C# Grupp 6

Agila metodkomponenter

Nicolas Björkefors, Patrik zetterblom, oskar olofsson, elias stagg, saleh bashir, muaz bahtiti, albin salmonson, dario borojevic, pernilla gerdin

2019

# Metodkomponenter

Vårt team har utsett en medlem, Pernilla, till *scrummaster* och det blir hennes ansvar att leda möten. Att ha en dedikerad person till att leda möten med kund och mellan teamet tillåter mer struktur än om alla hade kommit dit utan en plan, eller om alla har olika idéer om hur det ska gå till. Självklart kan andra medlemmar komma med input; de förväntas inte sitta tysta under mötena utan det är bara Scrum Masterns uppdrag att strukutera dem och se till att det går smidigt till. Men scrummastern ska inte styra laget, utan är en del av det (Cohn M 2004).

Korta *dagliga* *scrummöten* kommer hållas för att snabbt göra en statusuppdate om vad man åstadkommit, vad man ska göra, och om det finns några problem. Dessa problem ska inte lösas under mötet, utan det är bara till för att dela information.

Under *sprintplaneringsmöten* kommer teamet tillsammans med kund besluta om vilka user stories från product backloggen som ska utvecklas under kommande sprint. Det är väldigt viktigt att man har en korrekt prioriterad och tidsestimerad product backlog för att kunna göra detta. Men när det är gjort har teamet en tydlig plan för vad som ska göras den kommande veckan. Man kommer aldrig behöva undra vad som ska göras, eftersom vi har en prioriterad lista med funktioner som ska utvecklas.

Som nämndes ovan består *product* *backloggen* av user stories – detta är all funktionalitet systemet ska bestå av, nedbruten till hanterbara delar. Varje del ska vara prioriterad, samt tidsestimerad så teamet kan planera hur systemet ska utvecklas. *Sprint* *backlog* ska bestå av det teamet valt ut att arbeta under kommande sprint – i vårt fall ca en vecka. *Burn* *down* *chart* kommer innehålla vad vi åstadkommit i slutet av sprinten för att se om vår velocitet som lag stämmer överens med sprintens velocitet. Alltså om vi lyckades göra så många timmars arbete som vi föreutspådde. Om det inte går ihop måste vi se över vår tidsestimering – antingen hur lång tid det tar att utveckla funktioner, eller hur mycket arbete vårt team kan få gjort under en sprint.

Product baklockgen består som sagt *user* *stories*. Detta är funktioner i systemet beskrivna från användarens perspektiv. En simplifierad beskrivning som förklarar vad en användare ska kunna göra. User stories ska inte vara för stora, men samtidigt ska det inte brytas ned för mycket. En user story bör ta 4 till 16 timmar att utveckla (SOURCE?).

Under arbetet kommer vårt team utnyttja *parprogrammering* – att sitta och arbeta två-och-två – detta tillåter flera personer med olika perspektiv att jobba på samma sak. Detta främjar interaktion mellan individer och leder till bättre kod på kortare tid.

För att se till att vår kod hålls till en bra standard, och att vi inte får för många problem som behöver lösas som vi inte har tid med efter vi implementerat får sprint backlog kommer vi försöka utnyttja *slack*. Genom att planera in refaktorisering, debugging, och andra problem som kan uppstå i vår estimering ger vi oss själva lite utrymme utifall saker inte går som planerat. Refaktorisering är viktigt, och det är självklart bra att ha clean kod men det är inte lika tidskänsligt som vissa andra saker. Vi kan ta oss an det när vi har tid, eller inte är kapabla att jobba med annat.

Vi kommer att ha *collective code ownership*, delad kod i den bemärkelsen att hela teamet äger all kod tillsammans. Alla har tillgång till all kod och kan bidra och göra förändringar. Detta låter oss vara dynamiska och få olika personers expertis och perspektiv på alla delar av systemet.

För att komplemera shared code kommer vi köra en *whole team approach*. Detta innebär att teamet inser och jobbar utifrån förståelsen att man inte kan lyckas ensam. Man måste jobba tillsammans, och hela laget har ansvar för systemets kvalitet och framgång. Man stöttar varandra och täcker varandras svagheter; man måste vara redo att byta roller när det krävs.

Dessa komponenter främjar de fyra grundprinciperna på olika sätt. De utgör ett väldigt socialt sätt att arbeta på med många möten och arbeten i par. Detta lägger stort fokus på interaktion mellan teammedlämmar samt kund. Kundsamarbetet styrks också över kontraktförhandling i och med att kunden spelar en aktiv roll i utvecklingen. Att man delar upp arbetat på systemet till hanterbara delar i form av user stories och korta, iterativa sprintar främjar fungerande mjukvara snarare än utförlig dokumentation. Man måste vara dynamisk och hantera, eller i alla fall överväga, ändringar medan de dyker upp, och detta hjälps av ett modulärt upplägg, möjlighet att diskutera och dela information snabbt, samt möjligheten att avbryta en sprint för att rekonstruera den och börja igen.

# Kommunikation

Regelbunden kommunikation är det bästa sättet att bygga förtroende inom teams. (Henttonen and Blomqvist 2005). I rapporten från Pikkarainens et al (2008) så skriver de om en specifik studie som påvisar att agila praktiker kan medföra ett flertal positiva effekter för kommunikation. Det fanns ett flertal praktiker som visade sig vara gynnsamma när det gällde kommunikation. Vi kommer nu att gå in lite djupare på några av dessa praktiker.

