Chalmers University of Technology

DAT255

Software Engineering Project

Reflektionsrapport

26 oktober 2017

Gruppmedlemmar:	*CID:*
Oscar Carlsson	oscca
Joakim Hulthe	hulthe
Emil Josefsson	emiljos
Lovisa Landgren	lovlan
Sofia Larborn	soflarb
William Levén	levenw
Anton Levholm	levholm
Agnes Mårdh	agnesma
Kevin Rudnick	rudnick
John Segerstedt	sejohn
Karl Wikström	karlwik
Fredrik Åberg	abergf

Innehåll

1 Syfte **1**

2 Teori **1**
2.1 Definitioner . . . 1
2.2 Scrum . . . 1
2.3 Parprogrammering . . . 2
2.4 Konfigurationshantering . . . 2

3 Reflektion **3**
3.1 Applicering av *Scrum* . . . 3
3.1.1 Roller . . . 4
3.1.2 Socialt kontrakt . . . 5
3.1.3 Samarbete . . . 5
3.1.4 Använda metoder . . . 6
3.1.4.1 Parprogrammering . . . 6
3.1.4.2 Daily Scrum . . . 6
3.1.4.3 Gemensamma arbetspass . . . 7
3.1.4.4 Planerade arbetspass . . . 7
3.1.4.5 KPI som metod för förbättring . . . 7
3.1.5 Distribution av tid . . . 8
3.1.6 Ansträngning, *velocity* och nedbrytning av uppgifter . . . 9
3.2 Praxis för användning av nya verktyg och teknologier . . . 10
3.3 Sprint reviews . . . 11
3.4 Sprint retrospectives . . . 11
3.5 KPI:er . . . 12
3.5.1 Tydliga ansvarsområden . . . 13
3.5.2 Låg teknisk skuld . . . 14
3.5.3 Nöjd kund . . . 15
3.6 Relation mellan prototyp, process och värde för intressenter . . . 16
3.7 Prioriteringar . . . 17

Referenser **17**

1 Syfte

Syftet med projektet `onTruck` var att med hjälp av ett agilt arbetssätt, och i samarbete med andra projektgrupper, utveckla programvara för att kunna köra ett antal bilar tillsammans i en kolonn. Rapporten syftar till att beskriva, reflektera över och diskutera tillämpningen av arbetssättet *Scrum* i arbetet med `onTruck`.

2 Teori

Begrepp som inte kan anses vara självklara och direkt uppenbara för läsaren definieras under definitioner. Detta avsnitt redogör även för det agila ramverket *Scrum*, parprogrammering och konfigurationshantering.

2.1 Definitioner

Grupp och team: I texten nämns ofta 'gruppen' eller 'teamet'. Dessa syftar på samma sak, gruppen *Smiley* som framtagit projektet och rapporten.

SAMO: var en av rollerna i teamet. Denna stod för gruppens välmående och sociala arbetsmiljö.

2.2 Scrum

Att arbeta agilt innebär att man följer vissa värderingar och principer. Det agila manifestet beskriver dessa värderingar så här[2]:

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

Det finns många sätt att använda det agila ramverket *Scrum*. En av grundpelarna i *Scrum* är självorganisering i teamet, dvs att medlemmarna själva äger rätten att välja hur de vill arbeta, vilket skapar kreativitet och motivation[3]. För att kunna arbeta så effektivt som möjligt bör gruppen sitta på samma ställe. Att faktiskt följa detta är upp till teamet självt som till exempel kan ha en överenskommelse om att det är okej att jobba från en annan plats en dag i veckan[3].

Produktägaren ska gärna också finnas i närheten men samtidigt fördela sin tid mellan intressenter och teamet, och följs inte detta blir teamet lidande[3]. Om det inte går att ha produktägaren närvarande i projektet bör någon i teamet agera substitut för produktägaren under möten[3].

En *Scrum master* ska försöka göra sig själv överflödig med tiden genom att vägleda teamet till att klara administrativa saker själva[3]. Detta ökar teamets möjlighet att självorganisera och *Scrum master* får möjlighet att exempelvis skriva kod eller tester[3].

Om många personer ska jobba i samma projekt kan det vara svårt att se den bästa lösningen. Ett alternativ är att sätta alla i samma stora team och ett annat alternativ är att dela in personerna i

mindre team. En vanlig uppfattning av den optimala storleken på ett team är någonstans mellan fem och nio personer[3]. Det antal som är mest optimalt, och därför vad man borde eftersträva, är kanske snarare mellan tre och åtta personer[3]. Att vara elva personer i ett team fungerar, men inte optimalt, då *daily scrum*:s blir utdragna, gruppmedlemmar har svårt att veta vad resten av gruppen gör och det är svårt för *Scrum master* att se till att alla arbetar tillsammans mot sprintmålet[3]. Om teamet inte är bekväma med *Scrum*, och det inte finns någon möjlighet att dela upp uppgifterna i mindre delar som inte delar kod, är det en bra lösning att dela upp helgruppen i mindre delgrupper[3].

För att kunna arbeta optimalt i ett team ska majoriteten, men helst alla, arbeta heltid inom ett team [3]. Generellt är det bättre att ha ett team bestående av tre personer som arbetar heltid i teamet än åtta som arbetar deltid[3].

En annan aspekt av teamets uppbyggnad är huruvida gruppmedlemmar ska inneha specifika ansvarsområden och om alla gruppmedlemmar ska vara tillräckligt insatta i projektet för att kunna arbeta med samtliga delar, alltså en hög *truckfaktor*, också kallat att vara *cross functional*[5]. *8 dudes in a garage* består av ett team med specifika ansvarsområden där en person äger sin domän. Det var ett arbetssätt som de tyckte fungerade bra, men att det på gott och ont lägger mycket ansvar på den enskilda individen snarare än teamet i sin helhet[5].

En viktig del i *Scrum* är att lära sig att sätta realistiska tidsramar och sedan hålla sig till dem[3]. Det ger inte något att låta möten gå över den bestämda tidsramen eftersom mötesdeltagarna, under den återstående tiden, inte kommer uträtta mer än de redan gjort. Ett alternativ till att gå över tiden är att schemalägga resterande delen av tiden till ett annat tillfälle[3].

Att basera sin *velocity* på arbetstiden för sprinten är inte optimalt då det exempelvis finns många faktorer som spelar in, som teamets storlek, vilket gör att det inte är ett pålitligt värde[3]. Finns historiken av tidigare sprintar är det det man borde gå efter, annars är det bäst att gå på magkänsla[3].

2.3 Parprogrammering

Parprogrammering tar oftast längre tid än om de som programmerar arbetar var och en för sig. Trots detta väger fördelarna upp, som bättre samarbete, koddesign och en ökad problemlösningsförmåga[1]. I stort sett alla (95 procent) av de som arbetar i grupp tycker om sitt arbete mer och får större självförtroende jämfört med de som arbetar separat[1]. Parprogrammerare har även 15 procent färre defekter i sin kod[1].

Det finns även fördelar med parprogrammering direkt kopplade till *Scrum*, t.ex. att det finns en till person som kan se till att den som skriver koden faktiskt fokuserar på det som behövs för sprinten[3]. Å andra sidan bör man inte tvinga personer att parprogrammera, utan istället uppmuntra till det och tillhandahålla verktygen som behövs[3]. Ett alternativ till parprogrammering är *code review*, även om det inte har precis samma fördelar som parprogrammering[3].

2.4 Konfigurationshantering

Under kursens gång hölls en gästföreläsning av *Volvo*, där de gick igenom deras praxis gällande versionshantering. De använde sig utav `Configuration items (CI)` som är den filen, eller samling av filer, som ska versionshanteras. Sedan skapades det *baselines*, en specifik version av varje *Configuration Item* vid ett visst tillfälle, som användes för att skapa viktiga versioner som kan användas

för återställning, testning och *releases*[4]. Konfigurationshantering är en del av kvalitetssäkringen hos framförallt stora företag, då behovet är större när det rör sig om fler filer[4]. Ett exempel på när konfigurationshantering behövs är när man vill återställa en mängd filer till ett specifikt tillfälle då man vet att det fungerade[4].

