppgifter utan story points kan medföra problem. Till exempel kan det vara så att diverse tester inte är poängsatta och därmed inte genomförs, vilket kan leda till bristande kvalitet. Problemet kan ha sitt ursprung i det faktum att projektet är i ett tidigt stadium och att arbetarna ännu inte hunnit vänja sig vid Scrum-metodiken. Nedan följer en analys och diskussion av Sprint burn-down, Velocity, Burn-up och Happiness.

### 5.2.1 Sprint burn-down

Definitionsmässigt är Delphis syn på Sprint burn-down likställd med beskrivningen som återfinns i teorin. Med andra ord, både teorin och empirin menar att Sprint burn-down används för en kontinuerlig och kortsiktig uppföljning av projektets förlopp, men även för att se om projektet utvecklas enligt estimerad takt. Både empirin och teorin framhäver KPI:ns enkelhet och visuella styrka som en positiv egenskap. Att kunna upptäcka problem beskrivs som ett användningsområde i både teorin och empirin. En aspekt som endast teorin framför är att den kontinuerliga feedbacken fungerar som en motivationsfaktor för de anställda genom att direkt ge feedback på att mer och mer arbete blir gjort.

Problemet med att uppgifter inte rapporteras in i samband med att de färdigställs ger enligt teorin och empirin en rad negativa konsekvenser. Då grafen visar att mindre arbete gjorts än i verkligheten påpekar intervjupersonerna att detta kan leda till större press inom teamet att jobba mer, vilket i sin tur leder till nervositet och en negativ stämning inom teamet. Teorin framhäver problemet som en brist i det sociala perspektivet av hållbarhet, och att det skapar en bristande motivation hos utvecklingsteamet. Empirin tar även fram svårigheter i att definiera och värdera antalet story points som slutförts.

En möjlighet som teorin nämner för att undvika hack i grafen över Sprint burn-down är att göra user stories mindre, så att de snabbare och lättare kan definieras som färdiga. En intervjuad utvecklare pekade på projektets organisation som en orsak till varför beroenden uppstår för den inrapportering som ligger till grund för Sprint burn-down. Problemet hos Delphi, för klarmarkering av user stories, kan dock tänkas bero på komplexiteten i slutproduktens funktionalitet. Beroendena är helt enkelt svåra att undvika och det handlar därför snarare om att hantera konsekvenserna av dem. Dessa kan minimeras genom att utbilda ledare i det agila sättet att leda team. Kunskap behöver kommuniceras ut angående "hockey-stick"-mönstret, mer generellt att KPI:erna inte ska användas likt de görs i traditionell mjukvaruutveckling. KPI:erna är inte statiska mätetal som ledningen ska styra hårt på, utan ledningens roll är snarare att underlätta för teamen i deras dagliga arbete och skapa möjligheter för dem att utvecklas. På så sätt kan en mer socialt hållbar arbetsmiljö uppnås.

*Tabell 4 Jämförelse av Sprint burn-down mellan teori och empiri*

| **Sprint burn-down** | **Användningsområden** | **Problemområden** |
|---|---|---|
| Teori | ● Följa arbetsutvecklingen kortsiktigt<br>● Upptäcka problem<br>● Motivera medarbetare | ● Definition av avklarat arbete<br>● Hårt ställda krav |
| Empiri | ● Följa projektutveckling på låg nivå<br>● Utvärdera planering<br>● Upptäcka problem | ● Definition av avklarat arbete<br>● Definition och värdering av en story point<br>● Arbete som inte är poängsatt<br>● Beroenden för klarmarkering<br>● Sen rapportering |

### 5.2.2 Velocity

Det primära syftet bakom Velocity är, både enligt Delphi och litteraturen, att förbättra estimeringen. Det framgår av litteraturen att användningen av Velocity för att allokera resurser kan orsaka produktivitetsförluster. Detta då det kan bli avbrott i inarbetade rutiner. I empirin framgick det också att resurser i vissa fall har förflyttats mellan team. En förklaring kan dock vara att projektet är i ett tidigt stadium och att dynamik i projektorganisationen därför är svår att undvika. Det är viktigt att vara medveten om konsekvenserna av att inte hålla teamen konstanta och att i största möjliga mån försöka undvika större förändringar. Ett annat syfte med Velocity, som har framgått både i empiri och litteratur, är att utvecklingen av Velocity över tid kan studeras för att undersöka hur teamen utvecklas. KPI:n blir på så sätt ett verktyg med vilket framsteg kan identifieras.

I litteraturen framgår det att Velocity kan användas som ett prestationsmått, men att det inte alltid ger en rättvisande bild då ingen hänsyn tas till kundvärde. Det här var även något som framgick under de empiriska studierna, i vilka det fastställdes att en koppling till både kundnytta och arbetsinsats saknas. En konsekvens är då att Velocity bara blir en siffra utan stöd i vad som verkligen händer. En för hård styrning på Velocity kan, enligt teorin, även leda till en mängd olika problem. Exempel på dessa problem kan vara att genvägar tas, vilket kan innebära att Technical debt uppstår, samt att KPI:ns effekt som ett planeringsverktyg urholkas. I empirin råder det delade meningar kring att använda Velocity som ett prestationsmått. Två av tre intervjuade utvecklare menar att Velocity faktiskt används som ett prestationsmått medan den tredje säger att det är fel att skapa konkurrens mellan team och att Velocity därför inte används som ett prestationsmått. Problemen med att allt arbete inte är poängsatt och definitionen av färdigt, som togs upp i 5.2.1, påverkar även Velocity och berörs i såväl litteratur som empiri.