Open Office space:  
Resultat av studien påvisade att när man använde sig av open-officepraktiken så minskade behovet av dokumentation. Den agila principen säger att man endast ska dokumentera efter behov och eftersom alla hade vetskap om statusen på projektet och vilka gemensamma mål som fanns kunde man således minska det administrativa arbetet. (Pikkarainen et al 2008)

Ett negativt aspekt med open-office är enligt rapporten att det potentiellt kan påverka koncentrationsförmågan och fokuset i gruppen. Enligt Pikkarainen et al (2008) rapport så medgav de flesta utvecklarna att open-office gjorde att det ibland var svårt att hålla koncentrationen uppe. Ett exempel på en invändning från en av utvecklarna var: Det fanns alltid tillfällen där andra människor höll i möten i det öppna kontorslandskapet vilket påverkade möjligheten att fokusera. (Utvecklare, Projekt 1).

Trots detta så utvärderades det som ett bättre sätt att organisera projekt.

Daily meetings:

Några positiva effekter som hittades var att det var ett bra sätt att hålla utvecklare, projektledare och ibland även kunder uppdaterade kring projektet. Det framgår också i rapporten att när man utvärderade de dagliga mötena så fann man de mer effektiva än de tidigare veckomötena. Pikkarainen et al (2008). En invändning till de dagliga mötena var att "Dagliga möten erbjuder inte nog med information för testande". (Testare, Projekt1)

Vår reflektion utifrån rapporten av Pikkarainen(2008) et al är att agila metoder om de används på rätt sätt är väldigt bra verktyg för att öka kommunikationen inom systemutvecklingsprojekt. Det är dock viktigt att man anpassar praktikerna utifrån sina förutsättningar inom arbetsgruppen/kontoret. Som Pikkarainen et al (2008) skriver i sin rapport så har det funnits problem med koncentrationsförmågan när man använt sig av open-office praktiker. Därför är det viktigt att visa hänsyn när man håller möten o.s.v. i närheten av andra människor som inte är direkt involverade i mötet. Vi anser också att dagliga möten kan vara väldigt användbart vid systemutveckling eftersom teamet får en bättre överblick av hur projektet fortskrider utöver sin (ofta lilla roll) inom projektets skapande.

# Planerad strategi

Vi planerar att lägga upp arbetet i stort sätt efter Scrums riktlinjer och principer. Vi har en scrummaster som kommer styra mötena och hjälpa gruppen framåt om vi kör fast. Dagliga scrummöten kommer tillgodose informationsdelande och statusuppdateringar. Under utvecklingen kommer vi utnyttja parprogrammering för att effektivt skriva och reviewa kod. Under sprintmöten kommer vi diskutera med kund och plocka ut de viktigaste user stories att arbeta på under kommande sprint.

Det finns manga likheter mellan Scrum och Extreme Programming. Båda två är agila tillvägagångssätt som används vid utveckling av diverse system och tillämpar då saker som man finner i alla agila metoder. Till exempel utnyttjar båda metoder backlogs, både product- och sprintbacklogs. Man prioriterar och tidsestimerar och jobbar iterativt, och leverar kontinuerligt. Det finns dock några distinkta skillnader, en av dessa skillnader är hur man hanterar sprintarna.

När man arbetar enligt scrum så får man inte göra några ändringar under en pågående sprint, medans XP-team har flexibiliteten att ändra innehållet under en pågående sprint. Sprintarnas längd är också en viktig skillnad där en sprint i scrum varar i 2-4 veckor medans en sprint i XP är betydligt kortare, där pågår en sprint i 1-2 veckor.

Inom scrum så finns det en så kallad Scrummaster som har lite mer ansvar för projektet och har bland annat i uppgift att hålla i möten som rör projektet. I XP så finns ingen Scrummaster utan det är kundens uppgift att hålla i mötena.

Kort och gott så skulle man kunna säga att scrum är ramverket som vi arbetar utefter medans XP specificerar hur vi arbetar.

Två komponenter vi valt att inte använda är Informativ Arbetsmiljö och Testa-först-programmering. Informativ Workspaces liknar information radiators (en stor skärm med viktig teaminformation som kontinuerligt uppdateras) från Crystal Clear och siktar på att förse snabb och effektiv feeback för teamet via enkelt tillgänglig information. Arbetsmiljön kan då innehålla saker som visuella displays, information radiators, eller saker som lavalampor eller signaller kopplade till processer. Dessa saker kan förse sammanhang och information till teamet (Melo Oliviera de R, Goldman A, O. Melo C 2012). Vi väljer att utesluta denna component eftersom det inte är rimligt eller värt mödan i vårt fall att ordna en sådan miljö. Vi har ingen fast plats vi kommer kunna mötas på kontinuerligt, och kan därför inte dra nytta av den. Detta betyder att vi möjligen får nöja oss med arbetsmiljöer där arbetsron eller andra omständigheter inte är optimala.

När det gäller Testa-först-programmering så är det klart vi kommer testa vår kod – det är oundvikligt. Men vi kommer inte ha möjlighet att använda oss av något ramverk för att testa vår kod. Att ha fördefinierade tester för att testa alla aspekter av systemet medan man utvecklar det är väldigt värdefullt och tidssparande men det är inte något vi har en förståelse för i dagsläget och det är inte värt vår knappa tid att försöka implementera det i detta project. Vi kommer helt enkelt få testa vår kod manuellt.

**References**

Cohn, M. (2004). User Stories Applied - For Agile Software Development.   
Addison-Wesley.

Melo Oliviera de R, Goldman A, O. Melo C (2012) Designing and Managing Agile Informative Workspaces: Discovering and Exploring Patterns.  
IEEE computer society, 4790-4791. DOI 10.1109/HICSS.2013.171

Pikkarainen M, Haikara J, Salo O, Abrahamsson P, Still J (2008).The impact of agile practices on communication in software development. Empirical Software Engineering, 13(3), 303-337. DOI:10.1007/s10664-008-9065-9