3 Reflektion

Detta avsnitt ägnas åt reflektion kring gruppens arbete under projektet med avseende på tillämpning av ett agilt arbetssätt.

3.1 Applicering av *Scrum*

Projektet gick på halvtid vilket innebar att rekommendationen som säger att *Scrum* bör köras på heltid ej följdes. Till följd av detta uppstod vissa problem. Exempelvis behövde tiden som lades på essentiella delar inom *Scrum*, som *Sprint Planning* och *Sprint Retrospective*, halveras, jämfört med om gruppen hade arbetat på heltid. Vidare blev fördelningen av arbetstid under sprintarna ojämn, i och med att moment från parallellkursen tog upp tid. Gruppen ansåg att detta försämrade arbetsrytmen. Utöver detta började sprintarna på fredagar, vilket innebar att tiden där en sprint slutar och en annan sprint börjar blev väldigt mötesintensiv och arbetsflödet bröts direkt av helgen som inföll just då arbetet inletts.

Överlag har mycket resurser lagts på applicering av *Scrum* som helhet, då projektets syfte till stor del varit att implementera ett agilt arbetssätt. En direkt följd av detta var att det gav gruppen ett verktyg som ledde till erfarenheter, kunskap och insyn inom agila arbetssätt som kan vara bra att kunna till senare projektarbeten.

En optimal grupp för *Scrum* består av mellan fem och nio personer, medan vår grupp bestod av 12 personer. Av det följde både fördelar och nackdelar. Å ena sidan innebar gruppens storlek potentiellt mer spridd kompetens som kunde utnyttjas genom att många i gruppen hade förmågan att lösa ett brett spektrum av problem, vilket hjälpte gruppen framåt. Mer spridd kompetens gjorde alltså att fler problem kunde lösas och att fler idéer kunde diskuteras. Å andra sidan medförde det att mer tid och resurser gick åt till arbetet med att hålla *truckfaktorn* hög så att gruppen kunde vara *cross-functional*.

Däremot fanns det flera utmaningar i och med det stora antalet personer. Exempelvis blev det ibland rörigt när det gällde kommunikation, då information skulle spridas till flera gruppmedlemmar. En lösning för att minimiera problemet med en stor grupp hade kunnat vara att dela in gruppen i fler smågrupper, exempelvis två eller tre stycken. Arbetet skulle då kunna ha effektiviserats genom att varje grupp är mer eller mindre självständiga och kommunicerar sig emellan. Det kan i sig dock leda till att fördelarna som erhölls med en såpass stor grupp skulle försvinna, då *truckfaktorn* minskar.

Genom en diskussion med en grupp som implementerat *Scrum* som beskrivet ovan, det vill säga fler delgrupper, sades det att arbetet effektiviserats och att de genom det kommit längre med projektet. Däremot ställde arbetssättet till andra problem, främst administrativa och kommunikativa. De höll *daily scrum* i de mindre grupperna, och *scrum of scrums* med representanterna för de mindre grupperna. Resultatet blev att *Scrum master* fick lägga en stor andel tid på att närvara på interna möten. Att använda Scrum blev alltså svårare, eftersom uppdelningen innebar sämre

kommunikation och mindre fokus på administrativt arbete (exempelvis *Product Backlog*). Av det följde att fokuset flyttades från *Scrum* till att komma långt med projektet.

Med tanke på den gruppstorlek som förespråkas av Kniberg (se avsnitt 2.2), är det möjligt att användning av *Scrum* för ett projekt med mindre grupper skulle ha resulterat i en bättre implementation. Däremot lär man sig inte lika mycket om problematiken som uppstår under dessa projekt genom väldigt stora arbetsgrupper. Det kan leda till att man är mindre förberedd för liknande problem i framtiden.

3.1.1 Roller

Inför första sprinten beslutades vilka roller som skulle användas, samt vilka gruppmedlemmar som skulle tillsätta dessa roller. Tanken var även att under varje sprint arbeta i ett antal små fasta grupper, där en grupp arbetade med en *user story*. *Scrum master* och projektledare ansvarade för att överse detta. Rollerna skulle gälla till och med den tredje sprinten, varefter de skulle utvärderas. Ursprungligen såg rollerna ut enligt upplägget nedan.

Scrum master: Leda möten och ansvara för *Scrum*-upplägget. Utföra bilbyten. Se till att alla gruppmedlemmar har något att arbeta med.

Projektledare (& *SAMO*): Ansvara för kritiska beslut som gäller hela projektet då demokratiska beslut vore opassande eller omöjliga. Ta över ledning av möten i händelse av *Scrum master*:s frånvaro. Ansvara för att se till att alla i gruppen mår bra och är delaktiga. Bildar tillsammans med kundansvarig ett kundteam som ansvarar för kommunikationen mellan teamet och kunden.

Kundansvarig: Ansvara för dialogföring med kunden. Samt agerar som kund i situationer där kunden ej kan rådfrågas.

Sekreterare: Anteckna under möten, samt ansvara för att renskriva och ladda upp mötesprotokoll.

Det ursprungliga upplägget av rollerna skiljde sig en del mot hur rollfördelningen såg ut mot slutet. I samband med utvärderingen efter de första tre sprinterna ändrades rollerna något. Vissa roller bytte innehavare. Därtill valdes ytterligare en sekreterare in pga. behovet av två sekreterare.

Det ansvar *Scrum master* tog på sig inkluderade det administrativa arbetet att tillsammans med kundansvarig ta hand om gruppens *scrum board*:s, samt ansvar för den huvudsakliga kontakten med andra grupper och överlämning av bil. Efter projektets skede märktes det att *Scrum master*:s ansvarsområde var alltför stort. Detta hade kunnat lösas genom att avlasta *Scrum master* med, till exempel, en bilansvarig som skulle sköta byte av bil och kommunikationen som det innebar.

Kundansvarigs ansvarsområden var emellanåt otydligt och hade kunnat förbättrats av att kundansvariga representerade produktägaren gentemot teamet fullt ut vid exempelvis *Sprint Planning*.

Sekreterarrollen fungerade bra för det mesta, det fanns dock en oklarhet i om det var sekreterarens roll att renskriva protokoll och ladda upp dessa, samt när detta skulle göras, i så fall.

Det observerades att de mindre grupperna som ursprungligen var tänkta att vara fasta i praktiken blev mer flytande, och sammansättningen av dessa varierade mellan arbetstillfällen och *task*:s. Att det blev så kan bero på att gruppmedlemmar varit frånvarande, eller att arbetet med en *user story* helt fastnade på grund av yttre faktorer. En annan anledning kan vara att det helt enkelt inte i verkligheten är lika lätt att fastställa statiska grupper utan kunskap om varje individs kunskap och expertis.

En alternativ uppdelning av roller är att ha satta ansvarsområden för varje person, likt upplägget som exempelvis *8 dudes in a garage* använder sig av. En fördel är att det är väldigt tydligt vem som är ansvarig för vad och att det finns någon som på egen hand kan ta beslut inom sin domän om det krävs. En nackdel är att hela teamet inte kan vara insatta i helheten på samma sätt och kan arbeta inom olika domäner lika lätt. En anledning till att det hade fungerat mindre bra i detta projektet är att det saknades den expertis som krävs för att en individ ensam ska kunna vara ansvarig för en domän. En annan anledning är att teamet föredrog att byta mellan olika områden.

3.1.2 Socialt kontrakt

Teamet kom överens om ett socialt kontrakt i vilket det bland annat skulle definieras hur gruppen skulle arbeta och vilka verktyg som skulle användas. Detta kontrakt skrevs i samband med inlämningen D1.

En punkt som har förändrats sedan kontraktets skrivning var systemet med fikaprickar, som innebar att om man kom sen bjöd man senare gruppen på fika, togs bort och under ett *Retrospective*-möte utvärderades det sociala kontraktet. Då framgick det att formuleringarna inte helt stämde överens med gruppens samlade mening, därmed modifierades det sociala kontraktet.