Att använda Velocity som ett prestationsmått går emot grundidén med Scrum. När man talar om KPI:er i en agil kontext är en av grunderna att mätningarna ska följa trender, och inte statiska tal. Att flytta fokus, samt kommunicera ut detta, från det specifika värdet till en utveckling över tid kan minska pressen som utvecklarna upplever från ledningens sida. Ett alternativ för Delphi hade också kunnat vara att införa två separata uppsättningar av KPI:er, en för ledningen och en för utvecklarna. I det fallet hade Velocity kunnat ingå i endast utvecklarnas uppsättning av KPI:er. Då hade pressen på hög Velocity kunnat minskas eftersom ledningen inte haft insyn i teamens Velocity. Bara för att man använt mätningarna på ett visst sätt tidigare betyder inte det att det är mest optimala sättet. Syftet och nyttan med Velocity bör reflekteras över. Eftersom det huvudsakliga målet är att hjälpa utvecklarna vid estimering och planering bör ledningens användning ifrågasättas. Tidigare beskrivna risker är annars Technical debt samt sänkt kvalitet och kundvärde.

*Tabell 5 Jämförelse av Velocity mellan teori och empiri*

| **Velocity** | Användningsområden | Problemområden |
|---|---|---|
| Teori | ● Öka förutsägbarhet<br>● Mäta produktivitet<br>● Focus factor | ● Icke schemalagt arbete inte inkluderas<br>● Optimistiska estimeringar<br>● Technical debt<br>● Svårigheter i estimering |
| Empiri | ● Förbättra estimering<br>● Uppnå ständiga förbättringar<br>● Identifiera framsteg och möjligheter | ● Definition av avklarat arbete<br>● Press på estimatuppfyllelse<br>● Användning som prestationsmått<br>● Koppling till kundnytta<br>● Bristande användning<br>● Arbete som inte är poängsatt |

### 5.2.3 Burn-up

De empiriska resultat som framkommit från intervjuerna gällande användningsområden för KPI:n Burn-up stämmer överens med den beskrivning som återges i teorin. Fokus är statusuppföljning på hög nivå för projektet, med inblandning från både Delphi och Volvo Cars. I litteraturen beskrivs Burn-up som en övergripande kategori för grafer som visar tillväxt, medan Delphi har specificerat subtyper för epics och krav.

KPI:n beskriver situationen på hög nivå, vilket stärks av intervjupersonernas bild av att tolkningen är svår och kräver kommunikation. Angående klarmarkering av epics menar en utvecklare att det inte är möjligt i tillräckligt hög grad på grund av beroenden av andra team. På grund av dessa begränsningar uppstår onödig uppföljning och uppmärksamhet när utvecklare behöver besvara ledningens förfrågningar. Orosmoment skapas, vilka leder till stress och en social miljö som inte är hållbar i längden. Specifikt för Requirements burn-up är bland annat problemet att kravtillväxten inte visar hur stor arbetsmängd som lagts ned, då krav inte är estimerade i story points.

En låg uppfattad nytta och en bristande motivation kring Requirements burn-up leder till att ledningen och utvecklare lägger tid på aktiviteter som inte tillför något värde. Om nyttan KPI:n medför inte överstiger det krävda arbetet för dessa icke värdeskapande aktiviteter erhålls en suboptimering. Vidare kan den låga motivationen kring användningen av KPI:n leda till en ond cirkel för teamen. Låg motivation leder till bristfällig inrapportering, vilket i sin tur ger mindre tillförlitliga mätningar och i längden en ännu lägre motivation för att använda KPI:erna i vardagen.

*Tabell 6 Jämförelse av Burn-up mellan teori och empiri*

| **Burn-up** | Användningsområden | Problemområden |
|---|---|---|
| Teori | ● Projektuppföljning på hög nivå<br>● Överblick över adderat arbete | ● Nyttor gentemot Burn-down förloras vid ett bestämt omfång<br>● Svårigheter i tolkning |
| Empiri (Epic burn-up) | ● Följa projektutveckling på hög nivå<br>● Se status inför nästa leverans | ● Svårigheter i tolkning<br>● Beroenden av andra team för klarmarkering<br>● Låg tillförlitlighet initialt |
| Empiri (Requirements burn-up) | ● Kontrollera omfång<br>● Följa projektutveckling | ● Svårigheter i att motivera inrapportering<br>● Mängden krav<br>● Kopplingen till arbetsmängd |

### 5.2.4 Happiness

I litteraturgenomgången som låg till grund för det teoretiska ramverket identifierades Happiness, en KPI som Delphi inte använder i det aktuella projektet. Grundidén är att följa upp medarbetarnöjdheten kontinuerligt eftersom detta korrelerar väl med en medarbetares prestation. På så sätt kan utvecklare reflektera över arbetssituationen och föra tankarna uppåt till ledningen. Merparten av de utvecklare som intervjuats har nämnt att KPI:n är intressant. För Delphis del skulle införandet av Happiness som KPI potentiellt kunna ge en bättre arbetsmiljö och främja den sociala hållbarheten. Med medarbetare som gör ett effektivare jobb förbättras även den ekonomiska hållbarheten.

Medarbetarnöjdhet mäts på Delphi i dagsläget, men endast vartannat eller vart tredje år. Detta är för sällan för att uppnå de kontinuerliga förbättringar som KPI:n eftersträvar. En tillfällig medarbetarnöjdhet kanske existerar, men den återspeglar inte hur personen har känt över året som helhet. Ett problem med att införa KPI:n skulle vara den arbetsbörda som detta innebär. Happiness måste dokumenteras och kräver administration.