Att formulera ett socialt kontrakt för att kunna komma överens om gemensamma riktlinjer och värderingar har underlättat gruppens samarbete. Kontraktet har även varit till nytta då referering till de gemensamt beslutade riktlinjerna har gjorts vid uppkomna missförstånd. En förbättring av det sociala kontraktet hade kunnat vara att inte bara definiera *workflow* för *Git* utan även för andra verktyg, i detta fall *Trello*. En alternativ väg hade varit en mer regelbunden uppföljning av de sociala kontraktet, exempelvis vid varje *Sprint Retrospective*. Detta tyckte några i gruppen hade kunnat vara en förbättring, men de flesta ansåg inte att det hade lett till någon märkbar skillnad.

3.1.3 Samarbete

Samarbetet fungerade generellt bra. Många gruppmedlemmar uttryckte en stor tillit till gruppen, vilket är positivt ur en samarbetssynpunkt och tyder på väl fungerande kommunikation. Det uppstod dock två större situationer där samarbetet fungerade mindre bra. Den ena handlade om ett missförstånd i kommunikationen, framför allt i samband med användande av *Scrum Boards*, där flera arbetade parallellt med samma uppgift.

Den andra händelsen uppstod under bearbetningen av D3 där det även här handlade om en miss i kommunikationen där textens innebörd ändrades utan att gruppen tillfrågades. Detta ledde i sin tur till att gruppen inte var överens om den nya innebörden.

Dessa ansågs vara enstaka företeelser och inte en direkt konsekvens av brister i gruppens arbetssätt. Detta löstes genom en gemensam diskussion där de berörda gruppmedlemmarna kom fram till att de själva misslyckades med sin kommunikation samt att de skulle förbättra användandet av de kommunikationshjälpmedel gruppen använder sig av. Därtill skulle tydligheten för de riktlinjer angående hur man arbetar, till exempel för hur man som grupp producerar en text även om det bara är ett par gruppmedlemmar som arbetat med senaste utkastet, förbättras. Detta genomförde gruppen under rapportskrivningen då det bestämdes, i helgrupp, rapportens disposition samt innehåll.

3.1.4 Använda metoder

Utöver den av *Scrum* definierade arbetsmetodiken, tillämpades ett antal av gruppen utvalda metoder. Dessa metoder formulerades vid starten av projektet med målet att de skulle ge god projektstruktur och öka gruppens flöde av kvalitativ kod.

3.1.4.1 Parprogrammering

Vid diskussioner i början av projektet fastslogs att gruppen skulle använda sig av parprogrammering. Det bestämdes även att endast fick ske i arbetsgrupper av två eller fler personer, och därmed aldrig enskilt.

Utöver fördelarna som nämns i teorikapitlet bedömde gruppen att användande av parprogrammering skulle medföra en höjning av truckfaktorn och en minskad risk för bildning av flaskhalsar orsakade av gruppmedlemmars frånvaro. En nackdel som diskuterades var den potentiella förlusten av möjligheten att nyttja två personers arbetskraft. Denna nackdel bedömdes dock ha låg relevans för projektet i fråga, då det konstaterades att antalet *task*:s gällande programmering, som kunde planeras in åt gången, inte skulle räcka till att sysselsätta gruppmedlemmar individuellt.

Under sista sprinten släpptes dock parprogrammeringen lite grann och folk arbetade mer ensamma. Detta berodde på att gruppen behövde producera den sista funktionaliteten på kort tid, vilket innebar att parprogrammering blev nerprioriterat.

Ett alternativ till denna strikta tillämpning av parprogrammering skulle kunna ha varit att jobba helt och hållet enskilt. Fördelar med det skulle kunna vara mer effektivt arbete, förutsatt att den som arbetar har tillräcklig kunskap om uppgiften som ska utföras, då det inte skulle gå åt tid till att inviga en mindre insatt gruppmedlem.

En annan fördel med att inte använda strikt parprogrammering skulle kunna vara att hänsyn inte behöver tas till kunskapsskillnader mellan personer. Till exempel kan en situation uppstå där arbetsfördelningen i paret blir väldigt ojämn eftersom en av de två inte förstår. Dock känner gruppen att fördelarna med parprogrammering tar ut nackdelarna och att parprogrammering var ett viktig verktyg under projektets gång.

3.1.4.2 Daily Scrum

Varje arbetspass inleddes med 10-15 minuters *Daily Scrum*. Detta gjorde det tydligt för gruppen vad varje enskild gruppmedlem hade åstadkommit och för närvande arbetade med. Det som emellertid visade sig vara kanske mest värdefullt med genomgången var den tydlighet det gav för varje gruppmedlem var dennes arbetskraft kunde göra mest nytta, samt att en sysslolös gruppmedlem alltid blev tilldelad en arbetsuppgift i samband med genomgången. På så sätt arbetade gruppen för att samtliga skulle känna sig behövda och inkluderade. Detta kan ses som extra viktigt i en så pass stor grupp som denna, där risken är större att som individ komma bort och inte veta vad som borde arbetas på efter en slutförd uppgift. Utveckling framskred fort inom flera områden samtidigt, varför den kontinuerliga uppdateringen av vad som skedde på olika områden blev viktig för att hålla samman gruppen.

Ett alternativ till *Daily Scrum*-möten hade kunnat vara en anslagstavla på *Trello* där alla uppdaterade vad de arbetade på för tillfället, med möjlighet att till exempel visa om man behövde hjälp eller om man fastnat. Den sysslolöse kunde vända sig till *scrum master*, som har ansvar för att

alla alltid ska ha något att göra. Arbete enligt denna modell hade haft fördelen att den tid som läggs på att närvara vid *Daily Scrum* hade sparats in. Dock, då hade varje gruppmedlem tvingats ägna tid och tanke åt att hålla denna anslagstavla uppdaterad, och den hade riskerat att komma bort bland andra anslagstavlor på *Trello*. *Daily Scrum*-mötet har å andra sidan fördelen att trots att det är så kort, ger det effektivt en överblick över vad som arbetas med och hur gruppen i stort ligger till i processen.

3.1.4.3 Gemensamma arbetspass

Under den första planeringsfasen bestämdes det att hela gruppen under ordinarie omständigheter skulle sitta tillsammans under varje arbetspass, då detta skulle öka effektiviteten. På så sätt skulle möjligheten alltid finnas att kommunicera med andra i gruppen på plats utan fördröjningar, och uppkomna frågor rörande hela gruppen skulle kunna tas upp direkt. Detta fungerade bra och uppskattades av hela gruppen.

Däremot medförde användandet av metoden med att sitta hela gruppen tillsammans ett antal komplikationer. Ett av de största problemen var bristen på tillräckligt stora arbetsytor att arbeta tillsammans på. En direkt konsekvens av bristen på utrymme var att arbetspassen blev röriga och högljudda. För att lösa detta fattades ett gemensamt beslut om att tänka på att hålla ljudnivån nere, och att som enskild gruppmedlem säga ifrån om ljudnivån blev för hög. En annan lösning hade varit att sitta uppdelade i mindre delgrupper som då hade kunnat löst problemet med arbetsytor och röriga arbetspass, dock på bekostnad av smidig kommunikation och effektiva diskussioner.

3.1.4.4 Planerade arbetspass

Det bestämdes att sprintens arbetspass skulle bokas in vid planeringsfasen för varje sprint. Detta för att tillhandahålla en gemensam struktur i syfte att underlätta tidsdisponering och kommunikation i gruppen. *Daily Scrum* blev därmed något som blev naturligt lätt att införa i början av varje arbetsdag. Likaså underlättade denna planering av arbetspass tillämpandet av metoden ”Gemensamma arbetspass”. Det bestämdes också att samtliga gruppmedlemmar skulle hålla sig till dessa arbetspass och inte arbeta utanför de inplanerade tiderna, för att hålla stressnivån låg.