*Tabell 7 Jämförelse av Happiness mellan teori och empiri*

| **Happiness** | Användningsområden | Problemområden |
|---|---|---|
| Teori | ● Förbättra psykiskt och fysisk arbetsmiljö<br>● Förbättra produktivitet<br>● Planering och allokering | ● Behov av frekventa mätningar<br>● Behov av uppföljning<br>● Koppling mellan lycka och förbättring |
| Empiri | ● Används inte | ● Används inte |

## 5.3 Kvalitet

I litteraturstudien framgick vikten av att säkerställa kvaliteten i syfte att erhålla en hög kundnöjdhet. Det finns dock ingen enskild KPI som allena kan följa upp detta på ett lämpligt sätt. En kombination av till exempel antal defekter och komplexiteten ger ett djup till analysen och en bättre förmåga att agera om det skulle uppstå problem. På längre sikt är Technical debt värdefullt att följa upp i projekt, då koden och arbetsprocessen annars blir allt mer svårhanterlig med tiden. De empiriska resultaten visar att Delphi lider av bristfällig kunskap vad gäller hantering av defekter. En anledning anges vara projektets tidiga stadium, vilket innebär att det inte har uppstått så många defekter än. Korrekt hantering av defekter från projektets start är viktigt för att kunna säkerställa kvaliteten, men även för att bättre kunna prioritera arbetet på kort sikt och planera projektet på längre sikt. Det är därför viktigt att förankra ett tydligt och gemensamt arbetssätt i hela verksamheten. Nedan följer en analys och diskussion om KPI:erna Defect count, Bug correction time och Technical debt.

### 5.3.1 Defect count

Både teori och empiri föreslår Defect count som ett mått på kvalitet, och menar att det krävs för att en god kundnöjdhet ska kunna uppnås. De båda menar också att KPI:n kan användas i planeringssyfte. I intervjuer nämns bara planering ur ett tidsperspektiv medan teorin föreslår att användningsområdet kan utvidgas till att även innefatta kostnader och omfång. I teorin rekommenderas också att Defect count används i relation till tid för att vara ett mått på hur kvalitativ kod som producerats under en viss period. Att Delphi inte använder sig av KPI:n på det här sättet i dagsläget får konsekvenser. Det mest väsentliga är att företagen riskerar att bygga upp mycket Technical debt under perioder utan att de själva lägger märke till det. När Technical debt ökar blir det också allt svårare att lägga till ny funktionalitet. Ett mått på exempelvis Defect count per sprint hade varit användbart för att få en bättre uppfattning om när och varför Technical debt uppstår, och därigenom kunna kontrollera skulden på ett bättre sätt. Ett problem med att införa KPI:n, enligt både utvecklare och ledningen, är att det inte uppstått så många defekter i projektet ännu och att nyttan med KPI:n därför uppfattas som låg. Dessutom finns det fler problem som behöver tas hand om innan KPI:n har bra förutsättningar på företaget, exempelvis behöver ledningen ge tydligare riktlinjer kring inrapportering av defekter.

Ett problem med att införa KPI:n, enligt både utvecklare och ledningen, är att det inte uppstått så många defekter i projektet ännu och att nyttan med KPI:n därför uppfattas som låg. I teorin beskrivs att företag inte får fokusera alltför mycket på att hålla nere antalet defekter. För sträng kontroll leder till att produkten tar lång tid att utveckla. Även i empirin får det här medhåll. Defekter är en naturlig del av mjukvaruutveckling, enligt Delphi, och fokus bör istället ligga på att effektivt kunna fånga upp och lösa dem. Teorin beskriver även att antalet defekter är starkt kopplat till hur uttömmande testningsprocessen är. Därför är väl planerade test en förutsättning för att KPI:n ska vara användbar. Ytterligare ett problem, enligt teorin, är att KPI:n inte säger något om allvarlighetsgraden för de defekter som existerar. Om ingen hänsyn tas till det blir inte Defect count ett bra mått på mjukvarans kvalitet. Genom att poängsätta varje defekt utifrån allvarlighetsgrad kan sedan en genomsnittlig allvarlighetsgrad för defekterna i Delphis projekt räknas ut. Detta, i relation till antalet defekter, skulle ge ett bättre mått på produktens kvalitet. Genom att ha ett sådant mått går det att planera projektet bättre, och kunden får en tydligare bild över vad det är som levereras. Ett problem är att kunden kanske börjar styra ännu mer med avseende på defekter och försöker se till att de hålls nere i varje leverans. Defekter är en naturlig del av mjukvaruutveckling och för hård styrning kring dem gör att ny funktionalitet tar längre tid att utveckla.

Något som nämns i empirin men inte i teorin är problemen rent praktiskt med att rapportera defekter. Ledningen anser att många defekter rapporteras med otillräcklig dokumentation och att problem uppstår av den anledningen. Å andra sidan anser två av tre intervjuade utvecklare att de har fått otillräckligt med information kring inrapporteringen av defekter.

*Tabell 8 Jämförelse av Defect count mellan teori och empiri*

| **Defect count** | Användningsområden | Problemområden |
|---|---|---|
| Teori | ● Mäta kvalitet<br>● Planera projekt<br>● Motivera medarbetarna | ● Behov av uttömmande testning<br>● Hård styrning<br>● Avsaknad av allvarlighetsgrad |
| Empiri | ● Säkerställa kvalitet<br>● Uppfylla kundkrav<br>● Planera projekt | ● Otillräckligt med dokumentation<br>● Bristande riktlinjer kring rapportering<br>● Användning som resultatmått |

### 5.3.2 Bug correction time

Det huvudsakliga syftet med Bug correction time är att underlätta planeringsarbetet, vilket både empiri och teori är överens om. Teorin ger ett konkret förslag på hur detta kan användas, vilket är att bedöma hur lång tid det tar innan en viss utlovad kvalitet är uppnådd. Ett annat användningsområde som nämns i empiri är prioritering av defekter. Om defekter kategoriseras med avseende på allvarlighetsgrad kan Bug correction time användas för att jämföra kategorierna för att alltid se till att hög allvarlighetsgrad har lägst lösningstid. I teorin breddas användningsområdet ytterligare och det föreslås att KPI:n används även för att prioritera arbetsuppgifter. Prioriteringen handlar om vad det för tillfället bör läggas resurser på, såsom att utveckla ny funktionalitet eller att göra den existerande produkten felfri. Om kvaliteten på produkten blir alltför låg i någon leverans skapar det missnöje hos kunden som kan skada relationen. En nackdel med mätningen är att den inte säger någonting om hur allvarliga defekterna är, eller hur många defekter som ännu inte är lösta. Därför borde den kombineras med någon annan KPI, exempelvis Defect count.