Många gruppmedlemmar har uttryckt att denna metod bidragit till mindre stress tack vare bra struktur och planering av kursen. Det har dessutom varit lätt att planera in dessa arbetspass, då majoriteten av gruppmedlemmarna har haft likadant schema. En positiv aspekt med att ha fördefinierade arbetspass var att det förenklade disponering av tid för både projektet och andra åtaganden gruppens enskilda medlemmar hade utöver kursen.

3.1.4.5 KPI som metod för förbättring

Utöver det huvudsakliga målet med att använda *KPI*:er användes även dessa som metoder för att gruppdynamiken skulle förbättras. Två av våra *KPI*:er användes särkilt mycket till förbättringsarbetet. (*KPI*:erna i sin helhet finns beskrivna i avsnitt 3.5).

Den första mätte hur väl samtliga gruppmedlemmar kände att de hade något att göra. Utifrån resultatet kunde gruppen tillsammans på *Sprint Retrospective* diskutera varför värdet blev som det blev och hur det kunde förbättras. Utifrån diskussionen sattes det upp nya riktlinjer vilkas mål var att höja värdet till efterkommande sprint. Värdet förändrades inte på det sätt som gruppen

strävade efter, utan varierade enbart med små ökningar varje sprint. Den stabila nivån skulle kunna bero på de tekniska svårigheter projektet innebar och att flera i gruppen därmed kände en viss osäkerhet över hur de tekniska problemen bäst kunde lösas. En annan möjlighet är att potentiella brister i gruppens arbetssätt ännu inte lokaliserats. Användadet att detta *KPI* kan ha bidragit till gruppens goda sammanhållning då gruppen hade ett värde som kunde indikera ifall någon gruppmedlem hamnade efter, i vilket fall gruppen då kunde försöka åtgärda detta. Överlag ansåg gruppen att diskussionen kring *KPI*:et hjälpte gruppen framåt även om *KPI*-värdet i sig inte ökat markant.

Genom det andra *KPI*:et övervakades den tekniska skulden. Det främsta syftet med *KPI*:et var att se till att det inte ackumulerades obetald teknisk skuld. Regelbunden uppföljning av värdet underlättade strukturering av arbetet med att betala av den tekniska skulden. Gruppen ansåg att själva *KPI*-värdet i detta *KPI* inte var det viktiga då viss teknisk skuld är att vänta. Däremot var diskussionen kring *KPI*-värdet mer värdefullt då gruppen fick en inblick i den tekniska skuld som fanns och genom att prata om skulden regelbundet kunde detta bidra till att minska teknisk skuld proaktivt, då gruppmedlemmar då försöker undervika onödig sådan.

Rätt *KPI* kan vara ett värdefullt verktyg för att övervaka är att det kan indikera hur något ser ut, till exempel hur mycket teknisk skuld man har. En nackdel med att använda *KPI*:er är att det finns en risk att välja ett *KPI* som inte ger en rättvis bild. *KPI*:et som beräknades med hjälp av en enkät var beroende av att alla gruppmedlemmar deltog för att ge en så rättvis bild som möjligt, vilket inte alla alltid gjorde och därmed eventuellt gav ett något felaktigt resultat. En annan nackdel är att det finns en risk att det inte används som en indikator utan mer som ett mål att uppfylla och därmed ändrar sitt beteende för att uppnå målet. Om ett team till exempel har som *KPI* antal rader kod skulle det kunna göra att teamet skriver fler rader kod för att det ger ett högre *KPI* snarare än att det förbättrar koden. Rätt använt är *KPI*:er ett bra verktyg för att kunna få en bild av aspekter av ett projekt, men då är det viktigt att det just används på rätt sätt.

I allmänhet anser gruppen att *KPI*:er har varit ett användbart verktyg för att övervaka olika aspekter av processen. Dock observerades det att *KPI*-värdena inte alltid stämde överens med vad som var den allmänna upplevelsen hos gruppen. Exempelvis indikerade *KPI*:et för tydliga arbetsområden att det fanns ett antal gruppmedlemmar som var mindre nöjda med tydligheten i distributionen av arbete, samtidigt som den allmänna åsikten som kom fram under möten var att det fungerade bra.

3.1.5 Distribution av tid

Vid den inledande planeringen beslutades att gruppen skulle lägga ungefär 20 timmar i veckan på projektet. I dessa 20 timmar räknades föreläsningar och schemalagda pass in, vilket i praktiken innebar att effektiv arbetstid uppgick till 10-15 timmar, beroende på hur mycket föreläsningstid och schemalagda pass som var inplanerad under den aktuella sprinten. Det var varje enskild gruppmedlems ansvar att närvara och arbeta under planerade möten och arbetspass. En allmän observation inom gruppen var att tidsplanen under vissa sprintar brast lite så att den totala arbetstiden blev kortare än vad som var tänkt. En idé som uppkom i efterhand var loggning av gruppens arbetstid för att i realtid kunna få en tydlig överblick över hur väl tidsplanen följdes. En annan idé som uppkom var användning av ett system för individuell tidsrapportering, där varje enskild gruppmedlem skulle rapportera sin nedlagda arbetstid, så att gruppen skulle kunna få bättre överblick över spenderad tid.

Vidare gjordes en överenskommelse om att hålla allt arbete på projektet mellan klockan 8:00 och 17:00 på vardagar. Detta främst i syfte att förebygga stress orsakad av oregelbunden och utspridd arbetstid, men även för att underlätta inplanering av gruppmöten. Denna bestämmelse har hållits fast vid och följts under hela projektets gång, och gruppen upplever att den effektivt bidragit till att hålla stressnivån låg och underlätta planering. Funderingar finns gällande huruvida beslutet att endast arbeta mellan klockan 8:00 och 17:00 på vardagar kan ha begränsat hur långt gruppen kunnat avancera i arbetet, men även om att beslutet kan ha bidragit till högre effektivitet i arbetet tack vare medlemmars reducerade stressnivå.

Inledningsvis lades relativt lite tid på administrativt arbete, såsom underhåll av *scrumboard*, till förmån för tekniskt arbete. Detta var dock något som gruppen enades om att ändra efter den första sprinten i hopp om att kunna arbeta mer effektivt med bättre överblick och struktur. Det är gruppens allmänna uppfattning att det under projektets gång lagts en allt större andel tid på administrativt arbete, och att detta haft en direkt positiv inverkan på effektiviteten.

Vid diskussion med en annan grupp framkom att denna andra grupp tillämpade indelning i mindre delgrupper, vilka administrerade sina uppgifter mer individuellt. Dessa grupper hade tydliga arbetsområden och tanken var att detta skulle möjliggöra nyttjande av spetskompetens. Delgrupperna bestod under hela projektet, med endast mindre utbyten av gruppmedlemmar sinsemellan. De lade också mindre fokus på administrativt arbete till förmån för högre potentiell effektivitet. Det är möjligt att detta arbetssätt även hade gynnat arbetet med `onTruck`, men stred mot gruppens gemensamma mentalitet (se avsnitt 3.1.6). Det kan dock diskuteras att gruppen blev naturligt uppdelad i viss mån. Då valen av arbetsuppgifter var frivilliga fokuserade varje medlem mer på vissa områden, något som är ganska svårundvikligt och förmodligen inte negativt i sig.

Eftersom gruppen valt att ta fasta på principen om varje gruppmedlems egna ansvar, låg vid planering och utvärdering fokuset på gruppen som helhet och inte på varje enskild medlems arbetstid. Något som dock kom fram efterhand var att vissa specifika roller hade haft ett oproportioneligt stort lass att dra jämfört med övriga roller. Exempelvis har rollen som *Scrum master* upptagit mycket tid (se avsnitt 3.1.1).

3.1.6 Ansträngning, *velocity* och nedbrytning av uppgifter

Det bestämdes att under arbetstillfällena så skulle det bara vara fokus på arbete och inget utanför det. Efter några veckor infördes en rast varje timma på 10 minuter. Detta uppfattades leda till mer fokus, dock återstår frågan om det blev mer effektivt, då arbetstiden blev 1/6 kortare. Oavsett om gruppen har legat i fas med projektet så har samma tidsschema följts. Gruppen bedömde att stressnivån generellt skulle minska om ett tidsschema följdes, även om detta potentiellt kunde innebära att gruppen halkade efter.