Problem som identifierats i både litteraturstudien och intervjuerna är exempelvis behovet av att ha lätthanterliga verktyg för rapportering av defekter. På Delphi anses problemet, enligt ledningen, vara att utvecklare ger otillräcklig rapportering och enligt flera utvecklare finns det otillräckligt med riktlinjer för hur defekter ska hanteras. Ett förmedlat problem på Delphi är att defekter av hög allvarlighetsgrad i genomsnitt tar längre tid att lösa än defekter med låg allvarlighetsgrad. Eftersom allvarliga defekter riskerar att påverka kundvärdet i större utsträckning än mindre allvarliga, bör förhållandet i en optimal värld vara det omvända. Samtidigt finns det ingen information kring huruvida en hög allvarlighetsgrad för en defekt innebär att den är svårare att lösa. Om defekter med en högre allvarlighetsgrad inte är svårare att lösa tyder detta på att Delphi måste se över sin defekthantering. Eftersom projektet är i ett tidigt stadie har inte särskilt många defekter uppstått ännu, av den anledningen finns det också osäkerheter kring resultatet.

I teorin nämns att en bra modell krävs för att kunna avgöra hur stor Bug correction time kommer att vara för en defekt. Det ifrågasätts också vad nyttan med KPI:n egentligen är i och med att den mäter individuell prestation mer än teamprestation. Detta är något som bör undvikas inom agil utveckling.

*Tabell 9 Jämförelse av Bug correction time mellan teori och empiri*

| **Bug correction time** | Användningsområden | Problemområden |
|---|---|---|
| Teori | ● Planera projekt<br>● Prioritera arbete | ● Svårigheter i tidsuppskattning<br>● Behov av lätthanterliga verktyg<br>● Utvärdering av individer |
| Empiri | ● Planera projekt | ● Otillräckligt med dokumentation<br>● Bristande riktlinjer kring rapportering |

### 5.3.3 Technical debt

Technical debt är en KPI som Delphi inte använder sig av i dagsläget. Den mäter hur mycket omarbete som riskerar uppkomma i samband med att kod håller för låg kvalitet. Att Delphi inte använder sig av KPI:n medför både risker och konsekvenser. En risk är att stor Technical debt uppstår utan att företaget lägger märke till det, vilket i sin tur kan leda till att funktioner tar allt längre tid att implementera, utan att den egentliga orsaken till problemet kan identifieras. Delphi försöker i dagsläget undvika Technical debt genom statisk kodanalys och riktlinjer för designen. För att undersöka ifall de här metoderna är effektiva medel för att få bra kod hade det varit fördelaktigt att mäta KPI:n.

Ett problem med införandet av KPI:n är att det krävs en avancerad modell för mätningen. Delphi måste utgå från vad de själva anser som kvalitativt skriven kod och konstruera en egen modell utifrån detta.

*Tabell 10 Jämförelse av Technical debt mellan teori och empiri*

| **Technical debt** | Användningsområden | Problemområden |
|---|---|---|
| Teori | ● Säkerställa framtida kvalitet och effektivitet | ● Svårigheter i att motivera kostnader<br>● Svårigheter i att kvantifiera |
| Empiri | ● Används inte | ● Används inte |

## 5.4 Kundvärde

Kundvärde är huvudmålet i agil utveckling, men följs inte upp genom mätningar i det aktuella projektet. I dagens globala marknad, i vilken konkurrensen blir större och större, är det viktigt att maximera kundvärdet i leveranserna för att undvika att kunden går till en konkurrent. I empirin framgår det att samtliga intervjuobjekt anser att en värdeaspekt hade varit relevant samt även underlättat deras dagliga arbete. Resurser kan då allokeras till de aktiviteterna med högst kundvärde. Det sitter utvecklare från Volvo Cars på plats på Delphis kontor, men de empiriska resultaten visar att det inte är tillräckligt. Informationen går inte hela vägen från beställaren till utvecklaren. Två sätt att följa upp kundvärde under projektets gång är med hjälp av KPI:erna NPS och Earned business value, vilka presenteras mer i detalj nedan.

### 5.4.1 Net promoter score

Kundvärde mäts inte av Delphi, men förespråkas av litteraturen och efterfrågas av utvecklare. Konsekvenserna av denna avsaknad är att funktionalitet som inte efterfrågas av kunden riskerar att utvecklas i onödan. NPS ger kontinuerlig feedback som för utvecklingsteamet, ledningen och kunden är användbar främst i utvecklingsfasen, men även i faserna efter utveckling såsom underhåll och uppdatering av mjukvaran. Tack vare Net promoter scores enkelhet kan lätt stora mängder recensioner samlas in och ge direkt och pålitlig feedback till de involverade parterna i projektet. Om resultatet från NPS är positivt kan det även fungera som en motivationsfaktor då det är ett bevis på att utvecklarna och ledningen lyckats med projektet, vilket bidrar till den sociala hållbarheten inom företaget.

*Tabell 11 Jämförelse av Net promoter score mellan teori och empiri*

| **Net promoter score** | Användningsområden | Problemområden |
|---|---|---|
| Teori | ● Mäta kundlojalitet<br>● Motiverar medarbetarna | ● Behov av kompletterande frågor<br>● Kulturella skillnader<br>● Behov av viktning |
| Empiri | ● Används inte | ● Används inte |

### 5.4.2 Earned business value

Delphi använder i dagsläget ingen KPI för att mäta kundvärde, då det inte ses som relevant från ledningens sida på grund av att projektet är ett fastprisprojekt. Att kundvärde inte används som mått innebär att administrationen av projektet blir lidande. Primärt skulle Delphi få nytta av att veta hur mycket värde som har levererats till kunden, då detta ger ytterligare ett perspektiv över hur projektet fortlöper. Ledningen och kunden får därmed information om hur projektet utvecklas.

Enligt de intervjuade utvecklarna vill de gärna ha ett kundvärdesperspektiv på sitt arbete, för att underlätta prioriteringen av user stories. Detta skulle kunna lösas genom att införa KPI:n Earned business value. Som tidigare beskrivet är syftet med Earned business value att uppskatta hur mycket värde som levereras till kunden. Å andra sidan menar ledningen att projektets epics redan är prioriterade utefter uppskattade kundvärden. Detta visar på ett behov av förbättrad kommunikation i projektet.

Trots att ledningen inte ser något större incitament i att införa KPI:er för kundvärde i projektet, då det är av fastpriskaraktär, bör nyttorna med Earned business value poängteras. Genom att mäta värde får ledningen möjlighet att se hur mycket värde som levererats till kunden. Detta kan inte endast uppnås genom att kolla på antalet user stories eller funktioner som levererats, då alla user stories och funktioner har olika värde för kunden. Genom att kontinuerligt se hur mycket värde som levererats kan resurser administreras bättre, exempelvis genom att involvera fler utvecklingsteam på de delar av projektet som anses vara viktigast. Dessutom, genom att mäta kundvärde, ser man till att endast leverera kod som skapar nytta för kunden. Även om kod är välskriven och tillför funktionalitet är det meningslöst så länge som det inte tillför kundvärde. Ytterligare en fördel med att mäta kundvärde genom Earned business value är i eventuella förhandlingar med kunden. Om Delphi kan ta fram konkreta siffror på kundvärde som levererats kan Delphi stärka sina argument vid eventuella tvister med kunden. Genom att både Delphi och kunden har konkreta siffror på levererat kundvärde kan ett bättre samarbete och en högre nöjdhet uppnås.

*Tabell 12 Jämförelse av Earned business value mellan teori och empiri*

| **Earned business value** | Användningsområden | Problemområden |
|---|---|---|
| Teori | ● Underlätta beslutsfattande<br>● Prioritera arbete | ● Svårigheter i att definiera värde |
| Empiri | ● Används inte | ● Används inte |

# 6. Slutsatser och rekommendationer

Studien har behandlat en utvärdering av de KPI:er som Delphi använder för det studerade projektet, vilket genomförs i samarbete med kunden Volvo Cars. Till grund för dessa rekommendationer ligger en litteraturstudie samt empiriska resultat som kommer från intervjuer med anställda på Delphi och Volvo Cars. Kapitlet kommer att besvara den fjärde och sista frågeställningen för rapporten:

***4. Hur bör Delphi styra sina agila mjukvaruutvecklingsprojekt med KPI:er?***

Den första rekommendationen berör övervägandet av en mer övergripande modell för att strukturera och välja KPI:er. Exempelvis GQM, som utgår från vilka mål företaget har med sina projekt. De sex KPI:er som Delphi fokuserar på för tillfället syftar till att ge stöd till planeringsarbetet samt att följa upp kvaliteten i arbetet. Trots ett uttryckt mål med att leverera ett högt kundvärde använder Delphi inga KPI:er för att mäta det. GQM kan användas för att säkerställa att det finns ett syfte bakom alla KPI:er och att det finns KPI:er för alla företagets mål. Genom att arbeta med ett strukturerat system för att följa upp projekten möjliggörs det att en ökad acceptans uppnås för mätningarna. Ökad förståelse för målen med KPI:erna tros även leda till att problemen kan undvikas med att ledningen använder resultaten av mätningarna som en utvärdering av utvecklarnas prestation. Exempelvis bör KPI:er som används i planeringssyfte inte användas som prestationsmått på utfört arbete.

Att ta hänsyn till kundvärde är vitalt i dagens företagsklimat. Snabbrörliga marknader och krav på hög flexibilitet motiverar uppföljning av hur leveranserna mottages av kunden. Nästa rekommendation är således att införa mätning av kundvärde, med hjälp av KPI:erna Earned business value och Net promoter score. En implementering av dessa kan vara till nytta vid prioritering av arbete. Sällan lyckas hundra procent av funktionaliteten levereras till en viss deadline, varför det är av vikt att kunna prioritera det arbete som tillför högst värde.

För att uppnå ett högt kundvärde är det viktigt att kunna leverera produkter med en hög kvalitet. I dagsläget mäter Delphi huvudsakligen antalet defekter för att följa upp kvaliteten. Rekommendationen är att kombinera Defect count med en genomsnittlig allvarlighetsgrad för defekter samt att kvantifiera Technical debt. Syftet är att få ett mer rättvisande mått på produktkvalitet och möjliggöra mätning av komplexitet. Genom att förankra detta i ett gemensamt arbetssätt säkerställs den kort- och långsiktiga kvaliteten, vilket gynnar den ekonomiska hållbarheten genom en ökad kundnöjdhet.

Hög kvalitet bygger också på en proaktiv inställning till problemhantering. Genom att följa upp medarbetarnöjdhet kontinuerligt kan eventuella problemområden i projektet upptäckas i ett tidigt stadium, och på så sätt undvikas. Delphi rekommenderas därför att börja använda KPI:n Happiness, vilken identifierades i litteraturstudien. Genom att vidta åtgärder som håller medarbetarna nöjda uppnås även en social hållbarhet, samtidigt som verksamheten blir mer produktiv och effektiv. Kommunikationen främjas mellan ledningen och medarbetarna, vilket underlättar förståelsen och tolkningen av de mätningar som görs i projektet.