Gruppen valde att i första hand lägga fokus på välmående och lärande. Istället för att delegera arbetet till de personer som var mest lämpade eller hade bäst kunskap på området, så fick de som var mest intresserade göra uppgiften. Då projekt var förhållandesvis kort skulle detta kunna vara ett mindre effektivt arbetssätt. Dock skulle en högre *truckfaktor* tillkomma, och medlemmar bli mer delaktiga i hela projektet. Om projektet hade gjorts på en arbetsplats så skulle arbetssättet kunna vara mer effektivt i längden, då målet över tid med att ha en hög *truckfaktor* är att många medlemmar ska ha mycket kunskap inom flera områden. Det märktes att gruppens upplägg var bra då samtliga gruppmedlemmar var eniga i att projektet skulle kunnat hålla på en läsperiod till utan problem.

Gruppen beslutade tidigt att *velocity*:n skulle baseras på antalet effektiva arbetstimmar gruppen hade tänkt att arbeta under sprinten, se *Distribution av tid*. De effektiva arbetstimmarna multiplicerades med antalet gruppmedlemmar för att få den totala mängden timmar. Därefter halverades värdet, då gruppen ansåg att detta bättre representerade den faktiska arbetskapaciteten.

Även fast det inte är rekommenderat att använda ett fast värde för *velocity*:n så grundade gruppen *velocity*:n i arbetstimmar, vilket gjorde att gruppmedlemmarna snabbt kunde begripa relationen mellan *velocity* och *user story* -värde. Då projekttiden var relativt kort kunde ett abstrakt värde ha försvårat förmågan att förstå innebörden av en enhet av värdet. Eftersom gruppen saknar tidigare erfarenhet av *Scrum* var det svårt att upskatta *velocity* på känsla, varför teamet bestämde sig för att ha arbetstimmar som bas.

Emellertid med alla modifikationer som gruppen gjorde på värdet så blev *velocity*:n fortfarande till viss del abstrakt och diffus. Användning av ett helt abstrakt värde hade kanske bidragit till fler beslut baserade på 'magkänsla', då man undgår att basera sitt beslut på något konkret. Detta kunde resultera i att hålla tiden för planeringsmöten inom en timma, något som gruppen generellt hade svårt med. Däremot vid användning av ett helt abstrakt värde kan man hamna i tankesättet 'om *velocityn* är 10 grönsaker och gruppen har 20 timmar tillhands så motsvara en grönsak två timmar' och då är man tillbaka i ruta ett igen.

I början på varje sprint bröt *Scrum master* med en assistent ner alla uppenbara *tasks* till mindre, mer lätthanterliga *tasks*. I början hade gruppen svårt att göra något vettigt under tiden nedbrytningen skedde. Efter någon sprint så kunde gruppen börja jobba på teknisk skuld och annat administrativt arbete samt skriftliga inlämningar. I början bröts *tasks* ner som strikt vertikala skivor. Då detta blev ett tidsödande och ofta komplicerat sätt att göra nedbrytningen på ändrades tillvägagångsättet. Istället för att dela upp *tasks* i vertikala skivor så delades de in in mer horisontella indelningar. Detta medförde att nedbrytningen blev lättare och mer naturlig men också att ett beroende mellan vissa *tasks* uppstod.

3.2 Praxis för användning av nya verktyg och teknologier

För att jobba så effektivt som möjligt och slippa "återuppfinna hjulet", försökte gruppen tidigt i kursen att samarbeta med de andra grupperna. Detta visade sig dock vara svårt i början, då ingen ville dela med sig. Detta skulle kunna bero på gamla vanor från tidigare i utbildningen då det fanns väldigt strikta regler kring just delning av arbete. I många universitetskurser ses det som rent fusk att använda sig av andras arbete. Det skulle även kunna bero på att vissa grupper ser projektet som en tävling, där man vill bevisa sig bättre än alla andra. Ju närmare slutet av projektet desto mer delade alla grupper med sig av deras arbete. Anledningen till detta skulle kunna vara att föreläsaren kontinuerligt informerade alla grupper om att detta inte är en tävling, utan att samarbete mellan alla grupper är något önskvärt och positivt. Det skulle också kunna ha berott på att vissa grupper inte behövt hjälp av andra förens i slutet, och på grund av det inte tyckt att de vunnit något på att dela med sig tidigare.

Gruppen har under projektet fått en ny *MOPED* en gång i veckan. Detta var ett problem då den nya *MOPED*:en ofta var trasig. Det fick läggas mycket tid på att felsöka istället för att arbeta. Om inte *MOPED*:en hade behövts bytas varje vecka skulle det kunnat leda till ökad effektivitet. Dessutom så var det möjlig att när *MOPED*:en slutade fungera så påverkades gruppens motivation och energinivå negativt, vilket i sin tur kunde bidra till ett mindre effektivt arbetspass. Om gruppen hade haft samma *MOPED*:en hade det varit möjlig att arbete runt problemet, men eftersom bilarna byttes ut uppstod konstant nya fel och problem.

Volvos gästföreläsning gick igenom hur viktigt det var med versionshantering. Gruppen använde sig av *Git* som versionshanterare, men utnyttjade inte baselines, vilket kanske hade underlättat om projektet varit längre. 'Baselines' skulle kunna underlätta om fel uppstår, till exempel om *MOPED*:en på demonstrationen ej skulle vara kompatibel med mjukvaran, så skulle det kunna fungera med en äldre *baseline*. Det skulle även kunna underlättat om gruppen hade en *baseline* för kommunikationen mellan mobil-applikationen och *MOPED*:en, så man visste att de alltid skulle kunna kommunicera med varandra.

En möjlig *baseline*-implementation hade varit en viss konfiguration och version av *MOPED*-filerna. Denna skulle fungera ihop med en viss iteration av gruppens kod. Detta skulle ge fördelen att om något började krångla kunde det snabbt avgöras om operativsystemets konfiguration, eller om koden på *MOPED*:en, var del av problemet. Gruppen hade också kunnat snabbt återvända till en tidigare *baseline* om så krävdes, istället för att ändra enstaka filer, som snabbt kan bli väldigt rörigt.

Väldigt tidigt i projektet skrev gruppen ett arbetsflöde för *Git*. Ett problem som uppstod var att mallen aldrig diskuterades i helgrupp, så olika personer tolkade arbetsflödet olika. Det var först efter fyra sprintar som detta problem togs upp. Problemet borde enkelt ha kunnat undvikits om gruppen hade gått igenom arbetsflödet tillsammans, eller om gruppmedlemmar hade pratat med den personen som skrev arbetsflödet istället för att själv tolka oklarheter.

En praxis i vårt git-arbetsflöde var *peer review*. Detta är att flera personer måste godkänna en *pull request* innan den godkänns. Detta bidrog till att filtrera ut dålig kod, men riskerade att sänka hastigheten på projektet om medlemmar skjutit upp att kolla igenom *pull requests*.

3.3 Sprint reviews

Sprint review:s har framförallt stått för kommunikationen med produktägaren. *Review*:en har bestått av en produktdemostration följt av att kundansvariga diskuterat och kommit överens med produktägaren om nästa *sprint*:s mål. Till en början var inte hela teamet delaktiga under detta utan enbart kundansvarig och projektledaren medverkade. Efter ett par sprintar beslutades det att hela teamet skulle vara närvarande då det ansågs vara en fördel att alla var där, eftersom det rör alla och ger alla en möjlighet att prata med produktägaren. Produktägaren var dock inte alltid närvarande, och då tog någon annan utanför teamet över den rollen, vilket försvårade arbetet.