Avslutningsvis, det studerade pilotprojektet på Delphi är ett steg på vägen från den traditionella vattenfallsmodellen till agil utveckling. Omställningen är stor för alla inblandade parter. Med högre krav på flexibilitet och ett större kundfokus ställs nya utmaningar på projektstyrningen. KPI:er som verktyg får en ny roll med syftet att stödja teamen, snarare än att enbart kontrollera dem. Trots att Delphi arbetar med nya, agila KPI:er lever delar av organisationen kvar i den

traditionella projektstyrningen. Kommunikation och utbildning är två centrala delar i en framgångsrik tillämpning av agil utveckling, vilket är avgörande för att Delphi ska kunna möta framtidens krav.

# Referenslista


Anderson, J.C. & Narus, J.A. (1998) *Business marketing: understand what customers value,* https://hbr.org/1998/11/business-marketing-understand-what-customers-value [2016-05-06]

Arafeen, J. & Bose, S. (2009) *Improving Software Development Using Scrum Model by Analyzing Up and Down Movements On The Sprint Burn Down Chart-Proposition for Better Alternatives.* International Journal of Digital Content Technology and its Applications, 3(3):109-115.

Balbes, M.J. (2012) *Measuring the Business Value of Projects with Agile* https://adtmag.com/articles/2012/08/30/projects-with-agile.aspx [2016-05-06]

Basili, V.R., Lindvall, M., Regardie, M., Seaman, C., Heidrich, J., Munch, J., Rombach, D. & Trendowicz, A. (2010) *Linking Software Development and Business Strategy Through Measurement*, Computer, 43(4):57-65.

Bhattacharya, P. & Neamtiu, I. (2011) *Bug-fix time prediction models: can we do better?* University of California, Riverside, CA, USA

Boehm, B., Rombach, H., Zelkowitz, D. & Marvin, V. (2005) *Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili.* Berlin/Heidelberg: Springer-Verlag

Broadus, W. (2013) *Stakeholder Needs and Expectations: Planning Your Agile Project and Program Metrics* Defense AT&L, Nov/Dec2013, 42(6):50-54

Cheng, T.H., Jansen, S. & Remmers, M. (2009) *Controlling and monitoring agile software development in three dutch product software companies*. In Proceedings of the 2009 ICSE Workshop on Software Development Governance, SDG 2009, pages 29-35, Vancouver, BC

Clarios Technology (2016a) *Burn up vs burn down chart* http://www.clariostechnology.com/productivity/blog/burnupvsburndownchart [2016-05-12]

Clarios Technology (2016b) *What is a burn up chart?* http://www.clariostechnology.com/productivity/blog/whatisaburnupchart [2016-05-12]

Cockburn, A. (2006) *A Governance Model for Incremental, Concurrent, or Agile Projects* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.642.3181&rep=rep1&type=pdf [2016-05-14]

Cooil, B., Wallin Andreassen, T., Aksoy, L., Keiningham, T. (2016) *Net promoter score (NPS) - a balanced view* http://www.customerchampions.co.uk/net-promoter-score-nps-a-balanced-view/ [2016-05-06]

Delphi (2015) *Annual Report 2014.* United Kingdom: Delphi Automotive PLC. http://delphi.com/docs/default-source/financial-documents/delphi_2014_annual_report.pdf [2016-05-07]

Downey, S. & Sutherland, J. (2013) *Scrum Metrics for Hyperproductive Teams: How They Fly like Fighter Aircraft*, IEEE, pp. 4870.

Folkhälsomyndigheten (2014) *Social hållbarhet* https://www.folkhalsomyndigheten.se/motesplats-social-hallbarhet/social-hallbarhet/ [2016-05-08]

Geffel, J. (2011) *Some Thoughts Agile Scrum & Customer Value* http://www.valuedrivengroup.com/johns-blog/some-thoughts-agile-Scrum-customer-value [2016-05-06]

Gopal, A., Krishnan, M.S., Mukhopadhyay, T. & Goldenson, D.R. (2002) *Measurement programs in software development: determinants of success*, IEEE Transactions on Software Engineering, 28(9): 863-875.

Gustafsson, J. (2011) *Model of Agile Software Measurement: A Case Study*, Gothenburg: Chalmers University of Technology

Gutbrod, R. & Wiele, C. (2012) *The Software Dilemma* Heidelberg: Springer Verlag

Harnish, V. & Ross, S. (2013) *A better way to measure employee happiness* https://gazelles.com/static/resources/articles/aBetterWayMeasuerEmployeeHappiness.pdf [2016-04-27]

Hartmann, D. & Dymond, R. (2006) *"Appropriate agile measurement: using metrics and diagnostics to deliver business value,"* Proc. of the conference on Agile 2006, IEEE Computer Society, s. 126-134

Hong, G.Y. & Goh, T.N. (2004) *A comparison of Six-Sigma and GQM approaches in software development*, International Journal of Six Sigma and Competitive Advantage, vol. 1, no. 1

Hughes, B. & Cotterell, M. (2009) *Software project management*, 5th edition, London: McGraw-Hill Higher Education

Hughey, D. (2009) *The Traditional Waterfall Approach* http://www.umsl.edu/~hugheyd/is6840/waterfall.html [2016-04-25]

Hodgetts, P. (2004) *Refactoring the development process: Experiences with the incremental adoption of agile practices*. In Proceedings of the Agile Development Conference, ADC 2004, 106-113, Salt Lake City, UT

Investopedia (2016) *Key Performance Indicators - KPI* http://www.investopedia.com/terms/k/kpi.asp [2016-05-12]

Kan, S.H. (2003) *Metrics and models in software quality engineering*, Boston: Addison-Wesley

Katham, S. (2015) *Earned business value* https://www.Scrumalliance.org/community/articles/2015/may/agile-projects-–-earned-business-value [2016-05-07]

Kupiainen, E., Mäntylä, M. & Itkonen, J. (2014) *Why Are Industrial Agile Teams Using Metrics and How Do They Use Them?* WETSoM 2014 Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics

Kupiainen, E., Mäntylä, M. & Itkonen, J. (2015) *Using metrics in Agile and Lean Software Development - A systematic literature review of industrial studies*, Information and Software Technology, vol. 62, pp. 143-163.