Något som hade förbättrat sättet gruppen hade *Sprint Review*:s på, hade varit om tiden utnyttjades bättre och om gruppen hade förberett konkreta frågor för att få fram den informationen som behövdes. Att rollen kundansvarig representerade produktägaren fullt ut gentemot teamet hade kunnat förbättra *Review*:s då det kanske är ännu mer tydligt för kundansvarig vilka ansvarsområden det innebär och att det krävs information om bland annat prioritering för att sedan kunna representera produktägaren.

3.4 Sprint retrospectives

Sprint retrospectives hölls under samtliga sprinter. Hur de såg ut från början skiljer sig en del från hur de såg ut i slutet tack vare en förändring i mötets struktur som gjordes i andra och tredje sprinten. Mötets upplägg var från början att diskutera vad som gått bra och mindre bra med sprinten, de förbättringar teamet vill utföra i nästa sprint, samt att KPI:ernas värde för sprinten diskuterades. Mötena protokollfördes, vilket gjorde att det var lättare att gå tillbaka till det som

reflekterats över. Det fanns flera problem med upplägget, bland annat att det upplevdes som rörigt och blandades ihop med *Sprint Review*.

Efter de förbättringar som gjordes med mötets struktur upplevdes mötet som mindre rörigt och tydligare definierat. En viktig förbättring var att antalet förbättringar som skulle utföras i nästa sprint begränsades till 2-3 stycken, samt att det skedde en uppföljning av dessa på nästa *Sprint Retrospective*. Att ha färre förbättringar, och framförallt att följa upp huruvida de efterföljs eller inte, gjorde det mer sannolikt att de faktiskt efterföljdes.

En annan förbättring som infördes var en 'må bra'-runda där gruppen antingen gick varvet runt och alla gruppmedlemmar fick utrymme att berätta hur de mådde, eller i vissa fall att SAMO fick ordet och lyfte eventuella saker som uppkommit under sprinten. En sådan sak var utvärdering av fikaprick-systemet, enligt vilket en medlem fick bjuda på fika efter att ha kommit sent, som avskaffades då det inte fungerade som det var tänkt. Detta eftersom det blev för stort fokus på att räkna fikaprickar och det gjorde inte att fler kom i tid. Metoder som använts under sprinten började även utvärderas under mötet, vilket gjorde att gruppen kontinuerligt reflekterade kring de metoder som användes. Ytterligare en förändring som gjordes var att det sattes en tidsgräns för mötet på en timme, men det ansågs viktigare att avsluta mötet än att hålla hårt på den gränsen.

Det finns flera olika sätt att ha *Sprint Retrospectives* på och det är svårt att säga vad som är det bästa sättet, dessutom kan det variera mellan olika team. En fundering som kan uppkomma är om det är väsentligt att ha *Sprint Retrospectives*, eftersom man inte vill spendera för mycket tid med administrativa uppgifter enligt agila arbetssätt. Å andra sidan är det under detta möte som teamet får tillfälle att förbättra sig, vilket borde väga upp för att det tar tid från annat. Det kan därför vara viktigt att teamet själva får strukturera sina *Retrospectives* så att teamet får ut något av det och framför allt får möjlighet att genomföra förbättringar. I detta projekt var teamet eniga om att *sprint retrospectives* var bra, så detta var inget problem. Att tidsbegräna mötena hårdare hade kunnat göra teamet mer effektiva under mötena och gett mer tid till utveckling.

3.5 KPI:er

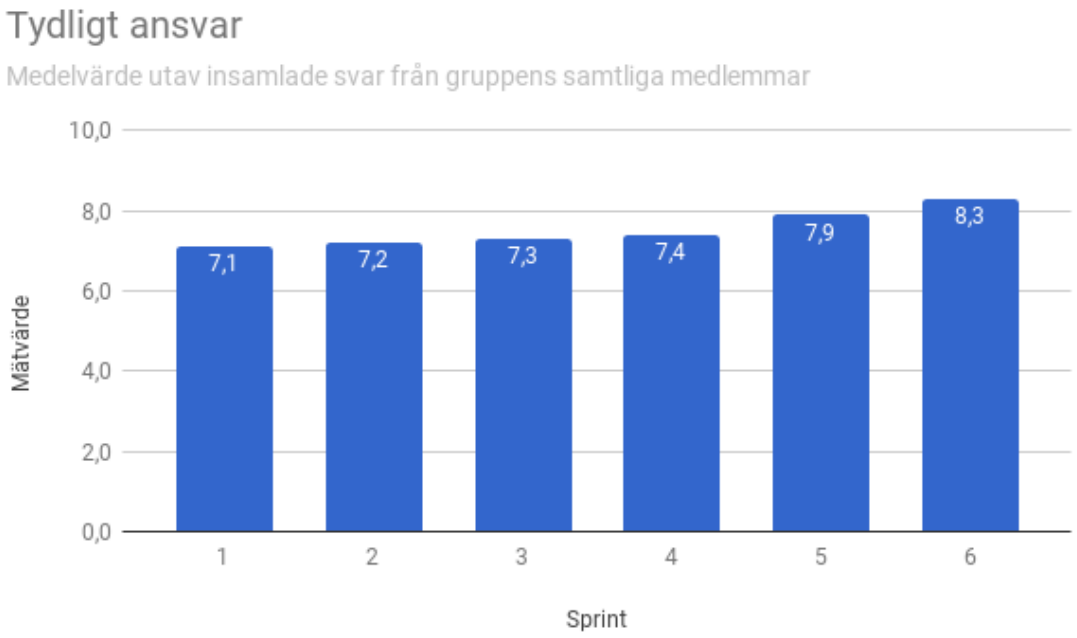
Våra KPI:er var:

Tydliga ansvarsområden
Medelvärdet av gruppens svar på hur väl man kände att man visste vad man skulle göra (skala 1-10).

Låg *Technical Debt*
Ett viktat medelvärde av hur många pull request*s,* todo:s *och refaktoreringar som finns kvar efter sprintens slut.*

Nöjd kund
Kundnöjdheten bestämdes utifrån hur stor andel av user stories *som avklarats. Dessa värden låg mellan 1-10.*

3.5.1 Tydliga ansvarsområden



Figur 1: KPI-diagram: *Tydliga ansvarsområden*

KPI:et *Tydliga ansvarsområden* var tänkt som ett verktyg som gav gruppen möjligheten att se till att gruppen som helhet har förståelse om vad som ska göras. Det fungerade även som ett verktyg för förbättring; var värdet lågt kunde det diskuteras om varför det var lågt och vad som kan förbättra det till nästa sprint.

Tanken var att om värdet förbättrades, förbättrades även gruppdynamiken och effektiviteten i verkligheten. Detta eftersom värdet var direkt kopplat till huruvida varje gruppmedlem kände att den visste vad som skulle göras. Strategin som valdes för detta var smart planering", vilket innebar att gruppen lade ner mycket tid på att planera och kommunicera med varandra så mycket som möjligt. Det hänger även ihop med att gruppen alltid satt tillsammans och arbetade. På så sätt kunde samtliga gruppmedlemmar få så mycket hjälp som möjligt med vad som skulle göras härnäst.

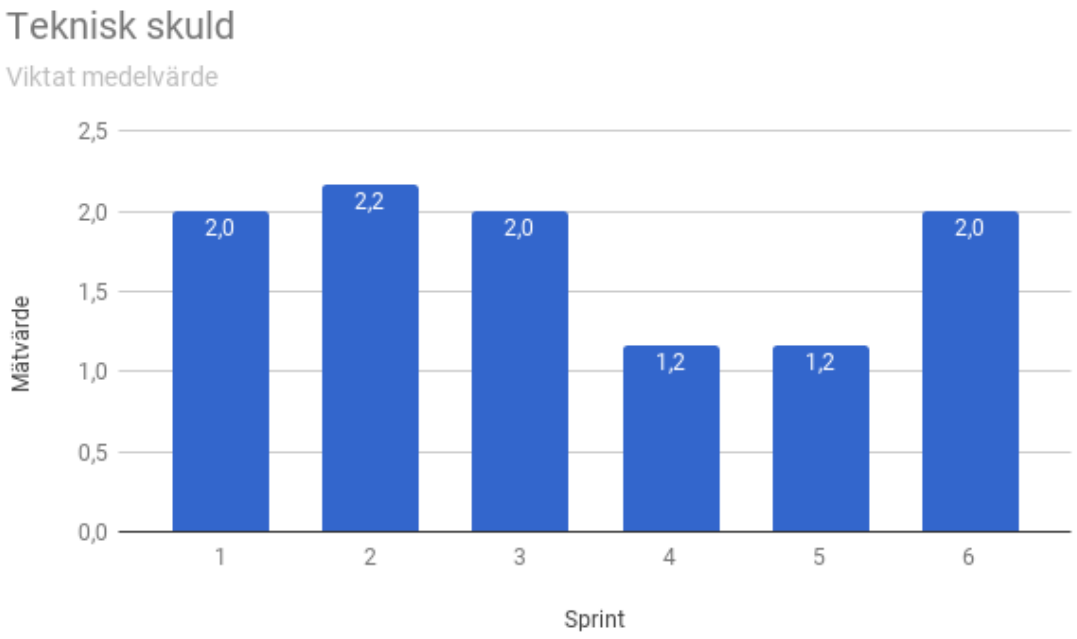
Att just detta valdes som KPI berodde på lärdomarna från *LEGO*-övningen. Gruppen kände att medlemmarna inte alltid hade någon tydlig arbetsuppgift och att medlemmar ibland arbetade ineffektivt på samma uppgift vilket skapade flaskhalsar. Detta var något gruppen ville förebygga och förhindra under det riktiga projektet, genom strategin och KPI:et.

Under projektet diskuterades värdet och åtgärder sattes in för att öka det. Till exempel tillgavs *Scrum Master* ytterligare en uppgift, vilket var att kontinuerligt se till och kolla med gruppmedlemmarna ifall de faktiskt hade något konkret att göra. I själva verket ökade inte KPI-värdet tillräckligt mycket för att se ifall åtgärderna faktiskt hjälpte. Ökningen skulle istället ha kunna bero på yttre faktorer som mer förståelse om vad projektet gick ut på ju mer tid som spenderades på det. Att ökningen inte blev så hög som förväntat kan i sin tur bero på förvirringen kring att arbeta med

ett agilt arbetssätt för första gången med så många gruppmedlemmar. En gruppdiskussion om arbetssättet och Scrum i allmänhet skulle kanske ha ökat resultatet mer istället. Däremot skulle projektet ha kunna sett ut som Lego-övningen gjorde; förvirring, flaskhalsar och ineffektivt arbete, ifall gruppen inte antog strategin och KPI:et.

En annan synpunkt kan vara vårt val av värde. Medelvärdet var intressant eftersom att det gav en helhet för gruppen, men även spridningen av mätdatan var intressant. Det var vanligt att högsta svar var 10 eller 9 och att lägsta svar var 4 eller 5. Det kan tolkas på sättet att under sprinterna fanns det folk som alltid visste vad de skulle göra, och folk hade sämre koll på vad de skulle göra. Eftersom att enkäterna var anonyma visste ingen vem som svarade vad, och ingen som konsekvent hade satt låga värden trädde fram och uttryckte något missnöje. Ifall gruppen hade diskuterat om varför personerna alltid svarat lågt kanske det hade kommit fram lösningar till detta. En följd av det hade kunnat vara ett eventuellt ökat medelvärde.

3.5.2 Låg teknisk skuld



Figur 2: KPI-diagram: *Teknisk skuld*

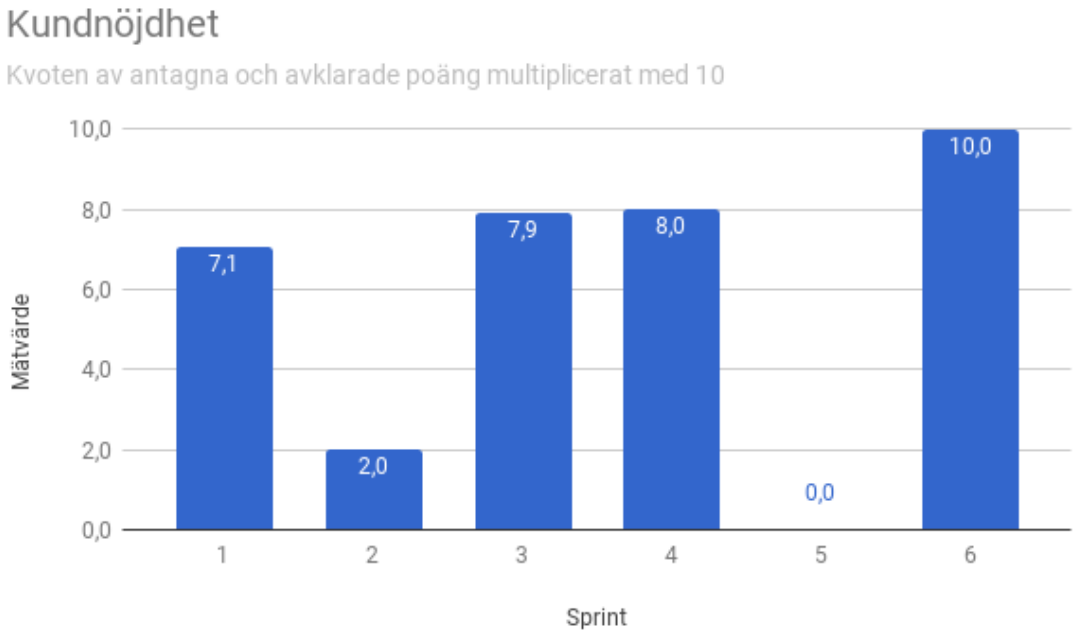
KPI:et *Låg teknisk skuld* hade huvudsakligen två syften. Det första var att gruppen under projektets gång alltid skulle vara medveten om vad som behövde göras utöver nya tasks och ha god koll på den tekniska skulden. Detta ansågs nödvändigt eftersom teknisk skuld annars lätt kan glömmas bort. Det andra syftet var att gruppen genom att veta hur mycket teknisk skuld som fanns effektivt skulle kunna reducera den. De två syftena tillsammans bidrog alltså till en medvetenhet om att det fanns teknisk skuld att åtgärda, och att man genom diskussion kan åtgärda men även förminska teknisk skuld i framtiden. På så sätt gav detta KPI gruppen en metod för att inte hamna efter med saker som anses ”jobbiga”, exempelvis koddokumentation eller refaktoreringar.

Under början av projektet när gruppen skapade strategier, var ”hantering av teknisk skuld” en av dem. Genom att ha det som strategi lyckades gruppen bevara fokus på *bra* kod som är komplett och utbyggbar (vilket minimerar teknisk skuld).

Överlag förminskades den tekniska skulden under projektet, förutom för den sista sprinten där den ökade. En lärdom var att detta var ett svårt KPI att utvärdera. Ökningen och minskningen behöver inte bero på strategin och KPI:et i sig, utan andra faktorer. Dessa faktorer kunde exempelvis ha varit att svårare kod skrevs vissa veckor, eller att syftet för en del kod kunde ändras (vilket i sin tur leder till fler refaktoreringar). Därför ansåg gruppen just *andel teknisk skuld* som ett bristfälligt KPI eftersom det var så svårt att utvärdera. Det i sin tur ledde till att strategin inte kunde verifieras som välfungerande eller ej.

Att nå till det som gruppen ville uppnå i D1, dvs minimal teknisk skuld, kanske skulle kunna ha åstadkommits på annat sätt. Exempelvis skulle man kunna ha lagt mer tid på diskussion om hur varje klass och subsystem i koden skulle se ut (och kunna göra i framtiden innan den skrevs.