Lavinsky, D. (2014) *Pareto Principle: How To Use It To Dramatically Grow Your Business* http://www.forbes.com/sites/davelavinsky/2014/01/20/pareto-principle-how-to-use-it-to-dramatically-grow-your-business/#52742a2b1259 [2016-05-06]

Le Duc, M. (2011a) *Induktion, deduktion och abduktion* http://www.leduc.se/metod/Induktion,deduktionochabduktion.html [2016-04-28]

Le Duc, M. (2011b) *Validitet och reliabilitet* http://www.leduc.se/metod/Validitetochreliabilitet.html [2016-05-07]

Lundequist, J. (1995) *Design och produktutveckling. Metoder och begrepp*. Lund: Studentlitteratur

Magnusson, J. & Nilsson, A. (2014). *Enterprise System Platform.* Lund: Studentlitteratur AB.

Mahnic, V. & Zabkar, N. (2012) *Measuring progress of Scrum-based software projects.* Electronics and Electrical Engineering, 18(8):73-76

Mannila, J. (2013) *Key Performance Indicators in Agile Software Development*. Tampere Satakunta University of Applied Sciences.

Markey, R. (2014) *The Benefits of a Competitive Benchmark Net promoter score* http://www.bain.com/publications/articles/the-benefits-of-a-competitive-benchmark-net-promoter-score.aspx [2016-05-06]

Martano, G. & Sedehi, H. (2012). *Metrics to evaluate & monitor Agile based software development projects.* Assasi: IEEE.

Matarelli, M. (2011) *Sustainable Pace: Trusting Your Teams* https://www.Scrumalliance.org/community/articles/2011/december/sustainable-pace-trusting-your-teams [2016-04-27]

Medallia (2016) *Net promoter score* http://www.medallia.com/net-promoter-score/ [2016-05-06]

Mindlance (2016) *Testing Metrics* http://www.mindlance.com/documents/test_management/testing_metrics.pdf [2016-04-27]

Mittal, N. (2013) *The Burn-Down Chart: An Effective Planning and Tracking Tool* https://www.Scrumalliance.org/community/articles/2013/august/burn-down-chart-%E2%80%93-an-effective-planning-and-tracki [2016-04-27]

Murphy, H. & Lang, A. (2014) *Business and Sustainability,* Cham: Springer International Publishing

Murugesan, S., Gangadharan, G.R. (2012) *Harnessing green IT: principles and practices,* IEEE Chichester, West Sussex, UK.

Padmini, K.V.J., Dilum Bandara, H.M.N. & Perera, I. (2015). *Use of Software Metrics in Agile Software Development Process.* Moratuwa: IEEE.

Petersson, B. & Zhang, S. (2013) *An Approach to Improve Quality of Software Using Metrics and Technical debt - A case study within Model- Driven Environment.* Chalmers University of Technology, Göteborg.

Power, K. (2011) *The Agile Office: Experience Report from Cisco's Unified Communications Business Unit* Agile Conference (AGILE) 2011. IEEE.

Project Insight (2016) *5 Basic Phases of Project Management* http://www.projectinsight.net/project-management-basics/basic-project-management-phases [2016-04-25]

Project Management Institute (2016). *What is Project Management?* http://www.pmi.org/About-Us/About-Us-What-is-Project-Management.aspx [2016-04-25]

Prowareness (2016) *Agile Metrics: Let the Numbers tell the Tale* http://www.Scrum.nl/media/Agile_Metrics/$FILE/whitepaper_agile_metrics.pdf [2016-04-26]

PSM Support Center (2016) *About PSM* http://www.psmsc.com/AboutPSM.asp [2016-04-27]

Rawsthorne, D. (2010) *Calculating Earned business value for an Agile Project* https://cs.anu.edu.au/courses/comp3120/public_docs/CollabNet_WP_Earned_Business_Value_041910.pdf [2016-05-07]

Reichheld, F.F. (2003) *The one number you need to grow*, https://hbr.org/2003/12/the-one-number-you-need-to-grow/ar/1 [2016-05-06]

Sandberg, A., Staron, M. & Antinyan, V. (2015) *Towards Proactive Management of Technical debt by Software Metrics,* University of Gothenburg: Gothenburg

Scrum Alliance (2016) *The Scrum Guide* https://www.Scrumalliance.org/why-Scrum/Scrum-guide [2016-05-12]

Sharma, T. & Smarthyam, G. (2015) *Limitations of Technical debt Quantification: Do you Rely on these Numbers?* https://www.infoq.com/articles/tech-debt-quantification [2016-04-27]

Staron, M., Meding, W. & Söderqvist, B. (2010) *A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation.* Information and Software Technology, 52(10):1069-1079

Sutherland, J. (2014) *Scrum: The Art of Doing Twice the Work in Half the Time*, Talent Development, Crown Business: New York

The Agile Alliance (2001a) *Manifest för Agil systemutveckling* http://agilemanifesto.org/iso/sv/ [2016-04-25]

The Agile Alliance (2001b) *Principles behind the Agile Manifesto* http://agilemanifesto.org/principles.html [2016-05-12]

Umarji, M. & Seaman, C. (2008) *Why do programmers avoid metrics?* presented at the Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, Kaiserslautern, Germany.