3.5.3 Nöjd kund



Figur 3: KPI-diagram: *Kundnöjdhet*

Från början var tanken med *Nöjd kund*-KPI:et att gruppen skulle bedöma hur nöjd kunden var genom en god kontakt med denne. På grund av problem med hur god kontakt gruppen kunde ha med kunden gick det inte att få ut bra data för KPI:et. Därför valde gruppen att utgå från våra avklarade poäng på *user stories* istället. KPI:et blev då i sig mindre formellt eftersom gruppen själv satte poäng utifrån user stories som skapats. Egentligen hade det inte mycket med kunden att göra, men det var nödvändigt då tiden med kunden i verkligheten var långt under förväntan.

Genom det förändrade KPI:et erhölls istället ett värde som *kan*, men inte behöver, ha en koppling med hur nöjd kunden är. I en perfekt värld kan det ursprungliga KPI:et ha fungerat bättre eftersom man då hade haft kontakt med kunden oftare. Alltså skilde sig den slutgiltiga versionen av detta KPI från den som först var tänkt till D1. Dock var tanken densamma; poäng skulle sättas utifrån antal klarade *user stories*. Den enda skillnaden var att gruppen satte poängen själva istället för att kunden gjorde det. Strategin som valdes för KPI:et (kundteam som sköter kontakt med kund) visade sig därför inte fungera som förväntat, eftersom kundkontakten var låg. Däremot var det bra att ha ett *kundteam* då de kunde agera kund utifrån tidigare erfarenheter med kunden. När det väl fanns kontaktmöjligheter var det även bra eftersom teamet passade på att ställa så många frågor som möjligt som hade diskuterats i grupp tidigare.

3.6 Relation mellan prototyp, process och värde för intressenter

Gruppen fokuserade tidigt på att få till en utförlig och välformulerad process. Detta ledde till att det skapades mindre värde för intressenter i det tidiga skedet. Senare under projektet hjälpte processen att göra det mer effektivt med arbetet på prototypen och på så vis skapa mer regelbundna värden för intressenter längre fram. Om gruppen inte skulle fokuserat på en ordentlig process i början så skulle det kunnat ha lett till flaskhalsar och sämre kod, då projektet inte hade gjorts ordentligt från början. I ett kort projekt som detta skulle det kunna leda till mer värde för intressenter, men även en prototyp som blir svårare att bygga ut. I ett långt projekt skulle den tiden som delegeras till administrativ planering vara proportionerligt mindre mot resten av arbetsområdena. Även finns det, i ett längre projekt, ett större krav på en god process.

Genom att kommunicera med kunden, eller en representant, en gång per sprint under *Sprint Review* kunde teamet lättare säkerställa att man levererade kundvärde. Det är dessutom lättare att ändra prototypens utformning och funktionalitet med regelbunden kontakt med produktägaren. Med en ännu mer deltagande produktägare, där teamet lättare kan kommunicera med produktägaren, ökar chanserna ytterligare att leverera något intressenter vill ha. Med vattenfallsmetoden finns det en risk att man har levererat något som inte ger värde åt intressenter, då man inte stämmer av med kunden regelbundet, och därför förbrukat tid och resurser i onödan.

I början av projektet lades mycket fokus på att leverera ett tydligt värde för kunden vid slutet av varje sprint. Utvecklingen fokuserade på lodräta inkrement. Efterhand visade det sig däremot att det fanns mycket i grundskiktet av mjukvaran som behövde undersökas, och i flera fall slutligen ersättas med nyskriven mjukvara. Fokus hamnade på att först fixa en stabilare grund, alltså vågräta inkrement. Utan denna bas skulle teamet inte ha möjlighet att leverera nya funktioner som har ett greppbart värde för kunden, och samtidigt göra något som inte ökar den totala arbetsbördan för mycket.

Fördelen med att bygga upp den här basen är att det underlättar för teamets framtida arbete. Nackdelen är att det kan vara svårare att visa upp något som är av värde för intressenter. Teamet skulle finna värde i att spendera en hel sprint på att utveckla exempelvis ramverk till framtida utveckling. Däremot skulle produktägaren och eventuella intressenter inte se något direkt värde i ett sådant vågrätt inkrement.

Processen är viktig även för resultatet av en *Minimum Valuable Product*, *MVP*. Med en agil process eftersträvar man att alltid ha en fungerande produkt som man sedan bygger ut funktionaliteten på. På så vis kan man lättare säkerställa att det finns en färdig prototyp oavsett när projektet avslutas. Följer man istället vattenfallsmetoden finns det en hög risk att om projektet plötsligt avslutas finns det inte en färdig produkt. Detta skulle då innebära att den tid och de pengar som

lades på den ickefärdiga produkten vore ett slöseri. På grund av möjligheten, vid agilt arbete, att anpassa sig till kundens ändrade önskemål, ökar det även chansen för att det är en relevant prototyp som framtagits. Detta då konstant konstruktiv kritik kan ges från produktägaren varje gång efter den har blivit presenterad med en fungerande prototyp.

Efter avslutningen av sprint sex presenterades detta projekts slutgiltiga *MVP*, D4, för produktägaren. Där gavs god kritik från en nöjd kund som uttryckte sin positiva reaktion på de prioriteringsval som gjordes (se avsnitt 3.7).

3.7 Prioriteringar

Under projektets gång fattades ett antal kritiska beslut gällande vad som skulle prioriteras. En lärdom från *LEGO*-övningen var att halvfärdiga hus inte har något värde för kunden, utan kunden vill ha hela hus även då det innebär färre sådana. I och med att kontakten med ordinarie produktägare under projektet blev oregelbunden kunde denne inte tillfrågas om vad som skulle prioriteras i händelse av tidsbrist. Besluten som fattades av gruppen baserades därför ofta på lärdomar från *LEGO*-övningen och liknande resonemang. På så sätt beslutades det att inte producera flera halvfärdiga komponenter, utan att istället fokusera på en komponent och arbeta för att göra den välfungerade och säker. Med detta som motivation beslutades det att nedprioritera kolonnkörning helt och hållet, och att istället utsågs en fungerande *Adaptive Cruise Control* (*ACC*) till huvudmål för projektet.

Ett annat problem som ställde till det för gruppen var det faktum att *MOPED*:en som skulle arbetas med i princip alltid hade trasiga sensorer. Då samtliga grupper bytte *MOPED* med varandra under ett rullande schema var det svårt att veta ifall nästa *MOPED* skulle vara fungerande eller ej. Under ett visst antal veckor var gruppen fast med en *MOPED* utan fungerande sensorer, och därmed fick det som tidigare bestämts under *Sprint Planning* förändras och omprioriteras. Under gruppens diskussionsmöten diskuterades det hur man skulle göra det bästa av situationen. Då detta problemet var genomgående under stora delar av projektet prioriterades exempelvis appen högt. Det gav gruppen en snabb, säker och stabil app att styra bilen ifrån. Andra sätt som problemet kunde lösas på var att diskutera med andra grupper hur deras implementation av funktionaliteten i fråga såg ut, eller fråga om de kunde dela med sig av sensordata.

Referenser

[1] L. Williams, "The collaborative software process", Utahs universitet, Avdelningen för Datavetenskap, Utah, Förenta staterna, 2000. URL: https://collaboration.csc.ncsu.edu/laurie/Papers/dissertation.pdf (hämtad 2017-10-20).

[2] K. Beck m. fl. (2001). "Manifest för Agil systemutveckling," *Manifest för Agil systemutveckling*, URL: http://agilemanifesto.org/iso/sv/manifesto.html (hämtad 2017-10-20).

[3] H. Kniberg, *Scrum and XP from the Trenches - 2nd Edition*, 2. utg. lulu.com, 2015, Sverige, ISBN: 1329224272. URL: https://www.infoq.com/minibooks/scrum-xp-from-the-trenches-2 (hämtad 2017-10-20).

[4] H. Eknefelt, *Configuration Management på fem minuter*, Volvo Cars, 2017. URL: https://github.com/hburden/DAT255/blob/master/Slides/L11-CM.pdf?raw=true (hämtad 2017-10-20).

[5] C. Frithiof, *Startup life*, 8 Dudes in a Garage, 2017. URL: https://github.com/hburden/DAT255/blob/master/Slides/L10-IGDB.pdf?raw=true (hämtad 2017-10-20).