United Nations (1987) *Our Common Future - Brundtland Report*. Oxford: Oxford University Press

Vásquez, F. & Díaz, R. (2011) *Applying ISO/IEC 12207:2008 with Scrum and Agile Methods* Berlin, Heidelberg: Springer, 169-180

Wolff, E. & Johann, S. (2013) *Managing Technical debt* http://www.infoq.com/articles/managing-technical-debt [2016-04-27]

Woodings, T. (2016) *Introduction to Software Project Management* http://teaching.csse.uwa.edu.au/units/CITS3220/lectures/09projManIntro.pdf [2016-04-25]

# Appendix

## Appendix A Intervjumall för ledningen

**Vad?**
Definition av KPI:n

**Varför?**
Syftet, det man vill åstadkomma

**Hur?**
Hur den mäts, vilka som rapporterar in den samt hur ofta

**Vilka?**
Vilka som använder sig av den, vilka kollar på den samt hur ofta kollar man på den?

**Alternativ?**
Har ni övervägt någon annan KPI som kanske skulle få samma resultat eller effekt?

**Samband och beroenden?**
Finns det någon KPI som samspelar med KPI:n i fråga?

**Svårigheter med KPI:n?**
Svårigheter vid exempelvis tolkning eller rapportering av KPI:n?

# Appendix B Intervjumall för utvecklare

Kort presentation om oss och vårt kandidatarbete.

Bakgrund
- Hur länge har du jobbat på, eller i samarbete med, Delphi?
- Vad är din roll i projektet?

**1. Velocity**

a. Identifiera skillnader mellan ledningens syn på KPI:er och medarbetarnas, uppnås syftet med KPI:n?
- Hur ofta läser du av Velocity?
    - Varför läser du av Velocity? Varför inte?
        - Hur justeras avvikande nivåer?

b. Identifiera skillnader mellan Delphis sätt att mäta vs litteraturens. Säkerställa att det finns en skillnad
- Hur stor är en story point?

c. Identifiera eventuella problem med KPI:n samt vad de får för konsekvenser.
- Vad ser du för problem med KPI:n?
    - Hur ofta upplever du att ni får slack/hur ofta blir det stories kvar?(Har du svårigheter med estimeringen?)
    - Vad blir det för konsekvenser?
- Hur stor del av arbetet mäts av Story points?
    - Försvårar det planeringsarbetet?
    - Finns det fördelar med att ta en uppgift med story points över en utan story points?
- Upplever du att Velocity är ett prestationsmått?
    - På vilket sätt är ledningen intresserade av er Velocity?
        - Sätter ledningen press på er att höja Velocity?
    - Jämför du er Velocity med andra teams?

**2. Sprint burn-down**

a. Identifiera skillnader mellan ledningens syn på KPI:er och medarbetarnas, uppnås syftet med KPI:n?
- Rapporterar du avslutade stories kontinuerligt eller i slutet av varje dag?
- Hur ofta läser du av KPI:n?
    - Varför läser du av KPI:n? Varför inte?
        - Hur justeras avvikande nivåer?

b. Identifiera skillnader mellan Delphis sätt att mäta vs litteraturens. Säkerställa att det finns en skillnad

c. Identifiera eventuella problem med KPI:n samt vad de får för konsekvenser.
- Vad ser du för problem med KPI:n?
    - Vad blir det för konsekvenser?

**3. Defects count**

a. Identifiera skillnader mellan ledningens syn på KPI:er och medarbetarnas, uppnås syftet med KPI:n?

- Vilka KPI:er använder du angående defekter?
- När rapporterar du in defekter?
    - Tar det lång tid, är det en komplicerad process, att rapportera in defekter?
    - Är det svårt att veta vilken information som ska fyllas i formuläret?
    - Ser du något värde i att fylla i hela formuläret?
- Hur ofta läser du av KPI:er relaterade till defekter?
    - Varför läser du av dessa KPI:er? Varför inte?
        - Vilka åtgärder vidtas om det är för många defekter?

b. Identifiera skillnader mellan Delphis sätt att mäta vs litteraturens. Säkerställa att det finns en skillnad

- Hur mäter du kvalitet?
- Hur borde Delphi mäta kvalitet?

c. Identifiera eventuella problem med KPI:n samt vad de får för konsekvenser.

- Pressar ledningen er på att hålla nere antalet defekter?
- Skiljer sig behandlingen av defekter beroende på allvarlighetsgrad?
- Är det några svårigheter/problem med att stänga en defekt, komplicerad process?

**4. Epic burn-up**

Använder du KPI:n?

a. Identifiera skillnader mellan ledningens syn på KPI:er och medarbetarnas, uppnås syftet med KPI:n?

- Hur ofta läser du av KPI:n?
- Varför läser du av Epic burn-up? Varför inte?

b. Identifiera skillnader mellan Delphis sätt att mäta vs litteraturens. Säkerställa att det finns en skillnad

c. Identifiera eventuella problem med KPI:n samt vad de får för konsekvenser.

- Vad ser du för problem med KPI:n?
- Påverkas du av resultatet på KPI:n, exempelvis övertid?
    - Som individ, som team?

**5. Requirements burn-up**

Använder du KPI:n?

a. Identifiera skillnader mellan ledningens syn på KPI:er och medarbetarnas, uppnås syftet med KPI:n?

- Hur ofta läser du av KPI:n?
- Varför läser du av Requirements burn-up? Varför inte?

b. Identifiera skillnader mellan Delphis sätt att mäta vs litteraturens. Säkerställa att det finns en skillnad

c. Identifiera eventuella problem med KPI:n samt vad de får för konsekvenser.

- Vad ser du för problem med KPI:n?
- Påverkas du av resultatet på KPI:n, exempelvis övertid? Som individ, som team?
- Ser du något värde att rapportera KPI:n eller hade det räckt med epics?
- Hur ofta rapporteras KPI:n in och tar detta lång tid?

**6. Andra KPI:er**

Identifiera eventuella andra KPI:er som skulle underlätta/förbättra arbetet.

- Finns det andra aspekter i mjukvaruutvecklingen som är värda att mäta? Fler behov av status?
    - Hade du sett något värde med att mäta kundvärde?
    - Hade du sett något värde med att mäta medarbetarnöjdhet?
    - Jobbar du för att försöka förebygga Technical debt? (
    - Hade du sett något värde med att mäta work sustainability?

**7. Fråga**

- Vilka KPI:er är viktigast? (Störst användning av)